Dissertation

Separation Algorithms for Cutting Planes Based on Mixed Integer Row Relaxations

Implementation and Evaluation in the Context of Mixed Integer Programming Solver Software

von
Dipl.-Wirt.-Inf. Philipp M. Christophel

Schriftliche Arbeit zur Erlangung des akademischen Grades
doctor rerum politicarum (dr. rer. pol.)
im Fach Wirtschaftsinformatik

eingereicht an der
Fakultät für Wirtschaftswissenschaften der
Universität Paderborn

Promotionskommission:
Prof. Dr. Leena Suhl (1. Gutachter)
Prof. Dr. Laurence A. Wolsey (2. Gutachter)
Prof. Dr.-Ing. habil. Wilhelm Dangelmaier (3. Gutachter)
Prof. Dr. Uwe H. Suhl
Prof. Dr. Joachim Fischer

Paderborn, Juli 2009

Für Julia Bibiana

Danksagung

An dieser Stelle würde ich gerne die Gelegenheit nutzen, mich bei den vielen Menschen, die mich auf dem Weg zu dieser Dissertation begleitet und unterstützt haben, zu bedanken. Als erstes wäre da meine Betreuerin, Prof. Leena Suhl, die mir diesen Weg durch Verständnis, Vertrauen und Flexibilität enorm erleichtert hat. Prof. Uwe Suhl möchte ich dafür danken, dass er mich mit seiner Begeisterung für MIP Solver zu meinem Thema inspiriert und meine Arbeit in vielerlei Hinsicht unterstützt hat. Prof. Laurence Wolsey gilt mein Dank für einen sehr wertvollen Forschungsaufenthalt in Belgien und zahllose Anregungen, die meine Arbeit sehr bereichert haben.

Außerdem auf meinem Weg begleitet haben mich viele hilfsbereite Kollegen, sowohl am DS&OR Lab, als auch am CORE in Belgien. Ihnen danke ich vor allem für interessante Gespräche und aufmunternde Mittagspausen. Franz Wesselmann und allen an MOPS Beteiligten gilt mein Dank für ihre freundschaftliche und fachliche Unterstützung.

Bereits vor diesem nun abgeschlossenen Projekt haben meine Familie und besonders meine Eltern mich auf allen meinen Wegen begleitet. Auch dieses Mal waren sie an meiner Seite und haben mir mit Regenschirm und guten Ratschlägen beigestanden. Dafür danke ich ihnen aus der Tiefe meines Herzens.

Diese Arbeit widme ich meiner Verlobten Julia. Sie hat alle Höhen und Tiefen dieses Projekts mit mir durchlebt. Ohne ihre Aufmunterungen und ihre ausdauernde Unterstützung hätte ich diese Arbeit nicht zu Ende bringen können. Danke!

Contents

1. Introduction

Mixed integer programming (MIP) is a method for modeling and solving optimization problems. These optimization problems can originate from many different fields as, for example, from transportation or production planning. This is also pointed out by Bixby and Rothberg in [25], " . . . in the last few years MIP has become a vital capability that powers a wide range of applications in a variety of application domains". Similarly, Ashford states in [10] that "MIP is now widespread in decision-oriented applications across the whole industrial spectrum." These statements reflect the current situation which is that companies all over the world use modeling languages, like AMPL [7] and MPL [73], to model decision and planning problems using MIP. These models then are solved using MIP solver software packages to find (near) optimal solutions to their problems.

MIP solvers are developed and distributed by a number of companies of which the most visible are ILOG, recently acquired by IBM, with its MIP solver CPLEX [54] and Dash Optimization, part of Fair Isaac, with its MIP solver Xpress-MP [39]. Besides these two well-known companies a number of other companies offer MIP solvers, among these the MOPS GmbH & Co. KG with its MIP solver MOPS [76], that is described in section 3.3 of this thesis.

Besides these commercial MIP solvers a number of academic and open-source projects exist. MINTO [3] is an academic MIP solver used in many scientific publications. SCIP [4] is also an academic solver for MIP problems with the specific feature that it combines techniques from MIP and constraint programming. The *computational infrastructure for operations research (COIN) project* [1] hosts a number of open-source projects dealing with MIP including the MIP solvers CBC and SYMPHONY. Another notable open-source MIP solver is GLPK [2].

An important feature of modern MIP solvers are *cutting planes* (or *cuts* for short). MIP solvers typically generate a number of different types of cuts. This thesis deals with a group of *cut generators* used in all modern MIP solvers. This group of cut generators is identified by the fact that they generate cuts from the mixed integer rows, i.e. the constraint matrix, of the MIP problems. This is in contrast to those cut generators that use the linear programming (LP) simplex tableau.

The situation in the MIP literature is that many publications deal with theoretical results about cuts instead of the implementation aspects of cut generators. A famous example from the past are the Gomory mixed integer cuts that, although theoretically well known since the 1960s, only in the 1990s were used in a way that they dramatically improved the performance of MIP solvers (see [31]). Although the situation has changed a lot since then, there are still not many publications about the implementation of cut generators. One reason for this is that the drivers of MIP technology, the large MIP solver companies, obviously do not have an interest in publishing details about their implementations. A reason why researchers are not motivated to publish about implementation details might be that implementation-based publications are probably not valued as much as theoretical results. This might change in the future, as the Mathematical Programming Society started the new journal *Mathematical Programming Computations*[1].

One important goal of this thesis is to give a detailed description of the implementation for three widely used cut generators, namely those for flow cover, flow path and complemented mixed integer rounding (cMIR) cuts. A second, even more important goal, is to drive the development of these cut generators to improve upon the methods described in current publications. As we see in chapter 5, the flow path cut generator does not leave much room for improvement. Therefore we also introduce two new cut generators that can be seen as potential substitutes for the flow path cut generator. Finally, an important goal of this thesis is to computationally evaluate the mentioned cut generators to allow conclusions about which cut generators are really needed by an MIP solver. This includes finding a default configuration of cut generators that leads to a good overall performance of an MIP solver.

In this thesis we start by discussing the theory of MIP (chapter 2) and MIP solver software (chapter 3). Then we give a review of the currently used mixed integer row relaxation-based cut generators in chapter 4. These are the cut generators for flow cover, flow path and cMIR cuts. Chapter 5 constitutes the main result of this thesis as it describes implementations and algorithmic improvements for these three important parts of modern MIP solvers. Additionally, two new cut generators for a new class of cuts, the path mixing cuts, are introduced. The presentation of the implementations of all cut generators is done in pseudo-code. The pseudo-code is meant to be detailed enough to give insights into the program flow but simple enough to be easily understandable. The presented cut generators then are computationally evaluated in chapter 6. These results are a second important contribution of this thesis, as the results indicate which implementation details and algorithmic improvements lead to a better performance of

[1] see `http://www.isye.gatech.edu/~wcook/mpc/index.html`

the cut generators. They also show which configuration of the described cut generators results in the best performance for an MIP solver. Finally, chapter 7 summarizes the results of this thesis and gives an outlook on future research opportunities.

2. MIP Theory

2.1. Mixed Integer Programming Problems

In this chapter we give an overview of theoretical mixed integer programming (MIP) results relevant for this thesis. We try to keep things simple. For a more in-depth treatment of the topics in this chapter we refer the reader to standard MIP textbooks like [98], [80], and [85].

This thesis deals with *MIP problems*. MIP problems are optimization problems of the form

$$\max cx + hy \tag{2.1}$$

$$Ax + Gy \leq b \tag{2.2}$$

$$l \leq x \leq u, \quad l' \leq y \leq u' \tag{2.3}$$

$$x \in \mathbb{R}^n, \quad y \in \mathbb{Z}^p. \tag{2.4}$$

where A and G are matrices of appropriate size with elements in $\mathbb{R}$ and c, h, b, l, l', u and u' are vectors with elements in $\mathbb{R}$. These problems consist of a linear *objective function* (2.1), linear *constraints* (2.2), *lower and upper bounds* (2.3), and *integrality conditions* (2.4). Throughout this thesis we use x or s for continuous and y for integer variables. Sometimes we specifically want to differentiate between *general integer variables*, i.e. variables with arbitrary lower and upper bounds, and *binary variables*, i.e. integer variables with a lower bound of 0 and an upper bound of 1. Then we use z for general integer variables and y for binary variables. Note that an MIP problem can also have equality and/or greater than or equal inequalities. These, as well as the bounds (2.3), can also be written in the form of the constraints (2.2) if needed. Concerning lower and upper bounds, note that these can also be infinite.

The matrices A and G together form the constraint matrix (A, G) of the problem. As the *columns* of this matrix represent the variables of the problem we use columns as an equivalent for variables. In the same way we use *rows* as an equivalent for constraints.

The constraint matrix together with the integrality conditions describes a *mixed integer set* or *set of feasible solutions* X:

$$X = \{(x, y) : (x, y) \in \mathbb{R}^n \times \mathbb{Z}^p, Ax + Gy \leq b, l \leq x \leq u, l' \leq y \leq u'\}$$

If we replace the objective function by $c(x, y)$ it is therefore possible to state an MIP problem as

$$z = \min\{c(x, y) : (x, y) \in X \subseteq \mathbb{R}^n \times \mathbb{Z}^p\}$$

where here z is an optimal solution to the optimization problem.

We now give examples for two classes of MIP problems. The first example is the class of *lot-sizing problems*. In example 2.1 we present the most simple lot-sizing problem: the single-item, single-level, uncapacitated lot-sizing problem. For further reading on more complex lot-sizing problems and their practical use we refer to [85]. Secondly, we give an example of a problem instance from the class of fixed-charge network design problems (see example 2.2).

Example 2.1. *Assume we want to plan the amount of some good a factory produces over a number of periods* $1 \ldots n$*. The production cost of one unit of the good is based on some fixed cost* h_j *for producing in a period and a variable cost* c_j *for each unit produced. Furthermore it is possible to store units to use them in a later period for storage costs* p_j*. Using the produced units and units from storage a certain demand* d_j *has to be satisfied in each period. In other words, the task at hand is to find the optimal size of the production lots for the periods subject to fixed costs, per item costs and storage costs. This class of problems is called the uncapacitated lot-sizing problem and can be modeled as an MIP problem in the following way:*

$$\min \sum_{j=1}^{n} (c_j x_j + h_j y_j + p_j s_j)$$

$$\begin{aligned} s_{j-1} + x_j - s_j &= d_j && \text{for } j = 1 \ldots n \\ x_j &\leq M y_j && \text{for } j = 1 \ldots n \\ s_0 = s_n &= 0 \\ x \in \mathbb{R}^n_+, s \in \mathbb{R}^{n+1}_+&, y \in \{0, 1\}^n \end{aligned}$$

where M *is a large enough number. The* x_j *variables specify the amount produced in the production periods. The binary variables* y_j *indicate whether in a certain period* j *something is produced or not. The* s_j *variables indicate the amount of units stored at the end of each period* j*.*

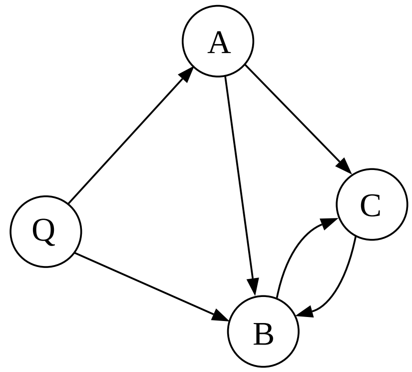

Figure 2.1.: Network structure for example 2.2.

Example 2.2. *Assume we want to plan the water supply of a housing estate. Figure 2.1 shows the pump station (node Q), three areas where houses need supply of water (nodes A, B, and C) and possible water pipes we can build (directed edges in the graph). Building a pipe allows us to send a certain maximal amount of water through the pipe in the direction of the edge. Associated with building a pipe is some building cost. Finding the cost optimal set of pipes to build such that all housing areas get enough water can be modeled as a* fixed-charge network design problem*. An MIP problem instance with some input parameters such as costs, demand, supply and limits on the throughput of the pipes looks as follows:*

$$
\begin{array}{lrl}
\min & 10y_{QA}+12y_{QB}+12y_{AB}+11y_{AC}+9y_{BC}+8y_{CB} & \\
& -x_{QA}-x_{QB} & =-100 \\
& x_{QA}-x_{AB}-x_{AC} & = 30 \\
& x_{QB}+x_{AB}-x_{BC}+x_{CB} & = 30 \\
& x_{AC}+x_{BC}-x_{CB} & = 40 \\
& x_{QA}-50y_{QA} & \leq 0 \\
& x_{QB}-70y_{QB} & \leq 0 \\
& x_{AB}-70y_{AB} & \leq 0 \\
& x_{AC}-30y_{AC} & \leq 0 \\
& x_{BC}-60y_{BC} & \leq 0 \\
& x_{CB}-40y_{CB} & \leq 0 \\
& x \in \mathbb{R}^6_+ \quad y \in \{0,1\}^6 &
\end{array}
$$

For each node we have a so called flow balance constraint (the first four equality constraints) that ensures that the demand is satisfied. For each edge we have a variable upper bound constraint (the last six constraints) that ensures that if the flow on an edge x_j is larger than zero the corresponding binary set-up variables y_j is one. These binary variables then imply costs in the objective function.

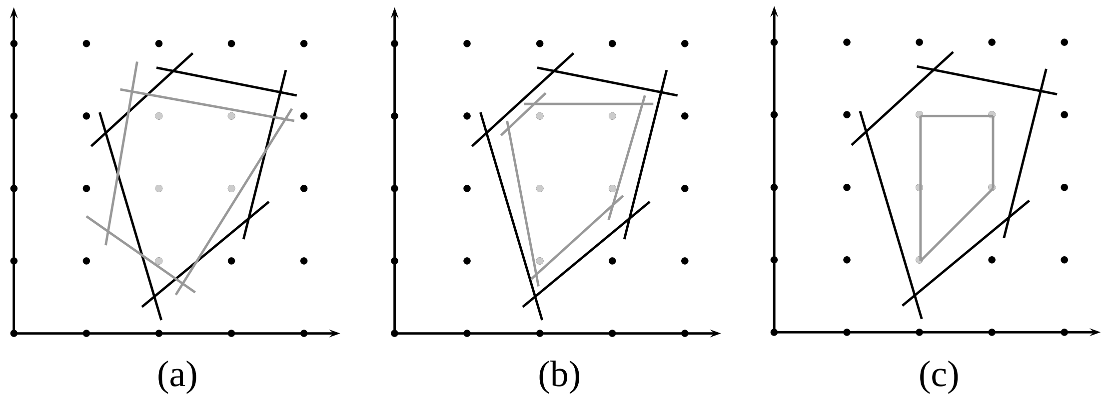

Figure 2.2.: Graphical representation of different formulations for the same set of feasible solutions (grey dots) of a MIP problem with two integer variables.

2.2. Formulations

Typically, an MIP problem can be modeled in more than one way. Each way to model a certain MIP problem with the exact same set of feasible solutions is called a *formulation*. To be more precise we use two definitions from [98].

Definition 2.1. *A subset of* $\mathbb{R}^n$ *described by a finite set of linear constraints* $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ *is a* polyhedron.

Definition 2.2. *A polyhedron* $P \subseteq \mathbb{R}^{n+p}$ *is a* formulation *for a set* $X \subseteq \mathbb{R}^n \times \mathbb{Z}^p$ *if and only if* $X = P \cap (\mathbb{R}^n \times \mathbb{Z}^p)$.

In figure 2.2a we give a visual representation of two different formulations for the same MIP problem with two integer variables. Knowing that there are several formulations for the same optimization problem it is natural to ask whether a formulation is better than another one. The graphical way of answering this question is to say that a formulation P_1 is better than another formulation P_2 if P_1 *lies within* P_2 as shown in figure 2.2b. Note that in 2.2a none of the two formulations is clearly better than the other one. This can be stated more precisely by a definition from [98].

Definition 2.3. *Given a set* $X \subseteq \mathbb{R}^n$*, and two formulations* P_1 *and* P_2 *for* X*,* P_1 *is a* better formulation *than* P_2 *if* $P_1 \subset P_2$.

Following up on this the next natural question to ask is whether there exists a *best formulation*. Again this question can be investigated graphically and it is obvious that the *smallest* formulation possible is one where all vertices of the polyhedron are integer points from the set of feasible solutions. This formulation is called the *convex hull* of X, denoted as $conv(X)$, and it is shown in figure 2.2c. We also give the formal definition from [98].

Definition 2.4. *Given a set* $X \subseteq \mathbb{R}^n$*, the* convex hull of X*, denoted by* $conv(X)$*, is defined as:* $conv(X) = \{x : x = \sum_{i=1}^{t} \lambda_i x^i, \sum_{i=1}^{t} \lambda_i = 1, \lambda \geq 0 \text{ for } i = 1, \ldots, t$ *over all finite subsets* $\{x^1, \ldots, x^t\}$ of $X\}$.

In principle there are two ways to improve the formulation of an MIP problem. The first is to use an *extended formulation*, we repeat the informal definition of an extended formulation from [85].

Definition 2.5. *An* extended formulation *for the set of feasible solutions of an MIP problem is a formulation involving new (and usually more) variables.*

In example 2.3 we show how such an extended formulation might look like. Much more about extended formulations can be found in [85].

Example 2.3. *In this example we want to show the well-known (see for example [98] and [85]) extended formulation for the uncapacitated lot-sizing problem introduced in example 2.1. As mentioned above, the idea of an extended formulation is to add new variables. In the case of the uncapacitated lot-sizing problems these new variables* w_{ij} *represent how many units of the product produced in period* i *are used to satisfy the demand in period* j*. We use the same* y_j *variables as before and can formulate the problem as follows:*

$$
\begin{aligned}
\min \sum_{j=1}^{n} \sum_{i=1}^{j} (c_i + \sum_{t=i}^{j-1} p_t) w_{ij} + \sum_{j=1}^{n} h_j y_j & & \\
\sum_{i=1}^{j} w_{ij} &= d_j & \text{for } j = 1 \ldots n \\
w_{ij} &\leq d_j y_i & \text{for } j = 1 \ldots n, i = 1 \ldots j \\
x_i &= \sum_{j=i}^{n} w_{ij} & \text{for } i = 1 \ldots n \\
w_{ij} &\in \mathbb{R}_+ & \text{for } j = 1 \ldots n, i = 1 \ldots j \\
x \in \mathbb{R}_+^n, y &\in \{0,1\}^n. &
\end{aligned}
$$

Note that we do not need the storage variables s_j *anymore but that we have to include their cost coefficients in the computation of the costs for the* w_{ij} *variables. We could* *also drop the variables x_j, $j = 1 \dots n$, and compute the values for them after we solved the problem. It can be shown that this formulation describes the convex hull of feasible solutions for this problem and thus is the best possible formulation.*

Typically finding extended formulations for an MIP problem is left to the modeler. The advantage of the second approach to improve the formulation of an MIP problem is that it can be done automatically. Instead of adding variables this approach adds constraints. Example 2.4 illustrates it and section 2.4 goes into the details.

Example 2.4. *We use the fixed-charge network design problem from example 2.2 and improve its formulation by adding an additional constraint. From the network structure in figure 2.1 we see that the demand of node A has to be satisfied by water running through the edge Q–A because no other edge leads into node A and the flow of the water can not be negative. i.e.* $x_j \in \mathbb{R}_+$. *In other words, the pipe from Q to A has to be build and thus* $y_{QA} = 1$ *in all feasible solutions. So the improved formulation* (P_2)*, consisting of the original rows of the formulation* (P_1) *together with* $y_{QA} \geq 1$*, is another formulation for the problem. It is an improved formulation because* $P_2 \subseteq P_1$*, as adding an additional constraint can not make* P_2 *larger, and* $P_2 \subset P_1$ *because for the point*

$$(x^*, y^*) = (30, 70, 0, 0, 40, 0, \frac{3}{5}, 1, 0, 0, \frac{2}{3}, 0)$$

it holds that $(x^*, y^*) \in P_1$ *and* $(x^*, y^*) \notin P_2$.

2.3. Relaxations and Bounds

For this thesis *relaxations* of MIP problems play a very important role. Informally it can be said that an MIP problem is contained in its relaxation, i.e. the set of feasible solutions of a relaxation is larger and the objective function value of the relaxation is better or equal for all feasible solutions of the original problem. More precisely this can be stated in the following definition (similar to [98]):

Definition 2.6. *A problem (RP)* $z^R = \min\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ *is a* relaxation *of (MIP)* $z = \min\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ *if:*

1. $X \subseteq T$, *and*
2. $f(x) \leq c(x)$ *for all* $x \in X$.

We now mention two ways in which relaxations can be used in MIP. Firstly, *valid inequalities* for a relaxation are also valid inequalities for the original MIP problem. This is elaborated in section 2.4. Secondly, the optimal solution of a relaxation forms a *dual bound* $\underline{z}$ on the optimal solution of the original MIP problem. A dual bound tells us how *good*, in terms of the objective funtion, an optimal solution can be at best. On the other hand each feasible solution to the problem $\bar{x} \in X$ gives us a *primal bound*, i.e. we know that a solution at least as good as this primal bound exists. So relaxations together with feasible solutions can help us to define in which interval of the objective function the optimal solutions to a problem lie. This is used in the branch-and-cut algorithm discussed in section 2.8.

A very important relaxation used in MIP is the *LP relaxation* of an MIP problem. It is obtained by dropping the integrality conditions on the integer variables. The result is a linear programming (LP) problem that is normally much easier to solve. If the optimal solution of an LP relaxation is an integer feasible solution (it is *integer*) then this solution is an optimal solution to the MIP problem. Note that if the formulation of an MIP problem describes the convex hull of the feasible solutions, solving the LP relaxation always results in a feasible solution and thus an optimal one. For more information about LP see for example the standard textbook [29].

Another way of relaxing an MIP problem is to drop constraints of the constraint matrix. This also means that each row of the constraint matrix alone forms a relaxation of the original problem. If we have such a single row relaxation of the problem we can also relax it further by removing some of the variables in the constraint as long as the constraint gets *less tight*. That means we can remove variables with positive coefficients in $\leq$-constraints and variables with negative coefficients in $\geq$-constraint. Removing variables in an equality constraint results in converting the constraint to an inequality constraint.

2.4. Valid Inequalities and Separation

As shown in example 2.4, formulations can be improved by adding certain constraints. These constraints have to satisfy two conditions. After adding them, the new constraint matrix still has to be a formulation for the problem. That means, the new constraint is not allowed to exclude any feasible solution to the MIP problem. On the other hand, in order to improve the formulation, it is necessary to exclude non-feasible solutions from the formulation. Constraints that satisfy the first of these conditions are called *valid inequalities*, we define them similar to [85].

Definition 2.7. *A* valid inequality *for a set of feasible solutions of an MIP problem X is a constraint of the form $\alpha x + \beta y \leq \gamma$ satisfied by all points in X; that is,*

$$\alpha\bar{x} + \beta\bar{y} \leq \gamma \text{ for all } (\bar{x}, \bar{y}) \in X.$$

Valid inequalities can be found by several different methods. A typical approach is to inspect a simple MIP problem and derive a set of valid inequalities called a *family of valid inequalities*. Note that any valid inequality for a relaxation of an MIP problem is also a valid inequality for the MIP problem itself (see [85]).

To improve a formulation, valid inequalities have to exclude, or in other words cut off, non-feasible solutions. To find these valid inequalities we define the *separation problem* (similar to [85]).

Definition 2.8. *Given a set of feasible solutions to an MIP problem X, a formulation P_X for X, and a family of valid inequalities $\mathcal{F}$, the* separation problem *for a given point $(x^*, y^*) \in P_X$ is to*

1. *either prove that there is no valid inequality in $\mathcal{F}$ that cuts off (x^*, y^*)*
2. *or to find a valid inequality $\alpha x + \beta y \leq \gamma$ from $\mathcal{F}$ that cuts off (x^*, y^*), i.e. where $\alpha x^* + \beta y^* > \gamma$.*

An algorithm that tries to solve a separation problem is called a *separation algorithm*. Separation algorithms come in two flavors, *exact* and *heuristic*. An exact separation algorithm guarantees to solve the separation problem, a heuristic does not.

If a separation algorithm is *successful* it returns a valid inequality that cuts off a certain point of the formulation. We call such a valid inequality a *cut*, *cutting plane* or *violated* valid inequality. The point to cut off in a separation problem is typically a solution to the LP relaxation of the problem. By repeatedly solving the LP relaxation, adding a cut, and resolving the LP relaxation with the improved formulation it is sometimes possible to solve MIP problems, depending on the problem instance and the family of valid inequalities used. Algorithms based on this idea are called *cutting plane algorithms*. Cutting plane algorithms on their own are considered unusable for solving most real world MIP problems, but in combination with branch-and-bound this approach is very successful. We discuss the resulting branch-and-cut algorithm in section 2.8.

Typically, families of valid inequalities contain many valid inequalities and even many cuts for the same point. It therefore is a natural question which of these cuts are the *best*.

One answer to this question is that the valid inequalities that are necessary to define the convex hull of X are not dominated by any other valid inequalities and hence are the best we can get. These valid inequalities are called *facet-defining*, we define them as in [85].

Definition 2.9. *A* facet-defining valid inequality *for X is a valid inequality that is necessary in a description of the polyhedron conv(X).*

For further reading about valid inequalities we refer to the aforementioned textbooks and to a recent tutorial on the theory of valid inequalities by Cornuéjols [32].

2.5. Mixed Integer Rounding Inequalities

In this section we describe mixed integer rounding (MIR), an approach for finding valid inequalities for MIP problems. It originates in [79], but the presentation here is from [98]. The idea is to first look at the simple MIR set $X^{\geq}$.

Definition 2.10. *The* simple MIR set *is defined as*

$$X^{\geq} = \{(x, y) \in \mathbb{R}^1_+ \times \mathbb{Z}^1 : x + y \geq b\}.$$

Note that in the definition of the simple MIR set we require $x \geq 0$, but y is not restricted in this way. For the simple MIR set the only non-trivial facet of the convex hull is described by the *simple MIR inequality* (from [98]).

Proposition 2.1. *The* simple MIR inequality

$$x \geq f(\lceil b \rceil - y),$$

where $f = b - \lfloor b \rfloor$, is valid for the simple MIR set $X^{\geq}$.

Proof. If $y \geq \lceil b \rceil$, then $x \geq 0 \geq f(\lceil b \rceil - y)$. If $y < \lceil b \rceil$, then

$$\begin{aligned} x &\geq b - y = f + (\lfloor b \rfloor - y) \\ &\geq f + f(\lfloor b \rfloor - y), \text{ as } \lfloor b \rfloor - y \geq 0 \text{ and } f < 1 \\ &= f(\lceil b \rceil - y) \end{aligned}$$

□

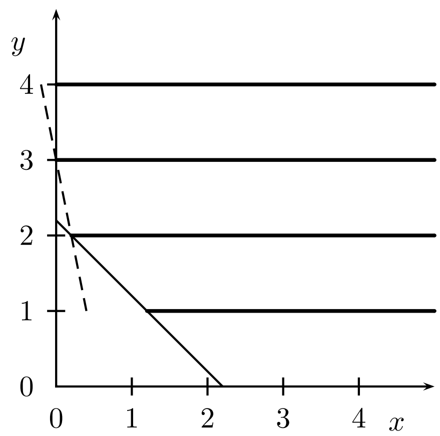

Figure 2.3.: Graphical representation of the simple MIR set and the simple MIR inequality from example 2.5.

Example 2.5 illustrates the simple MIR set and the simple MIR inequality. In section 4.2 of this thesis we show how the simple MIR inequality can be used to obtain valid inequalities for more complex sets.

Example 2.5. *Assume the simple MIR set*

$$x + y \geq 2.2$$

that is also shown in figure 2.3. In this figure the thick black lines are the feasible solutions of the MIP set, the thin black line is the constraint and the dashed line is the simple MIR cut

$$x \geq 0.2(3 - y).$$

In some situations it is appropriate to start with a slightly different simple MIR set that consists of a less than or equal inequality. For this set the corresponding simple MIR inequality is given in proposition 2.2, that, as well as its proof, is also from [98].

Proposition 2.2. *For the MIP set*

$$X^{\leq} = \{(x, y) \in \mathbb{R}^1_+ \times \mathbb{Z}^1 : y \leq b + x\}$$

the inequality

$$y \leq \lfloor b \rfloor + \frac{x}{1 - f},$$

where $f = b - \lfloor b \rfloor$, is a valid inequality.

Proof. Rewrite $y \leq b + x$ as $x - y \geq -b$ by multiplying with -1. Now observe that $-b - \lfloor -b \rfloor = 1 - f$. Using the simple MIR inequality we get $x \geq (1-f)(\lceil -b \rceil + y)$. As $\lceil -b \rceil = -\lfloor b \rfloor$ this results in the desired valid inequality. □

2.6. Mixing Inequalities

Another simple MIP set that has been studied to obtain valid inequalities for MIP problems is the *mixing set*. Günlük and Pochet defined it in [52], the presentation here is based on [85].

Definition 2.11. *The* mixing set *is the MIP set*

$$X_K^{MIX} = \{(x, y) \in \mathbb{R}_+^1 \times \mathbb{Z}^K : x + y_k \geq b_k \text{ for } 1 \leq k \leq K\}.$$

Note that the mixing set consists of K simple MIR sets with the same continuous variable x. The only non-trivial valid inequalities needed to describe the convex hull of a mixing set are the *mixing inequalities* (from [52]).

Proposition 2.3. *Let $T \subseteq \{1, \ldots, K\}$ with $|T| = t$, $f_k = b_k - \lfloor b_k \rfloor$, $k = 1 \ldots K$, and suppose that $i_1, \ldots, i_t$ is an ordering of T such that $0 = f_{i_0} \leq f_{i_1} \leq \cdots \leq f_{i_t} < 1$. Then the* mixing inequalities

$$x \geq \sum_{\tau=1}^{t} (f_{i_\tau} - f_{i_{\tau-1}})(\lceil b_{i_\tau} \rceil - y_{i_\tau}) \tag{2.5}$$

and

$$x \geq \sum_{\tau=1}^{t} (f_{i_\tau} - f_{i_{\tau-1}})(\lceil b_{i_\tau} \rceil - y_{i_\tau}) + (1 - f_{i_t})(\lfloor b_{i_1} \rfloor - y_{i_1}) \tag{2.6}$$

are valid for the mixing set X_K^{MIX}.

The most violated cut based on the mixing inequalities (2.5) and (2.6) for the mixing set X_K^{MIX} can be found efficiently using the following exact separation algorithm described in [85]. Reorder the rows $k = 1, \ldots, K$ such that $f_1 \leq \cdots \leq f_K$. For each row define $\beta_k = \lceil b_k \rceil - y_k^*$. Starting with the row which has the largest $\beta_i = \bar{\beta}$, iteratively add the next row i in the ordering as long as $\beta_i > (\bar{\beta} - 1)$ and $\beta_i > 0$. If $\bar{\beta} > 1$, use inequality (2.6), else use inequality (2.5). In example 2.6 we demonstrate this algorithm and show the resulting mixing inequality.

Example 2.6. *We assume the mixing set*

$$x + y_1 \geq 2.3$$
$$x + y_2 \geq 1.7$$
$$x + y_3 \geq 4.1$$

and the point

$$(x^*, y^*) = (2.3, 0, 0, 1.8).$$

This results in $\beta_1 = 3$, $\beta_2 = 2$, *and* $\beta_3 = 3.2 = \bar{\beta}$. *As* $\bar{\beta} > 1$ *we use inequality* (2.6) *and start with the third row of the mixing set that results in*

$$x \geq 0.1(5 - y_3).$$

As $\beta_1 > \bar{\beta} - 1$ *we now add the first row of the mixing set and get*

$$x \geq 0.1(5 - y_3) + (0.3 - 0.1)(3 - y_1).$$

We do not add the second row of the mixing set, because $\beta_2 \leq \bar{\beta} - 1$. *But as we use the second mixing inequality* (2.6) *we add one more term and get*

$$x \geq 0.1(5 - y_3) + (0.3 - 0.1)(3 - y_1) + (1 - 0.3)(4 - y_3)$$

or

$$x + 0.8y_3 + 0.2y_1 \geq 3.9.$$

This mixing cut is violated by the point (x^*, y^*).

The mixing set and the mixing inequalities are important because they can be used to derive valid inequalities for a number of MIP problem classes, foremost lot-sizing problems. Besides the mixing set presented here several variants of it are mentioned in [85], like the continuous mixing set (p. 249) and the divisible mixing set (p. 253). These variants of the mixing set also lead to valid inequalities for several lot-sizing problem variants.

2.7. Lifting Valid Inequalities

In this section we give a very short and simplified description of what is meant by *lifting* valid inequalities. For the details we refer to [80] and [66]. The presentation here is based on [66].

We first define MIP sets $Z^k(b)$ with the following structure

$$\begin{aligned} \sum_{t=1}^{k} G^t y^t &\leq b + x \\ y^t &\in X^t \text{ for } t = 1, \dots, k \\ x &\in \mathbb{R}^n_+ \end{aligned}$$

Lifting basically means making valid inequalities for a lower-dimensional subset, in this case $Z^1(b)$, valid for higher-dimensional MIP sets, in this case $Z^k(b)$, $k = 2, \dots K$.

One way to do lifting is to follow the sequential lifting approach consisting of the following steps:

1. Fix $y^t = 0$ for all $t = 2, \dots, K$.
2. Find a tight valid inequality $\beta^1 y^1 \leq \gamma^1 + \alpha x$ for $Z^1(b)$.
3. Iterations $k = 2, \dots, K$. Given a tight valid inequality $\sum_{t=1}^{k-1} \beta^t y^t \leq \gamma + \alpha x$ for $Z^{k-1}(b)$, lift the variables y^k and derive coefficients β^k such that

$$\sum_{t=1}^{k-1} \beta^t y^t + \beta^k y^k \leq \gamma + \alpha x$$

   is valid for $Z^k(b)$.

To determine the cut coefficients in step 3 typically an optimization problem has to be solved. In the approach as outlined above, the cut coefficients are computed sequentially and the actually computed coefficients depend on the ordering of this sequence. Fortunately, it is also possible to lift variables *sequence independent* using so called *superadditive lifting functions*. For this thesis it is sufficient to know that using a superadditive lifting function it is possible to compute each cut coefficient by computing a function value. This makes lifting computationally interesting.

2.8. The Branch-and-cut Algorithm

The branch-and-cut algorithm is a combination of the branch-and-bound algorithm and the idea of cutting plane algorithms mentioned in section 2.4. It currently is the typically used method for solving MIP problems to optimality.

The idea of the LP relaxation-based branch-and-bound algorithm, as first described by Land and Doig [61], is to do an implicit enumeration of the feasible solutions of the MIP problem. It starts with the solution of the initial LP relaxation of the problem. This problem is called the *root node*. If all integer variables take integer values in the solution of the root node, the algorithm is done, because the solution is also an optimal solution to the MIP problem. If not, it *branches*. Branching in this context means that it generates two subproblems out of the previous problem and investigates what the best possible solution to these subproblems is. These subproblems are called *nodes* and are added to a list of nodes. The most simple type of branching is to introduce bounds on one of the integer variables. More elaborate branching schemes involve using sets of variables, like special ordered sets (SOS), or branching on constraints.

For each node it is checked whether it can be *pruned*. For a node that can be pruned, no branching has to be done and thus it does not add new nodes to the list of nodes. A node can be pruned for three reasons:

1. *prune by optimality*,
2. *prune by infeasibility*, and
3. *prune by bound*.

A node can be pruned by optimality if in the solution of the LP relaxation of the underlying subproblem all integer variables take integer values. If this solution is the first integer solution found or if it is better than the best integer solution found so far it is stored and called the *incumbent*. Nodes with integer solutions can be pruned because from the theory of relaxations we know that there is no better solution in any of the subproblems of the node.

A node can be pruned by infeasibility if the LP relaxation of the underlying subproblem is infeasible. The reason for this is that if a problem is infeasible, all of its subproblems are infeasible too.

A node can be pruned by bound if the objective function value of the LP relaxation is worse than the one of the incumbent. The reason for this is that we know that all subproblems of such a problem will have a worse objective function value too, and thus, that among these we can not find a better integer solution than the incumbent.

If a node can not be pruned we have to branch and add two new nodes to the list of nodes. In the next iteration we have to choose a new node we want to investigate. This node is removed from the node list. Then we solve its LP relaxation, decide whether we

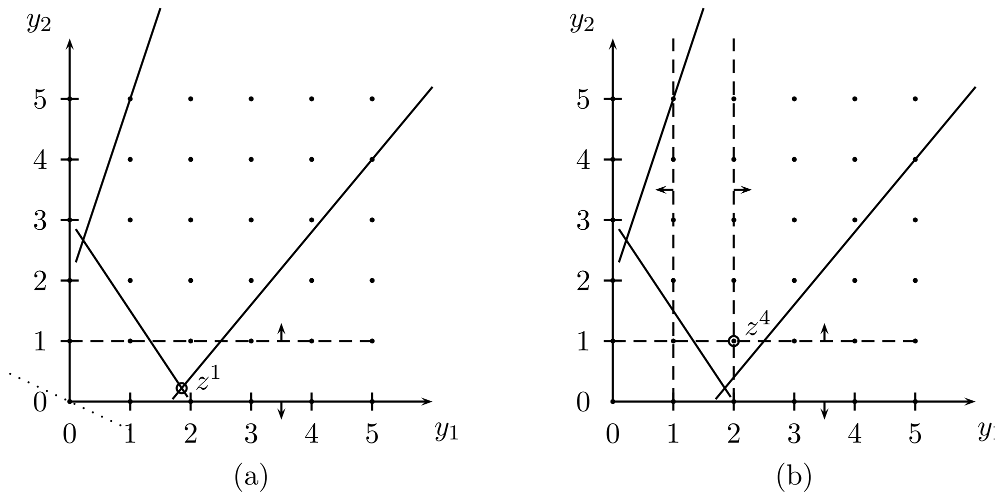

Figure 2.4.: The MIP problem from example 2.7. On the left side (a), z^1 marks the solution to the LP relaxation of the root node and on the right side (b), z^4 marks the optimal MIP solution.

can prune it, and then eventually branch again. If the node list is empty, it is proven that the current incumbent is an optimal solution to the MIP problem. An overview of the algorithm is given in figure 2.5a and we further illustrate it using example 2.7.

Example 2.7. *Assume the following MIP problem:*

$$
\begin{aligned}
z = \min 7y_1 + 15y_2 \\
1\frac{1}{2}y_1 + y_2 \geq 3 \\
3y_1 - y_2 \geq -2 \\
-1\frac{1}{5}y_1 + y_2 \geq -2 \\
y \in \mathbb{Z}_+^2.
\end{aligned}
$$

To solve this problem we first solve the LP relaxation and get the solution $y^1 = (1\frac{23}{27}, \frac{2}{9})$ *with objective function value* $z^1 = 16\frac{8}{27}$. *We then decide to branch using the variable* y_2. *The first subproblem we create gets the additional constraint* $y_2 \leq 0$ *and the second the additional constraint* $y_2 \geq 1$. *This branching is shown in figure 2.4a. Then we choose the first subproblem and solve its LP relaxation. As it is infeasible we can prune this node by infeasibility. The second subproblem is the only one in the node list so we choose to investigate it next. Solving the LP relaxation leads to the solution* $y^3 = (1\frac{1}{3}, 1)$ *with* $z^3 = 24\frac{1}{3}$. *As this node can not be pruned we branch again, this time using the variable* y_1. *This leads to the first subproblem where* $y_1 \geq 2$ *and the second where* $y_1 \leq 1$. *This branching is shown in figure 2.4b. Solving the LP relaxation of the first subproblem leads*

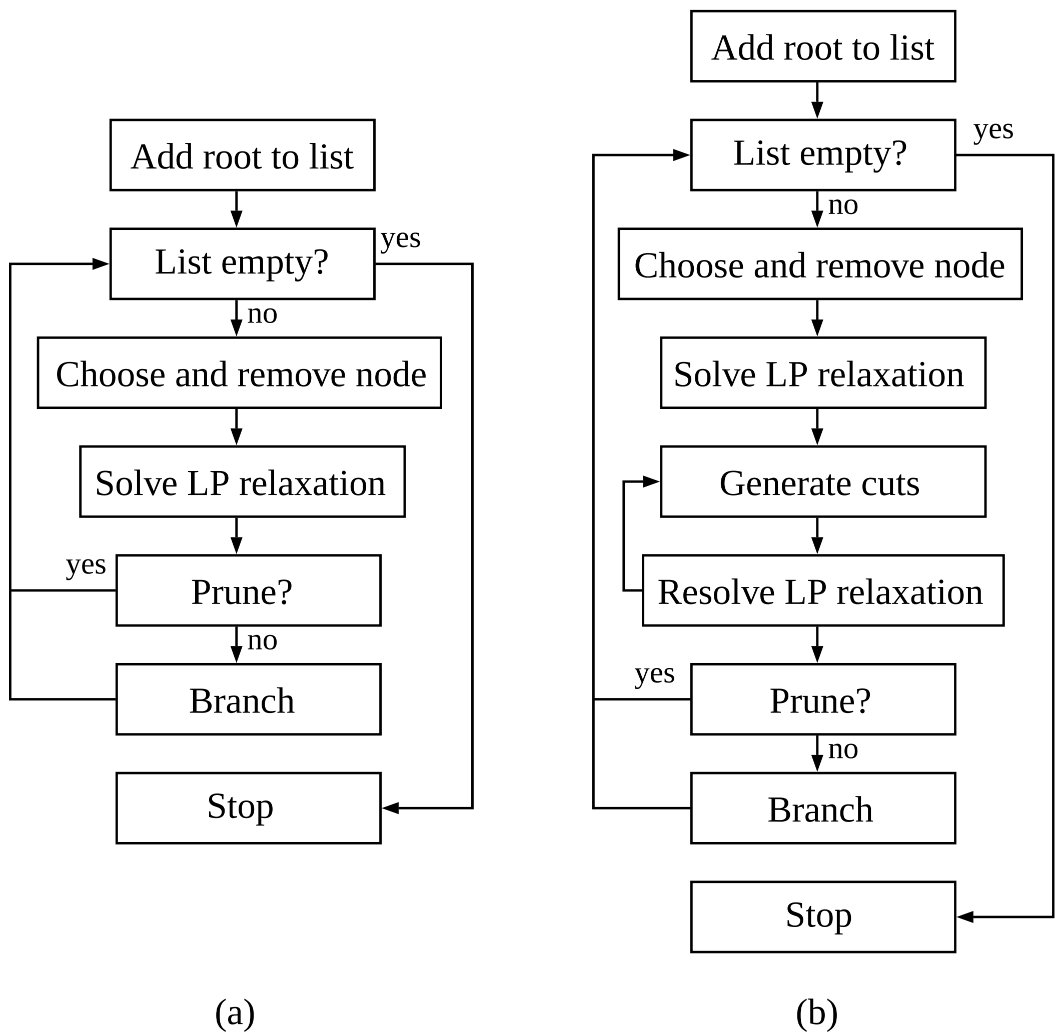

Figure 2.5.: Overview of the branch-and-bound (a) and the branch-and-cut (b) algorithms.

to the solution $y^4 = (2, 1)$ *with* $z^4 = 29$. *As it has an integer solution we can prune this node by optimality. Its solution becomes the incumbent, because it is the first integer solution we got. Solving the LP relaxation of the second subproblem leads to the solution* $y^5 = (1, 1\frac{1}{2})$ *with* $z^5 = 29\frac{1}{2}$. *This node can be pruned by bound, because its objective function value* z^5 *is worse than the one of the incumbent. As now the node list is empty the incumbent is an optimal solution to the MIP problem.*

How successful a branch-and-bound algorithm is in finding an optimal solution to an MIP problem heavily depends on the quality of the formulation of the problem. Thus it is a natural idea to expand the algorithm by a component that first automatically improves the formulation and then uses the branch-and-bound mechanisms. This extension of the branch-and-bound algorithm is called cut-and-branch, i.e. first cuts are generated using one or more separation algorithms and then the branch-and-bound algorithms starts. This can be extended to possibly generate cuts at all branch-and-bound nodes. This is then called a branch-and-cut algorithm and its structure is shown in figure 2.5b.

3. MIP Solver Software

3.1. The Use of MIP Solvers

In this section we outline a process model for the use of MIP solvers. Figure 3.1 shows typical steps when using an MIP solver for solving a real world problem. We call this approach the *MIP problem solving approach* and it is applicable to a wide range of decision and planning problems.

Before the actual MIP problem instance can be generated, it is necessary to understand the problem, write a model, and collect the input data. Understanding the problem means that it has to be decided, what the actual question to be answered is. This includes identifying factors that can be influenced and static parameters. It leads to the data that needs to be collected and to the decision variables of the MIP model. The model is typically first stated mathematically. In most cases it is helpful to identify the general class of the problem, as in operations research literature a large number of publications about models and solution approaches for certain problem classes exists.

Once the mathematical model is stated it is typically implemented in a *modeling language*. Modeling languages are software products that are either sold directly with an MIP solver or which allow to connect to a number of different MIP solvers. They enable a user to write the mathematical model in a specialized language, connecting it to the input data, and to generate the problem instances.

The input data typically origin in several data sources, such as databases or enterprise resource planning (ERP) systems. In many cases it has to be checked and cleaned before it can be used. It might also be necessary to generate the input data by methods like forecasting, because the exact input data is not known. Collecting the data is a very important step of the MIP problem solving approach because the final results heavily depend on it.

The ideal situation is that the improvement steps of the MIP problem solving approach are not needed. This is the case if the validation and evaluation of the first MIP solver run results in an acceptable solution. Often this will not be the case because of several

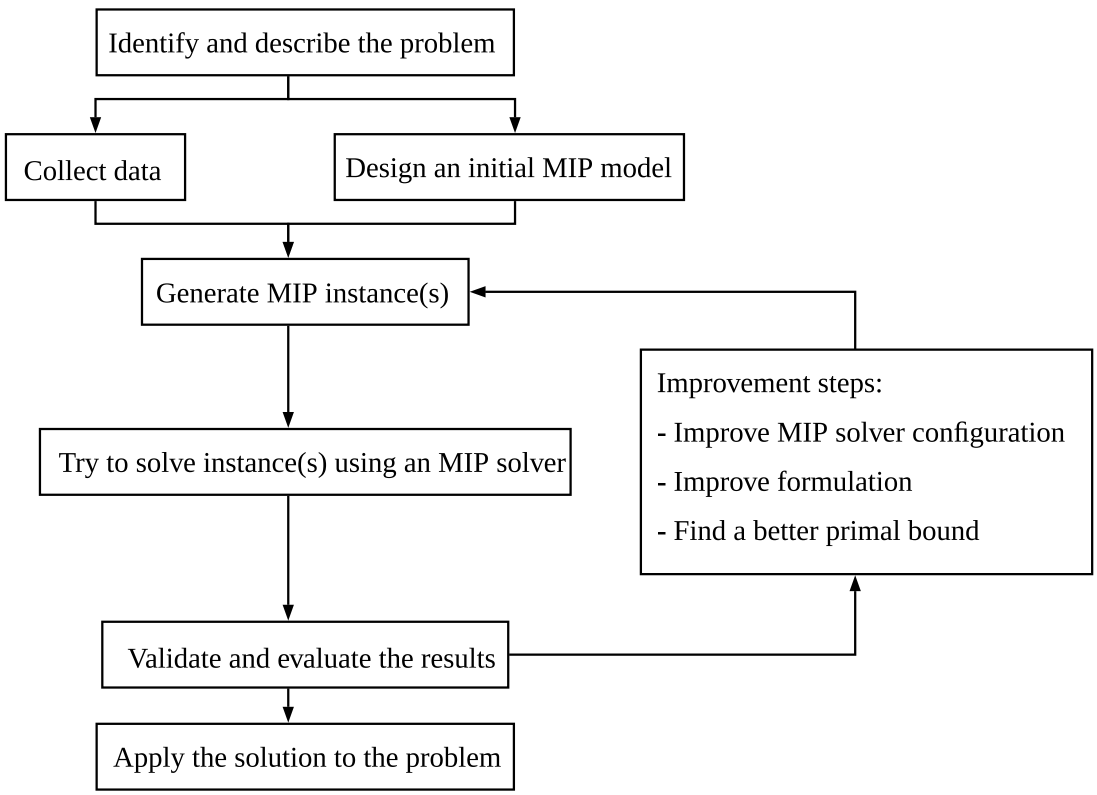

Figure 3.1.: The MIP problem solving approach.

reasons. It might be that the given problem can not be modelled close enough as an MIP problem. It might also be that the resulting model is much too large to be solved by an MIP solver. In both cases the MIP problem solving approach at least gives some idea of the structure and complexity of the problem and can direct future approaches.

Even if a model of acceptable size can be stated it might happen that the MIP solver does not return a usable solution. One situation is that the validation step might result in the observation that the model is not exact enough to give a solution to the real problem. Then it is necessary to refine the model and check the input data.

Another situation that might come up is that the MIP solver does not find any feasible solution. In this case it is advisable to implement a problem specific primal heuristic. The result of this heuristic can then be passed on to the MIP solver to further improve it or to show that it is an optimal solution. Even if the MIP solver finds feasible solutions, using a problem specific primal heuristic might improve the solutions found by the MIP solver very much. If the MIP problem is not solved to optimality within some time limit it is still possible to use the best solution found. By looking at the *global dual bound*, i.e. the best dual bound of all nodes in the node list of the branch-and-cut algorithm, it can be said how much better any optimal solution to the problem can be at best. Depending on the problem to solve, this might mean that the best solution found is good enough.

If the solution is not good enough, there are two more methods to improve it. One is to improve the formulation of the problem. As stated in section 2.2, the formulation of an MIP problem largely influences the performance of an MIP solver. Therefore adding valid inequalities or using an extended formulation might result in an acceptable solution. For certain problem types valid inequalities and extended formulations are described in literature, for example for several classes of production planning problems in [85]. Adding all valid inequalities of a family or using an extended formulation might lead to a much larger MIP model but also to extremely reduced time to solve the problem instances. A second possibility is to change the configuration of the MIP solver. MIP solvers typically have a large number of parameters that control its components. For example, it is typically possible to change the node selection and branching strategies used in the branch-and-bound part of the solver. For some solvers it is also possible to give a *weight* for each variable, which influences whether it is preferred for branching or not.

A major advantage of using the MIP problem solving approach is that instead of implementing specialized algorithms, out-of-the-box software can be used to solve many decision or planning problems. Instead of coding and dealing with all sorts of implementation problems it is only necessary to write a model in a modeling language. This also improves the maintenance of the project because if the initial problem changes, it is typically easier to change the model than to change the implementation of a problem specific algorithm.

To support the MIP problem solving approach MIP solvers aim for a number of qualities and features. The most important quality factor of a solver in the context of the MIP problem solving approach is whether it is able to solve a problem in a reasonable amount of time or at least return an acceptable solution to the problem. Therefore MIP solvers typically have very thoughtfully chosen default settings for their parameters or even adjust them dynamically. For some MIP solvers it is even possible to automatically tune them towards a set of test problem instances. It can also be seen as a quality indicator of an MIP solver if it has a large number of well-documented parameters that a user can adjust to the problems he wants to solve.

Features MIP solvers provide to further support the MIP problem solving approach typically are means to automatically improve the formulation and to find good primal bounds automatically. Automatically improving the formulation is typically done by preprocessing the problem and by generating cuts. How to generate cuts is the major topic of this thesis. Primal bounds are found using general purpose primal heuristics. See section 3.2 for more information about generating cuts and about primal heuristics in MIP solvers.

In addition to these features it is sometimes possible to tell the solver about problem-specific valid inequalities (often called *user cuts*) and primal bounds found outside the solver, for example by user implemented algorithms. For more complex situations MIP solvers are available as callable libraries to integrate them into decision support systems or to implement LP/MIP based solution techniques like *column generation* (see for example [98], chapter 11).

We now mention two examples of industry projects documented in literature that used MIP solvers in a way similar to the MIP problem solving approach. In [22], Bertsimas et al. discuss a project where they formulated an MIP problem for portfolio construction in the financial industry. They mention improving the solution time of an MIP solver by improving the parameter settings and the formulation. In the second example [44], Fleischmann et al. formulated an MIP model for the strategic planning of the supply chain of a large carmaker. They report solving all of their problem instances within 4 minutes because of the powerful preprocessing methods of modern MIP solvers.

3.2. MIP Solver Components

In this section we briefly describe the components of an MIP solver. How these components are used and how they interact with each other depends on the specific solver and on its configuration. For further information we refer to [14], [56], and [24]. The importance of the components of an MIP solver is evaluated in [25].

Input Methods

MIP solvers typically provide several ways of loading the problem instance into the solver. One is to read in problem files, the standard format for this is called *Mathematical Programming System* (MPS). Unfortunately many solver specific additions to the format have lead to the situation that MPS files of different solvers sometimes are not compatible. There are efforts for a new file format in the open-source project *Optimization Services* (OS) hosted by the COIN [1] open source project. Another way to get problems into the solver is using input functions of a *callable library* version of the solver. A callable library provides access to a set of functions that typically allow to load or create problems, set parameters, and to start the optimization process. Furthermore they enable users to implement specialized algorithms that use LP/MIP solvers to solve subproblems. These callable libraries are also frequently used to connect solvers to modeling languages that make generating and loading problems much easier.

Preprocessing Techniques

Solvers use LP and MIP *preprocessing techniques* to reduce the size and to improve the formulation of the problem. The size is reduced by removing redundant constraints and by fixing variables that can take only one value in an optimal solution. IP preprocessing techniques that aim at producing a tighter formulation are, for example, *bound reduction*, *coefficient reduction*, *reduced cost fixing* and *probing*. These techniques are described in [14]. Some preprocessing techniques can be used in all nodes of a branch-and-cut algorithm, others are only performed before or after solving the root node.

LP solver

An MIP solver always needs a *linear programming* (LP) solver to solve problems without integer variables and LP relaxations of MIP problems. LP, and implementing algorithms for LP, is a large research subject of its own. See [29] for a typical textbook about LP. There are three widely used algorithms for solving LP problems: the *primal simplex algorithm*, the *dual simplex algorithm*, and the *interior point algorithm* (also called *barrier algorithm*). State of the art LP solvers typically have all three algorithms implemented and the user can choose which one to use. In general the interior point algorithm solves many instances fastest but there are instances were the primal or the dual simplex algorithm is faster (compare [24]). A disadvantage of the interior point algorithm is that it needs more memory than the dual simplex algorithm (see [59]). Another disadvantage of the interior point algorithm is that it typically can not *warm start*, i.e. if two similar problems are solved one after the other the interior point algorithm has to start from scratch whereas the simplex-based algorithms can restart from the basis of a previous solution. Warm start is important when using an LP solver in a branch-and-cut framework where typically the dual simplex is used. Its advantage over the primal simplex algorithm is that from one node to the next the basis theoretically stays dual feasible and thus a step of the dual simplex algorithm called *dual phase one* is not needed (see [59]). Also see [59] for implementation aspects of the dual simplex algorithm.

Primal heuristics

Two types of primal heuristics are typically used in an MIP solver: *starting heuristics* and *improvement heuristics*. Starting heuristics try to find a feasible solution to the MIP problem without knowledge of another feasible solution. Many of these heuristics start with a solution to an LP relaxation and try to reach a feasible solution by rounding the solution values of the integer variables. Improvement heuristics on the other hand try to find better feasible solutions than the so far known. They typically do this by searching a *neighborhood* of the best known solution. Many currently used primal heuristics are described in [21].

Cut Generators

As MIP solvers use a branch-and-cut algorithm they need to implement separation algorithms as described in section 2.4. These implementations of separation algorithms are called *cut generators*. A cut generator gets a relaxation solution to cut off as an input information and tries to return several valid inequalities that cut off this solution. MIP solvers typically have a variety of different cut generators. Current MIP solvers typically have cut generators for all of these cuts:

- lifted cover cuts [50],
- lifted flow cover cuts [51],
- flow path cuts [93],
- clique cuts [86], [89],
- implication cuts [86], [89],
- Gomory mixed integer cuts [15],
- and complemented mixed integer rounding (cMIR) cuts [70].

Additionally, the MIP solver CPLEX [54] uses a cut generator for $\{0, \frac{1}{2}\}$-cuts [26] and the MIP solver Xpress-MP [39] additionally generates lift-and-project cuts [16].

Cut generators can be divided into *general purpose* and *problem structure-based* cut generators (see [14]). General purpose in this context means not depending on the existence of a certain problem structure in the MIP problem. The difference between these two approaches is not clear in all cases, for example, cMIR cuts are general purpose cuts but they implicitly use problem structure (see section 4.2). Therefore we differentiate in this thesis between cut generators using the LP tableau of the simplex algorithm, i.e. generate a cut for each column of the constraint matrix with a fractional solution value, and cut generators using relaxations of constraints (rows) as input. An example for a cut generator based on the simplex tableau is the cut generator for Gomory mixed integer cuts. Cut generators based on row relaxations are, for example, the flow cover, flow path, and cMIR cut generators described in chapter 4.

How cut generators are used in an MIP solver is an important aspect as well. In principle, adding more cuts leads to an improved formulation and thus helps to solve the problem. On the other hand, by each cut added, the formulation gets larger and thus the LP relaxation gets harder to solve. An MIP solver has to decide how many cuts it wants to add and which ones. For this purpose many MIP solvers use routines to select cuts generated instead of using all of them.

Branch-and-cut

The core of an MIP solver is its implementation of the branch-and-cut algorithm described in section 2.8. In this implementation a number of design decisions have to be made. Concerning cuts, one is in which nodes cuts should be generated. As generating cuts might be time-consuming and adding them increases the time needed to solve the LP relaxations, doing it in all nodes can result in a bad overall performance of the solver. Another design decision similar to this is, whether cuts are generated at nodes other than the root node are *local*, i.e. only valid for this problem and its subproblems, or *global*, i.e. valid for the original problem. In a subproblem of the branch-and-cut tree several bounds of variables have been changed due to branching. Therefore a cut for this subproblem is not necessarily a valid inequality for the original problem. If local cuts are generated, it has to be made sure that they are removed from the formulation before a subproblem where they are not valid is investigated. Removing cuts and storing which cuts belong to which set of nodes can cause many problems as well as a significant slowdown of the solver. Lifting (see section 2.7) can overcome this problem to a certain point, see [15].

An important implementation aspect of the branch-and-cut algorithm is the *branching strategy*. The branching strategy defines how branching is done. A possible goal when making this decision is to get two subproblems with large changes in the LP relaxations, because this likely leads to cutting off one of these subproblems. A number of techniques have been tried and implemented, see [5], [14], and [65] for an overview. A user of an MIP solver can typically choose among several branching strategies.

Also an important implementation aspect is the *node selection strategy*. The node selection strategy defines which node is chosen to be investigated next. The trade off here is between finding feasible solutions by investigating nodes with a bad dual bound, but probably closer to an integer solution, and improving the global dual bound by investigating the node with the best dual bound. We refer to [14] and [65] for a discussion of different methods. As with the branching strategy, a user can typically set the node selection strategy with a parameter.

3.3. The MOPS MIP Solver

In this section we briefly describe the MOPS (Mathematical Optimization System) MIP solver. The description here is based on the MOPS Whitepaper [77], the MOPS user manual [78], as well as a number of other publications about parts of MOPS ([96], [94], [45], [59], [88]).

The MOPS system started in 1987 as an LP solver and since 1994 also supports solving MIP problems. It is a commercial product sold and maintained by the MOPS GmbH & Co. KG situated in Paderborn, Germany. An overview of its components is given in figure 3.2 (from [77]).

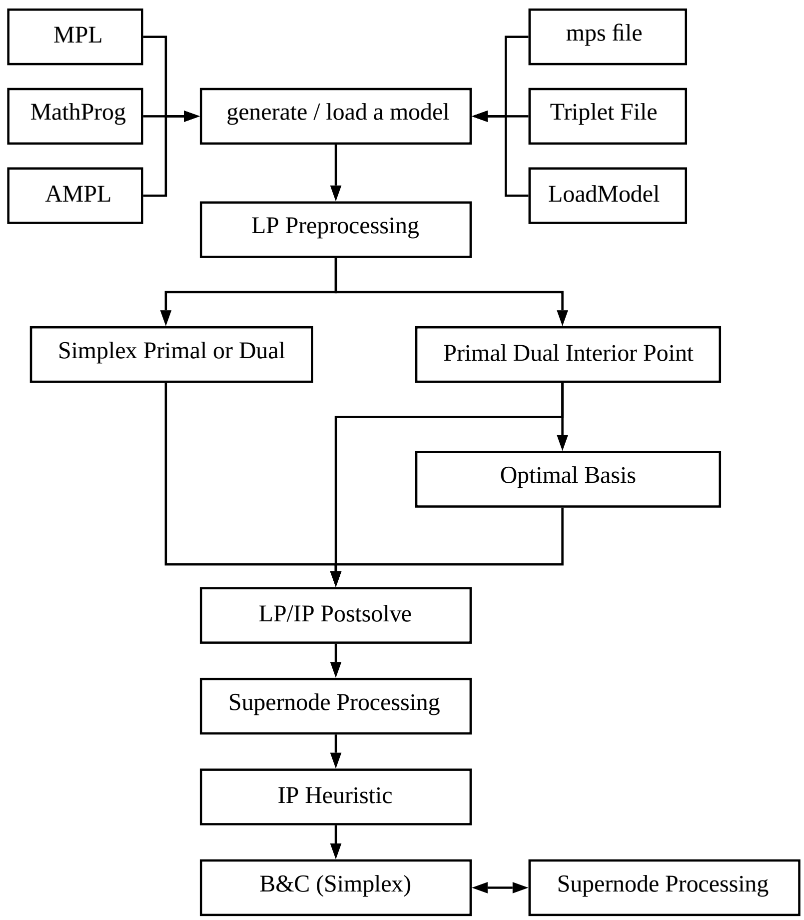

Figure 3.2.: Overview of the optimization process with MOPS (from [77]).

We now describe the part of MOPS that is most important in the context of this thesis in more detail: the cut generation. Cut generation happens in MOPS in the *supernode processing*. Supernode processing is another word for IP-preprocessing, i.e. a selection of techniques to strengthen the LP relaxation of an MIP problem. Some of these techniques are used in all nodes as part of the *node presolve*. These techniques aim at showing that the LP relaxation of a node is integer infeasible before actually solving it. Other techniques of the supernode processing, as the cut generation, are only used in *supernodes*. The root node always is a supernode and currently it is the only one. Thus MOPS actually uses a cut-and-branch algorithm.

After the LP relaxation has been solved, the main loop of the supernode processing runs through the preprocessing techniques and then calls all cut generators. The cut generators are called one after the other using the same LP relaxation solution to cut off. Cuts found are normally not added to the constraint matrix directly. Instead they are added to the *cut pool*. At the end of the supernode processing loop the *cut selection routines* select the cuts to add from the cut pool. It is possible to deactivate the cut pool with a parameter setting and directly add the cuts to the constraint matrix. This is never done in the computational tests conducted for this thesis. After the cuts have been added the LP relaxation of the improved formulation is solved.

By default, the main loop is repeated 10 times. Each iteration of the loop is called a *round* of cut generation. Then the primal heuristics of MOPS are called. If the heuristics find a feasible solution, MOPS does 10 more rounds of the supernode processing loop. In its current version, MOPS has cut generators for all the typically used cuts mentioned in section 3.2, except for flow path cuts.

We now briefly describe the cut pool and the cut selection techniques. The cut pool stores the cuts generated by several cut generators for the same LP relaxation solution and then uses sophisticated cut selection techniques. The cut selection techniques start by finding dominated and redundant cuts. After that it estimates the quality of the cuts and selects a set of cuts with high estimated quality. In this set it is tried to have cuts that are different from each other. This results in a reduced number of cuts added while the improvement in the dual bound is not much worse. For a more detailed description and a computational evaluation of the MOPS cut pool see [96].

Another important aspect of the MOPS solver is its set of *tolerance parameters*. As MIP solvers typically use floating point arithmetic they have to use very small numbers when comparing two values. In the implementations described in this thesis we use the following of MOPS tolerance parameters:

xdropm is the smallest number accepted in the constraint matrix. Its default value is 1×10^{-7}.

xtolin is the value used to check whether a variable is integer or not, i.e. a variable $y_j$is integer if $y_j^* - \lfloor y_j^* \rfloor \leq \texttt{xtolin}$. Its default value is 1×10^{-5}.

xtolzr is the zero tolerance. It is used whenever it has to be checked whether a value is really zero, i.e. a value x_j^* is considered 0, if $-\texttt{xtolzr} \leq x_j^* \leq \texttt{xtolzr}$. The default value for this tolerance parameter is 1×10^{-12}.

xtolx is the absolute primal relative feasibility tolerance (for the unscaled problem). It is used together with `xtolre`, the relative primal feasibility tolerance to decide whether a variable is within its bounds, i.e. a variable y_j with lower bound l_j and upper bound u_j is considered feasible if and only if

$$l_j - \texttt{xtolx} - \texttt{xtolre} \cdot |l_j| \leq x_j^* \leq u_j + \texttt{xtolx} + \texttt{xtolre} \cdot |u_j|$$

The default value for `xtolx` is 1×10^{-4}, for `xtolre` it is 1×10^{-10}.

There are numerous other tolerance parameters in the MOPS MIP solver. Some are used only in the LP part of MOPS , such as tolerances for factorization and arithmetic operations. Others are used in the branch-and-cut part and define, for example, when the gap between dual and primal bound is small enough to consider a solution optimal. For a detailed description of these parameters and their default settings we refer to the MOPS Whitepaper [77] and the MOPS user manual [78].

4. Separation Algorithms

4.1. The Flow Cover Cut Separation Algorithm

4.1.1. Flow Cover Inequalities

In this section we repeat results about flow cover inequalities and the flow cover cut separation algorithm. Flow cover inequalities are valid inequalities for the *binary single-node flow set* (this name for this set is from [85]) or sets similar to it.

Definition 4.1. *Consider*

$$X^{BSNF} = \{(x, y) \in \mathbb{R}^n_+ \times \{0,1\}^n : \sum_{j \in N^+} x_j - \sum_{j \in N^-} x_j \leq b, \tag{4.1}$$

$$x_j \leq u_j y_j \text{ for all } j \in N \tag{4.2}$$

where $N = (N^+, N^-)$*,*$n = |N|$*. We call* X^{BSNF} *the binary single-node flow set.*

We call rows of the form (4.1) *flow balance rows* and rows of the form (4.2) *binary variable upper bound rows.* The binary single-node flow set is a natural part of many mixed integer programming formulations. Its name is inspired by the fact that a binary single-node flow set can be visualized as a node of a fixed charge network design problem as shown in figure 4.1. Note that (N^+, N^-) is a *partition* of N as described in the appendix (p. 153).

A set similar to the binary single-node flow set was first studied by Padberg, van Roy and Wolsey in [82]. Van Roy and Wolsey then defined two families of valid inequalities for a variant of the binary single-node flow set that includes variable lower bounds in [92]. The flow cover cut separation algorithm and the first implementation of it are discussed in [93]. The families of flow cover inequalities used in this paper are called *simple generalized flow cover inequalities* (SGFCIs) and *extended generalized flow cover inequalities* (EGFCIs). All flow cover inequalities use the concept of *generalized covers.*

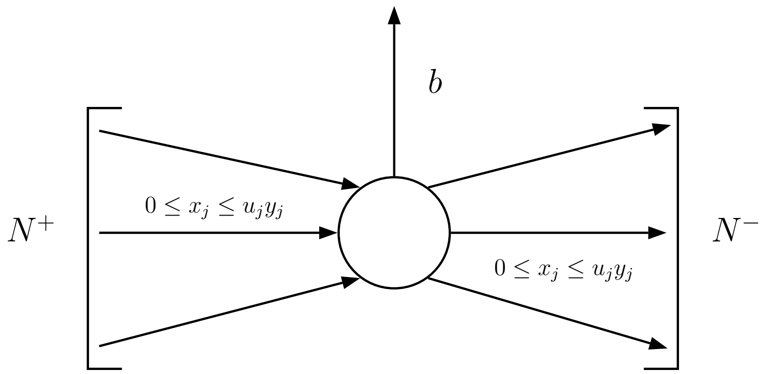

Figure 4.1.: A visual representation of the binary single-node flow set.

Definition 4.2. *A set $C = (C^+, C^-)$ with $C^+ \subseteq N^+$ and $C^- \subseteq N^-$ is called a generalized cover if*

$$\sum_{j \in C^+} u_j - \sum_{j \in C^-} u_j = b + \lambda$$

with $\lambda > 0$.

In proposition 4.1 we present the definition of the SGFCIs from [93] in the notation of this thesis.

Proposition 4.1. *If $C = (C^+, C^-)$ is a generalized cover then*

$$\sum_{j \in C^+} x_j + \sum_{j \in C^{++}} (u_j - \lambda)(1 - y_j) \leq b + \sum_{j \in C^-} u_j + \sum_{j \in L^-} \lambda y_j + \sum_{j \in L^{--}} x_j \tag{4.3}$$

where $C^{++} = \{j \in C^+ : u_j > \lambda\}$ and $N^- \setminus C^- = (L^-, L^{--})$ is called a simple generalized flow cover inequality (SGFCI) and is valid for X^{BSNF}.

We now take a closer look at the SGFCIs and try to give an intuition why these inequalities are valid for X^{BSNF}. This explanation is based on the proof in [98], p. 152.

First realize that a generalized cover is a set of indices such that not all x_j, $j \in C^+$ can be at their upper bound at the same time, even if all $j \in C^-$ are at their upper bound. Because if $x_j = u_j$ for all $j \in C$ the flow balance row would be violated by λ. To show that the SGFCIs are valid for X^{BSNF} we first look at the integer solutions $(\bar{x}, \bar{y})$ in X^{BSNF} where all $\bar{y}_j = 1$ for $j \in C^{++}$ and all $\bar{y}_j = 0$ for $j \in L^-$. Then the SGFCIs enforce that $\sum_{j \in C^+} x_j \leq b + \sum_{j \in C^-} u_j + \sum_{j \in L^{--}} x_j$ and this is valid. But if in an LP relaxation solution for one of these variables $k \in C^{++}$, $0 < y_k^* < 1$, then the SGFCIs result in $x_k \leq (u_k - \lambda) y_k + \sum_{j \in L^{--}} x_j$ and can cut off this fractional solution.

Now we look at the other integer solutions, namely those where at least one $\bar{y}_j = 0$ for $j \in C^{++}$ or at least one $\bar{y}_j = 1$ for $j \in L^-$. If $\bar{y}_j = 0$ for $j \in C^{++}$ this essentially means that the right hand side is reduced by $u_j - \lambda$. This is valid because $u_j - \lambda$ is the maximal value an x_j, $j \in C^{++}$ can take if all other variables $j \in C$ are at their upper bound. In other words, for those $j \in C^{++}$ where $\bar{y}_j = 1$ the statement that if all of these variables are at their upper bound the flow balance row exceeds the right hand side (now $b - (u_j - \lambda)$) by λ stays true. Concerning the variables $j \in N^- \setminus C^-$ we can place them in L^{--} and add x_j to the right hand side which clearly is feasible. Instead of x_j we could also use $u_j y_j$ because this is larger than x_j. But this can be strengthened to λy_j because we know that the variables $j \in C$ can exceed b by at most λ. These explanations are just hints to understand what is happening, for details see the formal proof in [98], p. 152. In example 4.1 we give a numerical example of an SGFCI.

Example 4.1. *Assume the single-node flow set*

$$\begin{aligned} x_1 + x_2 + x_3 - x_4 - x_5 &\leq 17 \\ x_1 &\leq 24y_1 \\ x_2 &\leq 20y_2 \\ x_3 &\leq 15y_3 \\ x_4 &\leq 18y_4 \\ x_5 &\leq 16y_5. \end{aligned}$$

We choose $C = \{1, 2, 5\}$,$\lambda = 11$,$L^- = \{4\}$,$L^{--} = \emptyset$ *and get the SGFCI*

$$x_1 + 13(1 - y_1) + x_2 + 9(1 - y_2) \leq 33 + 11y_4$$

which is valid for this single-node flow set.

In [87] Staellert introduced what he called a complementary class of flow cover inequalities. Atamtürk defined a special case of these in [11] and called them flow pack inequalities. He also mentioned that these flow pack inequalities can be derived using flow cover inequalities by generating them for the *reversed* flow balance row of a binary single-node flow set. *Reversed* row means in this context that we add a slack variable and multiply the row by -1. Example 4.2 illustrates this. Louveaux and Wolsey point out in [66] that the implementation in [93] already used flow cover inequalities from reversed rows.

Example 4.2. *Reversing the flow balance row from example 4.1 yields*

$$-x_1 - x_2 - x_3 + x_4 + x_5 - x_s \leq -17$$

where $x_s = 17 - x_1 - x_2 - x_3 + x_4 + x_5$ *is the slack variable for this row. We choose* $C = \{1, 5\}$,$\lambda = 9$,$L^- = \{3\}$,$L^{--} = \{2, s\}$ *and get the SGFCI*

$$x_5 + 7(1 - y_5) \leq -17 + 24 + x_2 + 9y_3 + x_s$$

which after replacing x_s *results in*

$$x_1 + x_3 - 9y_3 - x_4 + 7(1 - y_5) \leq 24.$$

This is precisely the flow pack inequality using the corresponding sets (see [11]).

The flow cover inequalities nowadays used in MIP solvers were introduced by Gu, Nemhauser and Savelsbergh in [51]. They proposed two families of valid inequalities using superadditive lifting functions. In proposition 4.2 we show the family of *lifted simple generalized flow cover inequalities* (LSGFCIs) defined by them.

Proposition 4.2. *Let* $C = (C^+, C^-)$ *be a generalized cover,* $C^{++} = \{j \in C^+ : u_j > \lambda\}$, $N^- \setminus C^- = (L^-, L^{--})$, $r = |C^{++} \cup L^-|$, $C^{++} \cup L^- = \{j_1, j_2, \dots j_r\}$ *with* $u_{j_i} \geq u_{j_{i+1}}$ *for* $i = 1, \dots r-1$,$M_0 = 0$ *and* $M_i = \sum_{k=1}^{i} u_{j_k}$ *for* $i = 1, \dots r$*. Furthermore, let* $m = \sum_{j \in C^+ \setminus C^{++}} u_j + \sum_{j \in L^{--}} u_j$, $mp = \min_{j \in C^{++}} u_j$, $ml = \min\{m, \lambda\}$, *and* t *be the largest index in* $C^{++} \cup L^-$ *such that* $u_{j_t} = mp$*. Also let* $\rho_i = \max\{0, u_{i+1} - (mp - \lambda) - ml\}$ *for* $i = t, \dots, r-1$*. Then the lifted simple generalized flow cover inequality (LSGFCI)*

$$\begin{aligned} \sum_{j \in C^+} x_j + \sum_{j \in C^{++}} (u_j - \lambda)(1 - y_j) + \sum_{j \in N^+ \setminus C^+} \alpha_j x_j - \sum_{j \in N^+ \setminus C^+} \beta_j y_j \\ \leq b + \sum_{j \in C^-} u_j - \sum_{j \in C^-} g(u_j)(1 - y_j) + \sum_{j \in L^-} \lambda y_j + \sum_{j \in L^{--}} x_j \end{aligned} \tag{LSGFCI}$$

with $(\alpha_j, \beta_j) = (0, 0)$ *if* $M_i \leq u_j \leq M_{i+1} - \lambda$, $(\alpha_j, \beta_j) = (1, M_i - i\lambda)$ *if* $M_i - \lambda < u_j < M_i$, *and the superadditive lifting function* g,

$$g(z) = \begin{cases} i\lambda & M_i \leq z \leq M_{i+1} - \lambda, \quad i = 0, \dots, t-1 \\ z - M_i + i\lambda & M_i - \lambda \leq z \leq M_i, \quad i = 1, \dots, t-1 \\ z - M_i + i\lambda & M_i - \lambda \leq z \leq M_i - \lambda + ml + \rho_i, \quad i = t, \dots, r-1 \\ i\lambda & M_i - \lambda + ml + \rho_i \leq z \leq M_{i+1} - \lambda, \quad i = t, \dots, r-1 \\ z - M_r + r\lambda & M_r - \lambda \leq z \leq b + \sum_{j \in N^-} u_j \end{cases}$$

is valid for X^{BSNF}.

Computational experiments in [51] showed that the separation algorithm based on LSGFCIs performed better than variants of it using other families or combinations of other families. In example 4.3 we show a numerical example of the lifting used in LSGFCIs.

Example 4.3. *We now try to lift the cut generated in example 4.1 to make it into a LSGFCI. In this example* $C^{++} \cup L^{-} = \{1, 2, 4\}$*. Note that usually we need to sort this set but in this example it is already sorted by decreasing* u_j*. As a result of this we get* $M = (0, 24, 44, 62)$*,* $ml = 0$ *and* $t = 2$*. We also compute that* $\rho_2 = 9$*. We start by lifting* (x_3, y_3)*. Note that because we use a superadditive lifting function the lifting is sequence independent and it therefore does not matter which variable we start with. For* (x_3, y_3) *we see that* $M_1 - \lambda = 24 - 11 < 15 < 24 = M_1$*. Therefore we can use* $(\alpha_3, \beta_3) = (1, 13)$ *if this improves our cut. For lifting* (x_5, y_5) *we have to compute* $g(16)$*. As* $M_1 - \lambda = 24 - 11 \leq 16 \leq 24 = M_1$ *for this variable the second case in the definition of* g *is used. Hence* $g(16) = 16 - 24 + 11 = 3$*. The resulting cut is*

$$x_1 + 13(1 - y_1) + x_2 + 9(1 - y_2) + x_3 - 13y_3 \leq 33 - 3(1 - y_5) + 11y_4.$$

Finally, we would like to mention a few families of flow cover inequalities that were defined for variants of the binary single-node flow set. Wolsey defined in [97] a family of valid inequalities for the single-node flow set with generalized upper bound (GUB) constraints. A generalized upper bound constraint limits the binary variables in a set $S \subseteq N$ in a way that only one of them is allowed to be 1. In [13] the single-node flow set with additive variable upper bounds is studied. Additive variable upper bounds extend variable upper bounds by a constant factor on the right hand side and the possibility to have a sum of variables form an upper bound on the continuous variables. The single-node flow set with integer variable upper bounds was studied in [58].

4.1.2. The Separation Algorithm

The flow cover cut separation algorithm originates in [93] and is one of the first separation algorithms for mixed integer programming problems that was used in a branch-and-cut approach. Despite the fact that it is more than 20 years old it is still used in current MIP solvers. The flow cover cut separation algorithm as described in [93] and reused in [51] has three major steps:

1. reformulation,
2. cover finding, and
3. cut generation.

In the following we discuss these three steps in detail.

The strength of the flow cover cut separation algorithm is that, although the flow cover inequalities are defined for the binary single-node flow set, they can be used for all kinds of mixed integer rows. This is due to the fact that all mixed 0-1 rows with bounded variables can be reformulated as single-node flow sets. This was originally stated by van Roy and Wolsey in [92] and can also be found in [80], p. 286. In this subsection we show how this reformulation is used in flow cover cut separation algorithms using the notation of this thesis. The input for a flow cover cut separation algorithm is a mixed 0-1 row of the form

$$\sum_{j \in R} a_j x'_j + \sum_{j \in B} g_j y'_j \leq b \quad x' \in \mathbb{R}_+^{|R|}, \quad y' \in \{0,1\}^{|B|} \tag{4.4}$$

and additional information about simple and binary variable upper bounds:

$$\begin{aligned} x'_j &\leq u'_j y_{v_j} && \text{for all } j \in \bar{R} \subseteq R \\ x'_j &\leq u'_j && \text{for all } j \in R \setminus \bar{R}. \end{aligned}$$

Note that we assume here that all variables have lower bounds of greater than or equal to zero and finite upper bounds. To reformulate the row we add new variables to get

$$\sum_{j \in R \cup B} (a_j x'_j + g_j y'_j) \leq b \tag{4.5}$$

$$a_j x'_j + g_j y'_j \leq (|a_j| u'_j + |g_j|) y_j \text{ for all } j \in R \cup B$$

with

$$\begin{aligned}
&g_j = 0 && y_j = y'_{v_j} && \text{for all } j \in \bar{R}, v_j \notin B \\
&g_j = 0 && y_j = 1 && \text{for all } j \in R \setminus \bar{R} \\
&g_j = g_{v_j} && y_j = y'_{v_j} && \text{for all } j \in \bar{R}, v_j \in B \\
&a_j = 0 && y_j = y'_j && \text{for all } j \in B \setminus \bigcup_{j \in \bar{R}} v_j \\
&a_j = 0 && g_j = 0 && \text{for all } j \in \bigcup_{j \in \bar{R}} v_j.
\end{aligned}$$

We define $N = R \cup B$, $x_j = a_j {x'}_j + g_j y'_j$, $u_j = |a_j| u'_j + |g_j|$ and see that (4.5) is a binary single-node flow set.

The problem of finding the most violated SGFCI for a given binary single-node flow set becomes easy as soon as we decided on the generalized cover $C = (C^+, C^-)$. To find C the approach of the flow cover cut separation algorithm is to solve a *flow cover finding knapsack problem*. This knapsack problem is

$$\begin{aligned} \min & \sum_{j \in N^+} (1 - y_j^*) k_j - \sum_{j \in N^-} y_j^* k_j \\ & \sum_{j \in N^+} u_j k_j - \sum_{j \in N^-} u_j k_j > b \\ & k \in \{0, 1\} \end{aligned} \tag{4.6}$$

where k is an incidence vector for C, i.e. if $k_j = 1$ then j is in C and $k_j = 0$ otherwise. The knapsack constraint ensures that solutions are generalized covers. The objective function can be seen as a simplification of the violation of an SGFCI. We repeat this simplification that is shown in [98] and [93]. The first step is to assume that $x_j = u_j y_j$ and $u_j \geq \lambda$ for all $j \in N$. Note that from these two assumptions follows that we can choose $L^- = N^- \setminus C^-$ and $C^{++} = C^+$without decreasing the violation. Under these assumptions the SGFCI gets

$$\sum_{j \in C^+} u_j y_j + \sum_{j \in C^+} (u_j - \lambda)(1 - y_j) \leq b + \sum_{j \in C^-} u_j + \lambda \sum_{j \in N^- \setminus C^-} y_j.$$

This can be rewritten as

$$-\lambda \sum_{j \in C^+} (1 - y_j) \leq \lambda + \lambda \sum_{j \in N^- \setminus C^-} y_j$$

because $\lambda = b + \sum_{j \in C^-} u_j - \sum_{j \in C^+} u_j$. After dividing by λ and subtracting $\sum_{j \in N^-} y_j$ from both sides of the inequality we get

$$\sum_{j \in C^+} (1 - y_j) - \sum_{j \in C^-} y_j \geq 1 - \sum_{j \in N^-} y_j.$$

As the right hand side is independent from C we can use the left hand side as the objective function for the flow cover finding knapsack problem. Note that the assumptions we made do not hold in general and therefore solving the flow cover finding knapsack problem does not automatically result in the set C that leads to the most violated flow cover cut. Hence this method is heuristic in contrast to the cover finding knapsack problem for cover cuts (see for example [98]).

Once we decided on a cover the final step is the cut generation. Here one decision to make is which variables to place in L^-. The rule

$$L^- = \{j \in N^- : \lambda y_j^* < x_j^*\}$$

leads to the most violated cut. Furthermore the cut coefficients for the lifted variables have to be computed. The final step is to rewrite the generated cut in the space of the original variables.

4.2. The Aggregated cMIR Cut Separation Algorithm

4.2.1. Mixed Integer Rounding Inequalities

Mixed integer rounding inequalities are valid inequalities for the mixed integer knapsack set. They go back to [79], but the presentation here is based on [98] and [70].

Definition 4.3. *The mixed integer knapsack set is defined as*

$$X^{MIK} = \{(y,s) \in \mathbb{Z}_+^{|I|} \times \mathbb{R}_+^1 : \sum_{j \in I} g_j y_j \leq b + s, y_j \leq u_j \text{ for } j \in I\}. \tag{4.7}$$

Note that, as in the case of the single node flow set in section 4.1, nearly all mixed integer rows can be reformulated to a mixed integer knapsack set. In section 2.5 we show the simple MIR inequality that is valid for the simple mixed integer set. The MIR inequality is a straight forward application of the simple MIR inequality (from section 2.5) to the mixed integer knapsack set X^{MIK}.

Proposition 4.3. *The mixed integer rounding (MIR) inequality*

$$\sum_{j \in I} \left(\lfloor g_j \rfloor + \frac{(f_j - f)^+}{1 - f} \right) y_j \leq \lfloor b \rfloor + \frac{s}{1 - f} \tag{4.8}$$

where $f = b - \lfloor b \rfloor$ and $f_j = g_j - \lfloor g_j \rfloor$ is valid for X^{MIK}.

Proof. We split I into the disjunct sets I_1 and I_2 and relax the mixed integer knapsack set to

$$\sum_{j \in I_1} \lfloor g_j \rfloor y_j + \sum_{j \in I_2} (\lceil g_j \rceil - 1 + f_j) y_j \leq b + s.$$

Then we rewrite it as a simple mixed integer set $X^{\leq}$ (see page 14)

$$\underbrace{\sum_{j \in I_1} \lfloor g_j \rfloor y_j + \sum_{j \in I_2} \lceil g_j \rceil y_j}_{y' \in \mathbb{Z}} \leq b + \underbrace{s + \sum_{j \in I_2} (1 - f_j) y_j}_{s' \in \mathbb{R}_+} .$$

Note that for this step $y_j \geq 0$ for $j \in I$ is a necessary condition. Applying the simple MIR inequality yields

$$\sum_{j \in I_1} \lfloor g_j \rfloor y_j + \sum_{j \in I_2} \lceil g_j \rceil y_j \leq b + \frac{s + \sum_{j \in I_2} (1 - f_j) y_j}{1 - f}$$

or rewritten

$$\sum_{j \in I_1} \lfloor g_j \rfloor y_j + \sum_{j \in I_2} (\lfloor g_j \rfloor + \frac{1 - f}{1 - f} - \frac{1 - f_j}{1 - f}) y_j \leq b + \frac{s}{1 - f}.$$

If we choose $j \in I_2$ if $f < f_j$ this gives the MIR inequality. □

There is a number of results about the relation between the MIR inequalities and other families of valid inequalities. For these results we refer to [79], [68], [70], and [98].

Another family of valid inequalities for the mixed integer knapsack set can be defined by *complementing* some of the integer variables of the mixed integer knapsack set before applying the MIR inequality. Complementing means adding the upper bound of a variable to both sides of the inequality and substituting $u_j - y_j$ by a new variable $\bar{y}_j$. This can be combined with rescaling the mixed integer knapsack using a value $\delta > 0$ to obtain a fractional right hand side that is needed for a meaningful MIR inequality. A result of these two operations is the definition of the *complemented mixed integer rounding (cMIR) inequalities.*

Proposition 4.4. *Let $I = (T, C)$ and $\delta \in \mathbb{R}_{>0}$. The cMIR inequality associated to (T, C) and δ,*

$$\sum_{j \in T} F_{f_\beta, \delta}(g_j) y_j + \sum_{j \in C} F_{f_\beta, \delta}(-g_j)(u_j - y_j) \leq \delta(1 - f_\beta)\lfloor \beta \rfloor + s \tag{4.9}$$

where

$$\beta = \frac{b - \sum_{j \in C} g_j u_j}{\delta},$$

$f_\beta = \beta - \lfloor \beta \rfloor$, $f_{\frac{g}{\delta}} = \frac{g}{\delta} - \lfloor \frac{g}{\delta} \rfloor$ and

$$F_{\alpha,\delta}(g) = \delta(1-\alpha)\left(\left\lfloor \frac{g}{\delta} \right\rfloor + \frac{(f_{\frac{g}{\delta}} - \alpha)^+}{1-\alpha}\right)$$

is valid for X^{MIK}.

Proof. We complement the variables in C by subtracting their upper bounds from both sides of the mixed knapsack inequality, divide the inequality by δ and get

$$\sum_{j\in T} \frac{g_j}{\delta} y_j + \sum_{j\in C} \frac{-g_j}{\delta}(u_j - y_j) \leq \frac{b}{\delta} - \sum_{j\in C} \frac{g_j}{\delta} u_j + \frac{s}{\delta}.$$

Applying the mixed integer rounding inequality from section 2.5 and multiplying by $\delta(1 - f_\beta)$ yields the cMIR inequality. □

The cMIR inequality was introduced in [68] and [70]. Its inherent strength is that, depending on a careful construction of the mixed knapsack set, many families of valid inequalities can be generated using cMIR inequalities. These are, for example, residual capacity inequalities [67] and mixed cover inequalities [69]. The construction of the mixed knapsack set together with the generation of cMIR cuts is called the cMIR approach and is discussed in the next section. There we also show that some flow cover inequalities can also be derived using the cMIR approach.

Instead of defining the MIR inequalities for the traditional mixed integer knapsack set it is also possible to define them for what we call the *reversed mixed integer knapsack set.*

Definition 4.4. *The reversed mixed integer knapsack set is defined as*

$$X^{RMIK} = \{(y, s) \in \mathbb{Z}_+^{|I|} \times \mathbb{R}_+^1 : \sum_{j\in I} g_j y_j + s \geq b, y_j \leq u_j \text{ for } j \in I\}. \tag{4.10}$$

The reversed mixed integer knapsack set is basically a mixed integer knapsack set with a greater than or equal sign and a positive coefficient for the continuous variable. The reversed MIR inequality is mentioned in several publications as a by-product (for example in [12]).

Proposition 4.5. *The reversed mixed integer rounding (MIR) inequality*

$$\sum_{j \in I} (f \lfloor g_j \rfloor + \min\{f_j, f\}) y_j + x \geq f \lceil b \rceil$$

where $f = b - \lfloor b \rfloor, f_j = g_j - \lfloor g_j \rfloor$ *is valid for* X^{RMIK}*.*

Proof. We partition I into two sets I_1 and I_2 and relax X^{RMIK} to get a simple MIR set $X^{\geq}$:

$$\underbrace{\sum_{j \in I_1} \lceil g_j \rceil y_j + \sum_{j \in I_2} \lfloor g_j \rfloor y_j}_{y' \in \mathbb{Z}} + \underbrace{\sum_{j \in I_2} f_j y_j}_{s' \in \mathbb{R}_+} + s \geq b.$$

Now we generate a simple MIR inequality (see section 2.5) for the simple MIR set with the variables (y', s'). Note that again $y_j \geq 0$ for $j \in I$ is necessary for this step. The result is

$$\sum_{j \in I_2} f_j y_j + x \geq f \left(\lceil b \rceil - \sum_{j \in I_1} \lceil g_j \rceil y_j - \sum_{j \in I_2} \lfloor g_j \rfloor y_j \right)$$

which can be rewritten (assuming without loss of generality that $f_j > 0$ for $j \in I_1$) as

$$\sum_{j \in I_2} (f \lfloor g_j \rfloor + f_j) y_j + \sum_{j \in I_1} f(\lfloor g_j \rfloor + 1) y_j + x \geq f \lceil b \rceil$$

or

$$\sum_{j \in I_2} (f \lfloor g_j \rfloor + f_j) y_j + \sum_{j \in I_1} (f \lfloor g_j \rfloor + f) y_j + x \geq f \lceil b \rceil.$$

After recombining I_1 and I_2 this becomes

$$\sum_{j \in I} (f \lfloor g_j \rfloor + \min\{f_j, f\}) y_j + x \geq f \lceil b \rceil$$

which is the desired inequality. □

In the same way cMIR inequalities are defined for mixed integer knapsack sets, reversed cMIR inequalities can be defined for reversed mixed integer knapsack sets. By adding a slack variable it is possible to generate the cMIR cuts using reversed cMIR cuts and the other way around.

Proposition 4.6. *Let (T, C) be a partition of I and $\delta \in \mathbb{R}_{>0}$. The reversed cMIR inequality associated to (T, C) and δ,*

$$\sum_{j \in T} G_{f,\delta}(g_j) y_j + \sum_{j \in C} G_{f,\delta}(-g_j)(u_j - y_j) + x \geq f \left\lceil \frac{\beta}{\delta} \right\rceil \tag{4.11}$$

where

$$\beta = b - \sum_{j \in C} g_j u_j,$$

$f = \beta - \delta \left\lfloor \frac{\beta}{\delta} \right\rfloor$, $f_g = g - \delta \left\lfloor \frac{g}{\delta} \right\rfloor$ *and*

$$G_{\alpha,\delta}(g) = \alpha \left\lfloor \frac{g}{\delta} \right\rfloor + \min\{f_g, \alpha\}$$

is valid for X^{RMIK}.

Proof. Complement the variables in C and divide by $\delta > 0$ to get the greater than or equal mixed integer knapsack

$$\sum_{j \in T} \frac{g_j}{\delta} y_j + \sum_{j \in C} \frac{-g_j}{\delta}(1 - y_j) + s \geq \frac{b}{\delta} - \sum_{j \in C} \frac{g_j}{\delta} u_j$$

and apply the reversed c-MIR inequality. Multiplying the result by δ yields the desired inequality. □

We now briefly discuss the concept of *cMIR rank*. The rank of a cMIR inequality is defined in [69], p. 34. For the context of this thesis it is sufficient to know that a cMIR inequality has a rank of one if the smallest set needed to generate it contains only original constraints of a problem, possibly reversed as described above. A cut has rank 2 if the smallest set needed to generate it contains original constraints and additionally at least one rank one cMIR inequality. This continues in the same way recursively for higher ranks. Note that for mixed 0-1 problems the cMIR rank is well defined in the sense that each valid inequality for a mixed 0-1 problem has a finite cMIR rank. This is not the case for general mixed integer problems. As demonstrated by Dash and Günlük in [37], rank one cMIR inequalities can improve the formulation of many problem instances very much. For other problem instances higher-rank cMIR inequalities are very important, these are for example lot-sizing instances. See section 5.6 for details.

Finally, we would like to mention two families of valid inequalities with a strong connection to the MIR inequalities. The first is the family of two-step MIR inequalities introduced in [36] by Dash and Günlük. They can be obtained by applying the simple mixed integer rounding inequality twice. This principle was extended to n-step MIR inequalities by Kianfar and Fathi in [57]. The separation and the computational effectiveness of two-step MIR inequalities was investigated in [35]. The result of their paper is that two-step MIR inequalities are among the more important cuts that are not MIR cuts but their importance for practically solving mixed integer programming problems is still not clear. A second family of valid inequalities related to the MIR inequalities is the family of mingling inequalities introduced by Atamtürk and Günlük in [12]. Their mingling procedure uses upper bounds of variables to get strong valid inequalities that previously only could be generated using lifting techniques. They show that the mingling inequalities dominate cMIR inequalities under certain conditions. A separation algorithm is not described in their paper and the computational effectiveness remains to be tested.

4.2.2. The Separation Algorithm

The aggregated cMIR cut separation algorithm was introduced in [68] and [70]. It is build upon the *cMIR approach* (or cMIR separation routine) that consists of three steps:

1. aggregation,
2. bound substitution, and
3. cut generation.

The cMIR approach has the nice property that it can be used to derive several families of valid inequalities for different problem classes. For the purpose of this thesis it is especially interesting to know that, as Marchand showed in [68], the SGFCIs from section 4.1 can also be derived using the cMIR approach.

To show this we start with a single node flow set (see page 35) and relax it by replacing some $x_j,\ j \in (C^+, C^- \cup L^-) \subseteq (N^+, N^-)$ by their variable upper bound constraints $x_j = u_j y_j - t_j$ where the variables t_j are slack variables. After relaxing the resulting inequality by dropping x_j for $j \in N^+ \setminus C^+$ and t_j for $j \in C^- \cup L^-$ the result is the mixed integer knapsack set

$$\sum_{j \in C^+} u_j y_j - \sum_{j \in C^- \cup L^-} u_j y_j \le b + s$$

where $s = \sum_{j \in C^+} t_j + \sum_{j \in N^- \setminus (C^- \cup L^-)} x_j$. The next step is to complement some variables $j \in (C^+, C^-) = C$, where C is a generalized flow cover (see definition 4.2) and to divide the row by $\delta = \max_{j \in C^+ \cup L^-} u_j$. The result is the mixed integer knapsack set

$$\sum_{j \in C^+} \frac{-u_j}{\delta}(1 - y_j) + \sum_{j \in C^-} \frac{u_j}{\delta}(1 - y_j) + \sum_{j \in L^-} \frac{-u_j}{\delta} y_j \leq \frac{-\lambda}{\delta} + \frac{s}{\delta}$$

because $b - \sum_{j \in C^+} u_j + \sum_{j \in C^-} u_j = -\lambda$. We can relax this to

$$\sum_{j \in C^+} \frac{-u_j}{\delta}(1 - y_j) - \sum_{j \in L^-} y_j \leq \frac{-\lambda}{\delta} + \frac{s}{\delta}$$

because $-1 \leq \frac{-u_j}{\delta}$ for $j \in L^-$ and $\frac{u_j}{\delta}(1 - y_j) \geq 0$ for $j \in C^-$.

Using the MIR inequality we get

$$\sum_{j \in C^+} \left(\left\lfloor \frac{-u_j}{\delta} \right\rfloor + \frac{(f_j - f)^+}{1 - f} \right) (1 - y_j) - \sum_{j \in L_-} y_j \leq -1 + \frac{s}{\lambda}$$

where $f = \frac{-\lambda}{\delta} + 1$ and $f_j = \frac{-u_j}{\delta} + 1$. Now observe that $(\frac{-u_j}{\delta} + \frac{\lambda}{\delta})^+ = \frac{(-u_j + \lambda)^+}{\delta}$. After rewriting we get

$$\sum_{j \in C^+} \left(-1 + \frac{(-u_j + \lambda)^+}{\lambda} \right) (1 - y_j) - \sum_{j \in L_-} y_j \leq -1 + \frac{s}{\lambda}.$$

We multiply by λ and substitute for s.

$$\sum_{j \in C^+} \left(-\lambda + (-u_j + \lambda)^+ \right) (1 - y_j) - \sum_{j \in L^-} \lambda y_j \leq -\lambda + \sum_{j \in C^+} t_j + \sum_{j \in N^- \setminus (C^- \cup L^-)} x_j$$

Substituting t_j and λ yields

$$\sum_{j \in C^+} x_j + \sum_{j \in C^+} \left(u_j - \lambda + (-u_j + \lambda)^+ \right) (1 - y_j) - \sum_{j \in L^-} \lambda y_j \leq b \sum_{j \in C^-} u_j + \sum_{j \in N^- \setminus (C^- \cup L^-)} x_j$$

and realizing that $u_j - \lambda + (-u_j + \lambda)^+ = (u_j - \lambda)^+$ results in the definition of the SGFCIs.

In [66] Louveaux and Wolsey state that LSGFCIs (see section 4.1) in general can not be generated using cMIR inequalities. Nevertheless there are cases where applying the cMIR inequalities yields the same results as applying the LSGFCIs. We show this in example 4.4. The results of a computational comparison of flow cover and cMIR cut separation algorithms are discussed in section 6.4.4.

Example 4.4. *We reuse the single node flow set from example 4.1 and generate a cMIR cut for it. The first step is to substitute* $u_j y_j - t_j$ *for* x_j *where* t_j *is the slack variable for the variable upper bound constraint of* j*. We drop those slack variables that have a positive coefficient. The result is the following mixed integer knapsack set*

$$24y_1 + 20y_2 + 15y_3 - 18y_4 - 16y_5 \leq 17 + \underbrace{t_1 + t_2 + t_3}_{s}.$$

Now we choose $C = \{1, 2, 5\}$ *(as in example 4.1) and* $\delta = \max_{j \in C^+ \cup L^-} u_j = 24$ *and generate the cMIR cut*

$$-11(1 - y_1) - 11(1 - y_2) + 2y_3 - 11y_4 + 3(1 - y_5) \leq -11 + s.$$

After substituting s *and* $t_j = u_j - x_j$ *we get the cut*

$$x_1 - 13y_1 + x_2 - 9y_2 + x_3 - 13y_3 - 11y_4 - 3y_5 \leq 8.$$

By not substituting the variable bound of x_3 *and relaxing* $3(1 - y_5)$ *we can get the SGFCI from example 4.1. In this special case the cMIR inequality is the same as the LSGFCI from example 4.3.*

The cMIR separation algorithm follows the steps of the cMIR approach and tries to solve the underlying separation problem using appropriate heuristics. The first step, the *aggregation heuristic*, constructs a base row for the following steps out of input rows of the constraint matrix. First a single row of the constraint matrix is used as a base row. If no violated valid inequality for this row is found the aggregation heuristic searches for a row to add to the current base row to get an aggregated row that is used as the new base row. This is repeated a given number of times. The actual task of the separation heuristic is to find a new row to add to the current base row. This is done in two steps, the identification of a continuous variable to eliminate and the selection of a row that can be used to eliminate the identified variable.

For a base row p that is the aggregation of a set $P \subseteq M$, where M is the set of all rows of the constraint matrix, p is in the form

$$\sum_{j \in N_p} a_{pj} x_j + \sum_{j \in I_p} g_{pj} y_j = b_p \qquad x \in \mathbb{R}^{|N_p|}, y \in \mathbb{Z}^{|I_p|}.$$

The set of possible variables to choose from for elimination is given by Marchand and Wolsey in [70] as

$$N_p^* = \{j \in N_p : a_{pj} \neq 0, l_j y_j < x_j^* < u_j y_j \text{ and } \exists q \in M \setminus P \text{ with } a_{qj} \neq 0\}$$

Their aggregation heuristic suggests to choose the variable $k \in N_p^*$ with the largest distance to its bounds Δ_k, that means this index k is defined as

$$k = \arg\max_{j \in N_p^*} \{\Delta_j\} \qquad \Delta_j = \min\{x_j^* - l_j y_j^*, u_j y_j^* - x_j^*\}.$$

Note that here and in the following we assume that for all variables $j \in N$ a variable lower and upper bound exists, that means $l_j y_j \leq x_j \leq u_j y_j$. If this is not the case we assume that the binary variable for the variable lower or upper bound is fixed to one. If a variable is not bounded u_j or l_j is assumed to be a very large but finite number.

Once the variable for elimination is selected we need to choose a row $q \in M \setminus P$ to add to the current base row. Marchand and Wolsey do not specify what rule to use for this decision. After that the new row q is rescaled to eliminate k and added to the base row. Note that Marchand and Wolsey suggest to limit the total number of rows that are aggregated to 6.

Within the aggregation heuristic, for each base row the *bound substitution heuristic* is called to transform it into a mixed integer knapsack. The bound substitution heuristic has to decide for each variable $j \in N$ whether to replace it with its lower bound, $x_j = l_j y_j + t_j$, or its upper bound $x_j = u_j y_j - t_j$ where the non-negative variables t_j are slack variables of the bound constraints. The result is a row of the form

$$\sum_{j \in I} g_j' y_j + \sum_{j \in N} a_j' t_j = b'$$

that after relaxing the variables $j \in N$ where $a_j' > 0$ becomes the mixed integer knapsack

$$\sum_{j \in I} g_j' y_j - s \leq b'$$

with

$$s = \sum_{j \in N : a_j' < 0} a_j' t_j.$$

Note that because all variables $j \in N$ are replaced by non-negative slack variables it is not necessary to assume that the continuous variables in the base row have a lower bound of greater than or equal to zero. But the bound substitution fails for variables that are unbounded to both sides, that means if $l_j = -\infty$ and $u_j = \infty$. These variables are also called *free variables*. The decision that has to be made for the bound substitution is whether to replace the lower or the upper variable bound. Marchand and Wolsey suggest three criteria:

1. choose the bound that is closer, i.e., use $x_j = l_j y_j + t_j$ only if $x_j^* - l_j y_j^* \leq u_j y_j^* - x_j$, else use $x_j = u_j y_j - t_j$
2. Minimize the value of $s^* = \sum_{j \in P : a'_j < 0} a'_j t_j^*$
3. Maximize the value of $s^* = \sum_{j \in P : a'_j < 0} a'_j t_j^*$

The *cut generation heuristic* (or separation heuristic) finally has to decide on the set C of variables to complement and the rescaling factor $\delta > 0$. The algorithm by Marchand and Wolsey initially sets $C = \{j \in N : y_j^* \geq \frac{u_j}{2}\}$ and generates cuts for several values of $\delta = \{a_j : j \in N \text{ and } 0 < y_j^* < u_j\}$. It chooses the δ that leads to the most violated cut and then tries to improve this cut by generating cuts with $\frac{\delta}{2}$, $\frac{\delta}{4}$, and $\frac{\delta}{8}$. For the then most violated cut the set C is iteratively increased by variables not yet in C and it is checked whether the resulting cut is more violated. The variables are added to C in non-decreasing order of $|y_j^* - \frac{u_j}{2}|$, if the resulting cut is more violated than the best one so far, it stays in C, if not, it is removed and the next variable is tried.

Unfortunately, the description of the separation algorithm by Marchand and Wolsey leaves some open questions. The most important is that in the aggregation heuristic it is not mentioned which row is selected to add to the current base row. In a publication by Gonçalves and Ladanyi [48] this question is investigated. Their paper is about one of the implementations of the cMIR cut separation algorithm used by the COIN CBC open source MIP solver [1]. They compare whether using the first or a random row in the aggregation heuristic makes a difference. Their result is that it does not. That indicates that taking the first row is as good as taking a random one. Another important aspect they point out is that for a good performance of the separation algorithm it is important to also use the *reversed rows* as input for the algorithm. The reversed rows are computed from the input rows by adding slack variables and then multiplying them by -1.

Another description of an implementation of the cMIR separation algorithm is given by Wolter in [99]. This publication also discusses the selection of the row to aggregate and suggests two new rules based on density and the value of *dual variables*. It also investigates small changes to bound substitution and cut generation. Additionally accuracy safeguards similar to those mentioned in section 5.2 are described. Louveaux and Wolsey show in

[66] that by using their *MIR approach*, which is a variant of the cMIR approach that also uses sequence independent lifting, it is possible to generate a family of valid inequalities that dominates the LSGFCIs. An implementation of this approach without lifting can also be found in [99].

A different approach for separating MIR inequalities is to optimize over the *MIR closure*. The MIR closure can be described as the best possible formulation that can be obtained by adding rank one MIR inequalities. This approach for separating MIR inequalities involves finding a linear combination of all rows of the constraint matrix instead of working with a small set of rows, as the algorithm described above do. This linear combination is found by linearizing a non-linear optimization problem with integer variables and solving it with an MIP solver. This approach is described in [37]. There are also approaches to optimize over the split closure [27], [18] that is known to be equivalent to the MIR closure. All of these approaches are currently too time-consuming to be used in MIP solvers. Nevertheless, for some very hard instances, they can be used to find cuts that lead to solving instances that otherwise are not solvable (see [18]).

4.3. The Flow Path Cut Separation Algorithm

4.3.1. Flow Path Inequalities

The flow path inequalities are based on an idea from the paper [91] by van Roy and Wolsey which is about valid inequalities for uncapacitated fixed charge networks. They were first explicitly defined in [93]. Although nearly all MIP solvers have an implementation of the separation algorithm for flow path cuts, very little is published about them besides these two initial publications. One, for example, is [30] where a flow path cut generator is mentioned among other cut generators of a cutting plane framework. Flow path inequalities are valid inequalities for *fixed charge paths*.

Definition 4.5. *A fixed charge path is described by the constraints*

$$\begin{aligned}
-x_1 + \sum_{j \in N_1^+} x_j - \sum_{j \in N_1^-} x_j &\leq b_1 \\
+x_{k-1} - x_k + \sum_{j \in N_k^+} x_j - \sum_{j \in N_k^-} x_j &\leq b_k \text{ for } k = 2, \ldots, K-1 \\
+x_{K-1} \qquad + \sum_{j \in N_K^+} x_j - \sum_{j \in N_K^-} x_j &\leq b_K \\
x_k &\geq 0 \text{ for } k = 1, \ldots, K-1 \\
l_j y_j \leq x_j &\leq u_j y_j \\
y_j \in \{0,1\} \quad j &\in \bigcup_{k=1}^{K} (N_k^+ \cup N_k^-).
\end{aligned}$$

There are two families of flow path inequalities described in [93], simple and extended network inequalities. Note that in compliance with [91] we changed b_t to b_t^+ where $b_t^+ = \max\{0, b_t\}$.

Proposition 4.7. *The simple network inequality*

$$\sum_{k=1}^{K} \sum_{j \in C_k^+} x_j \leq \sum_{k=1}^{K} \sum_{j \in C_k^+} \left(\sum_{t=k}^{K} b_t^+ \right) y_j + \sum_{k=1}^{K} \sum_{j \in N_k^-} x_j \tag{4.12}$$

where $C_k^+ \subseteq N_k^+$ for $k = 1 \ldots K$ is valid for fixed charge paths as defined in definition 4.5.

Proposition 4.8. *The extended network inequality*

$$\sum_{k=1}^{K} \sum_{j \in C_k^+} x_j \leq \sum_{k=1}^{K} \sum_{j \in C_k^+} \left(\sum_{t=k}^{K} b_t^+ + \sum_{i \in Q_t^-} u_t \right) y_j + \sum_{k=1}^{K} \sum_{j \in N_k^- \setminus Q_k^-} x_j \tag{4.13}$$

where $C_k^+ \subseteq N_k^+$ and $Q_k^- \subseteq N_k^-$ for $k = 1 \ldots K$ is valid for fixed charge paths as defined in definition 4.5.

Typically, violated flow path inequalities are only found for certain problem classes. These problem classes are foremost fixed charge network design and lot-sizing problems. For lot-sizing problems this can be explained by the fact that the well-known (l, S) *inequalities* described in [19], which are known to be facet defining for the uncapacitated lot-sizing problem, can be generated using flow path inequalities.

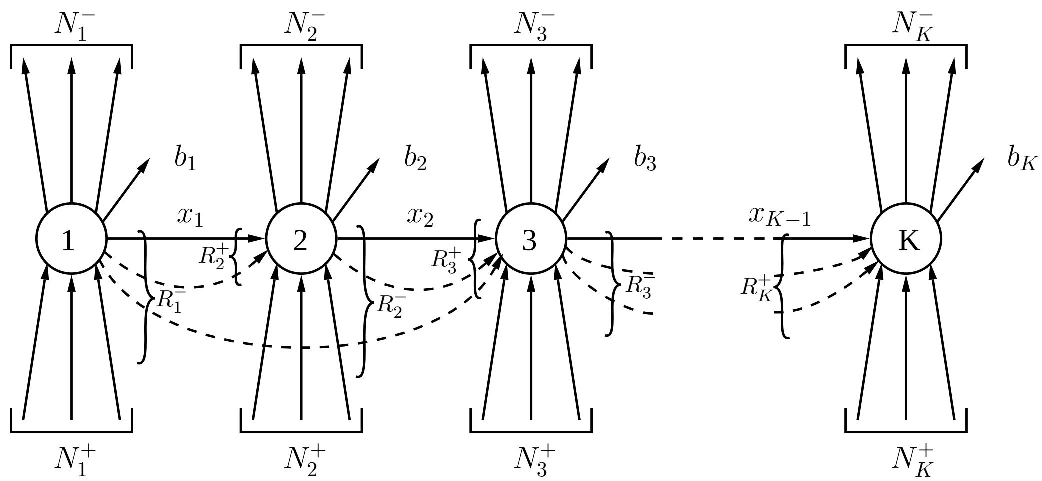

Figure 4.2.: A (generalized) fixed charge path (from [93]).

4.3.2. The Separation Algorithm

The separation algorithm for flow path cuts from [93] uses the fact that the flow path inequalities are also valid for a slightly more general structure than the fixed charge path shown in the previous section. It is called the *generalized fixed charge path*, figure 4.2 visualizes it. Note that a generalized fixed charge path can be seen as a collection of binary single node flow sets.

Definition 4.6. *A generalized fixed charge path is described by the constraints*

$$\sum_{j \in R_k^+} a_{kj}x_j - \sum_{j \in R_k^-} a_{kj}x_j + \sum_{j \in N_k^+} a_{kj}x_j - \sum_{j \in N_k^-} a_{kj}x_j \leq b_k \qquad k = 1 \dots K$$

$$l_j y_j \leq x_j \leq u_j y_j$$

$$y_j \in \{0,1\} \quad j \in \bigcup_{k=1}^{K} (R_k^+ \cup R_k^- \cup N_k^+ \cup N_k^-).$$

where for any k the sets R_k^+, R_k^-, N_k^+, and N_k^- are disjunct, $a_{kj} > 0$ for all $j \in R_k^+ \cup R_k^- \cup N_k^+ \cup N_k^-$. Furthermore $j \in R_k^+$ implies $j \in R_i^-$ for some $i < k$ and $j \notin R_i^+$ for all $i > k$ as well as $j \in R_k^-$ implies $j \in R_i^+$ for some $i > k$ and $j \notin R^-$ for all $i < k$.

The flow path cut separation algorithm has two phases:

1. identify a generalized fixed charge path and
2. find the most violated flow path inequality.

The path is identified using a path-augmenting greedy heuristic. It is called with each mixed 0–1 row of the constraint matrix as an input row. It starts by choosing this input row as the first row of the path and then adds more rows to it, until a cut is found or a certain maximal path length is reached. When choosing which row to add, the procedure is similar to the aggregation heuristic of the cMIR cut separation algorithm. First, it identifies an outgoing arc, i.e. a variable j where $a_{kj} < 0$, with largest outflow x_j^* in the current LP solution. Second, it identifies a new row $k+1$ to add to the path where j is an inflow arc, i.e. $a_{k+1j} > 0$. After finding a rescaling factor γ such that $a_{kj} + \gamma a_{k+1j} = 0$ it adds the rescaled row to the path.

Once a new row is added the algorithm checks whether a simple network inequality with $C_k^+ = \{j \in N_k^+ : x_j^* > (\sum_{i=k}^{K} b_i^+) y_j^*\}$ is violated. Note that to do so the rows of the path might have to be reformulated into single node flow sets as shown in the previous section. If the simple network inequality is violated, it is tried to generate an extended network inequality by adding variables $j \in N_k^-$ to Q_k^- if they increase the violation. For a variable $j \in N_k^-$ this is the case it $x_j^* > (\sum_{s=1}^{k} \sum_{i \in C_s^+} y_j^*) u_j$. If the simple network inequality from the current path is not violated, the algorithm continues by searching for the next row to add to the path.

Unfortunately, very little is published about the implementation of flow path cut generators. The reason for this might be that the impact of a flow path cut generator is in general very low. Flow path cuts are only found for a fraction of the mixed integer programming problems in the typical test sets (see the results in section 6.5). Nevertheless flow path cuts are very important to solve certain problem classes. A reason for this situation is that the structure utilized by the flow path cuts is very specific. On the contrary the effort needed to implement flow path cut generators is fairly high. We address these problems in section 5.6 by proposing a new family of valid inequalities that includes the flow path inequalities and by describing an easily implementable cut generator for them.

5. Implementations, Algorithmic Improvements, and New Algorithms

5.1. Objectives

5.1.1. Objectives of the Implementation

This section describes the implementation of the three aforementioned separation algorithms and two new ones. The priority objective of this implementation is to get *good* cut generators that have a strong positive impact on an MIP solvers performance. Characteristics of good cut generators are discussed in the next section.

Another objective of this implementation is to implement the cut generators in a way that supports a fair comparison between them. Therefore all cut generators are implemented in a framework that supports this objective in two ways. The first is that it provides all cut generator with the same input rows. The second is that it allows the cut generators to share procedures needed in more than one cut generator. A further advantage of this framework is that from a software design point of view it simplifies adding new cut generators and the maintenance of the existing ones. In some aspects our framework is similar to the one described in [30]. Its implementation aspects are discussed in section 5.2.

The implementation described here is done for the MOPS MIP solver and thus is specific to its computational environment. For example, we do not focus very much on limiting the number of violated cuts generated by our cut generators because the MOPS MIP solver in its current version uses a cut pool to select the presumably best cuts generated by several cut generators.

The MOPS MIP solver is a commercial product and hence has to guarantee a certain level of accuracy. On the other hand, practical problem instances contain many input values that are not exact but depend on forecasting or estimation. Thus concerning accuracy, the implementation described here is meant to be as accurate as the other parts of the MOPS MIP solver but in most situations does not make sacrifices for accuracy.

5.1.2. Characteristics of Good Cut Generators

As pointed out in section 2.4, the separation problem is to find a cutting plane in a family of valid inequalities $\mathcal{F}$ that cuts off a non-integral solution (x^*, y^*) or to proof that none exists. The objective of a theoretical separation algorithm is to solve the separation problem exactly or heuristically using as few operations as possible.

For implementations of separation algorithms, i.e. cut generators, used in an MIP solver, the objective is to support the MIP problem solving approach (see section 3.1) as much as possible. We summarize the qualities that lead to this in four *key characteristics for good cut generators*:

- quality,
- diversity,
- efficiency, and
- accuracy.

We now explain what we mean by these four words, argue why they represent key characteristics of good cut generators, and discuss how they can be evaluated.

By saying that a cut generator should be of high *quality* we mean that it generates cuts of high quality. We speak of high quality cuts if adding these cuts improves the formulation of a mixed integer programming problem significantly. A result of a significantly improved formulation can be that a branch-and-bound algorithm can solve the problem to optimality in less time. Another result might be that feasible solutions to the problem are found faster or that better feasible solutions are found. Measuring the quality of cuts constitutes a big problem. Therefore indirect measures have to be used. One possibility is to judge the quality of a cut generator, and thus also the quality of the cuts it generates, is to look at theoretical properties of the family $\mathcal{F}$ that the underlying separation algorithm searches. For some families it is known that they contain inequalities that are facet-defining for the convex hull of certain mixed integer sets. Generating facets of the convex hull of a problem or a structure within a problem should in general improve the formulation and lead to a speed up of the solution time. The problem with this approach is that theoretical properties of a family of valid inequalities might be misleading because, although certain valid inequalities might be facet-defining, they might not be necessary to solve practical problems.

Another aspect of the quality of the cuts generated by a cut generator is the extent to which the underlying separation algorithm searches the family of valid inequalities $\mathcal{F}$.

There are separation algorithms that are exact, this means they solve the separation problem to optimality. In other words, if there is a cut in family $\mathcal{F}$ that cuts off (x^*, y^*) the algorithm guarantees to find it. Most separation algorithms however are heuristic. They only search a part of the inequalities in a family and hence can not guarantee to find a cut if one exists. How large this part is, or in other words how extensive the search is, can have a major influence on the quality of a cut generator. Other methods for evaluating the quality of cut generators are to compare dual bounds after adding several rounds of cuts or to measure solution time for a set of test problem instances. These methods are discussed in depth in section 6.1.

One aspect of *diversity* is the ability of a cut generator to find cuts that are different from those cuts other cut generators find. Diversity is a goal that is important if several cut generators in an MIP solver are able to generate the same cuts. This can happen either if two cut generators search the same family of valid inequalities $\mathcal{F}$ or if the family used by one cut generator is a subset of the family used by another cut generator. It is clearly an advantage to have cut generators in an MIP solver that search different families of valid inequalities for cuts. Sometimes it might be a good solution to have one cut generator that searches a family of valid inequalities very broadly and one cut generator that searches a small part of this family very thoroughly.

Another aspect of diversity is that a good separation algorithm should find many cuts cutting off (x^*, y^*) instead of just a single one. The reason for this is that the chance to cut off an LP solution that otherwise would come up in the next round rises and hence less rounds might be needed to get to the same result. As resolving the LP relaxation sometimes is time-consuming, needing less rounds is advantageous from an efficiency point of view (see below). Then again an implementation should not generate too many cuts that are very similar so that the size of the model does not explode without a large improvement of the formulation. Cut selection and management as described in section 3.3 and in [96] can compensate for this. Ceria mentions in [28] that adding several cuts each round was one of the reasons behind the computational success of Gomory mixed integer and lift-and-project cuts.

A third aspect of diversity is that a cut generator should find cuts for many different problem classes. Typically MIP solvers are designed to be tools for MIP problems in general. Therefore it is not advisable to implement separation algorithms that generate cuts just for a very specific problem class. Whether or not an implementation of a separation algorithm finds cuts for many different problem classes can depend on the family $\mathcal{F}$, the way the family is searched and even on implementation details.

As the time spent in cut generators contributes directly to the total runtime of the MIP solver, they obviously should not run for a very long time. On the other hand, spending some time to find a cut that speeds up the solution process afterwards is worthwhile. So by *efficiency* it is meant that time is only spent if it pays off. For many problem instances separation algorithms based on mixed integer row relaxations run very fast because they usually only work with the sparsely filled rows of the constraint matrix. By controlling the rows of the matrix used, the runtime spent can typically be controlled very good. As the problems MIP solvers have to solve get larger and larger, efficiency gets a more important aspect. Note that there is also the aspect of memory efficiency; but with the increasing availability of 64-bit machines this gets less important. Measuring the efficiency is typically best done by comparing the overall runtime of the MIP solver or the runtime of certain parts of the solver. Whenever these times are compared the trade-off between efficiency and quality has to be considered.

The *accuracy* of cut generators is closely connected to the accuracy of MIP solvers. Surprisingly few publication deal with either of these topics (see [71], [9] and [81]). A cut generator that is not accurate enough might generate a cut that cuts off a feasible solution of the MIP problem. Reasons for this are bugs in the implementation or the use of floating point arithmetic. Bugs can typically be fixed as soon as they are encountered but floating point arithmetic stands in the way of totally accurate cut generators. Modern MIP solvers use floating point arithmetic because it is much faster than using exact arithmetic. This is for example confirmed by Applegate et al. in [9], and Fukasawa and Goycoolea state in [46] that they observed exact arithmetic to be 100 times slower than floating point arithmetic. As pointed out in computer science literature (for example in [47]), implementing a robust algorithm using floating arithmetic holds some challenges. One way of dealing with the problems floating point arithmetic produces is to use tolerances. MIP solvers typically have a large number of tolerances for all aspects of their computations. These tolerances are set to certain default values. These default values typically work well unless one tries to solve numerically difficult problem instances. Usually the user can change the tolerances to improve the overall accuracy of the solver. Normally it is not necessary to have totally accurate cut generators, they just need to be accurate enough to not jeopardize the overall accuracy of the solver and their accuracy should be adjustable by the user. In Section 5.2 we show techniques to improve the accuracy of cut generators and in section 6.1 we propose a simple method for testing whether cut generators are good enough.

Obviously there is a trade-off between the several characteristics of cut generators. It is important to understand that a good cut generator does not solely depend on the underlying separation algorithm or the family of valid inequalities searched by the separation algorithm. Different implementations of the same algorithm can produce very different results depending on implementation details such as the exact way heuristic decisions are made or the data structures used.

5.2. Framework

5.2.1. Overview

In this section we describe the framework build around the cut generators implemented for this thesis. Algorithm 1 shows the main procedure of the framework. This procedure is intended to be called by an MIP solver in each round of the cut generation. In the main loop (lines 3 – 27) it runs through a set of usable rows that has been identified in line 2. The procedure in line 2 also identifies variable bounds and other row types. See subsection 5.2.4 for details. In the *aggregation loop* (lines 6 – 26) the framework stores the rows of a path in the set *path* and the sum over the rows in the aggregated row *aggRow*. The parameter `xmxagg` limits the maximal length of paths considered and is set to 6 by default. More about aggregation and paths can be found in section 5.2.5. In lines 4 and 21 a slack variable is added to the current base row so that in the *reversing loop* (lines 7 – 18) first the original and then the reversed row can be passed on to the separation algorithms. It is a design principle of our implementation that for each original input row (each row in the set of usable rows) at most two cuts are generated by each cut generator, one from the original and one from the reversed row. The reason for this is that we want to avoid cluttering up the cut pool with too many similar cuts.

The separation algorithms implemented in this framework can choose to use the set of rows *path* or the aggregated row *aggRow* as an input. Note that in this framework each algorithm can have preconditions under which it is called. The sections about the individual separation algorithms discuss appropriate preconditions. Also note that if a separation algorithm reports that it has found a cut for the current starting row it is not called again in later iterations of the aggregation loop. If all active separation algorithms have found a cut the framework moves on to the reversed row or to the next row in the set of usable rows.

Algorithm 1 A framework for the separation algorithms

```
procedure MIXED_INTEGER_ROW_CUTS
    FIND_VBS_AND_ROW_TYPES(vub, vlb, usableRows)            ▷ See section 5.2.4
    for row ∈ usableRows do
        aggRow ← row + slack
        path ← path ∪ row
        for k = 1 . . . xmxagg do
            for s ∈ {1, −1} do
                aggRow ← aggRow · s
                if conditionForAlgorithm1 = TRUE then
                    SEPARATION_ALGORITHM_1(aggRow|path, . . . )
                end if
                if conditionForAlgorithm2 = TRUE then
                    SEPARATION_ALGORITHM_2(aggRow|path, . . . )
                end if
  ⋮
                if conditionForAlgorithmt = TRUE then
                    SEPARATION_ALGORITHM_t(aggRow|path, . . . )
                end if
            end for
            nextRow ←FIND_NEXT_ROW(aggRow)                  ▷ See section 5.2.5
            if nextRow ≠ ∅ then
                aggRow ← aggRow + nextRow + slack
                path ← path ∪ nextRow
            else
                exit loop
            end if
        end for
    end for
end procedure
```

5.2.2. Data Structures

In this subsection we briefly discuss the data structures used in the implementation of the framework and the separation algorithms. First we look at the input data structure of our framework, the constraint matrix. It is important to mention that the MOPS MIP solver provides two representations of the matrix, an indexed row-wise representation and an indexed column-wise representation. Indexed means that only non-zero elements are stored. This enables us to run over the elements of a row or a column in nz steps, where nz is the number of non-zero elements in a row or column instead of needing n or m steps, which is the size of the full matrix. Before the start of each round of the cut generation MOPS updates the two representations automatically. Some operations in the framework and the cut generators can be sped up by using the appropriate representation.

The data structures used in the cut generators mainly store vectors that represent rows or cuts. For an input constraint matrix A with n columns and m rows these vectors are typically of size $n+m$. The first n elements represent the structural variables of the MIP problem and the last m can hold information about slack variables.

We take a look at three ways of storing these vectors:

1. dense,
2. packed,
3. and indexed.

Dense means that we use a single array of size $n+m$ in which we store all elements directly. This is simple and only needs one array of size $n+m$ but working with the elements of the vector requires to run through all $n+m$ elements. If only a few of the elements are non-zero, as it is usually the case in the rows and columns of mixed integer programming problems, it is more efficient to only store the non-zero elements.

One way to do this is in a packed data structure. In a packed data structure we use one array of size $n+m$ as a stack to store the indices of the non-zero elements. In a second array of size $n+m$ we then store the corresponding value of the element in the same position as in the first array. This has the advantage that we can read and write the array with a loop that runs from 1 to nz, the number of non-zero elements, instead of 1 to $n+m$. A disadvantage of this data structure is that we need two arrays and that operations like adding two vectors together are complicated to perform.

The third option to store a vector is to store it indexed. This data structure needs three arrays of size $n+m$. The first is used as a stack to store the indices of the non-zero

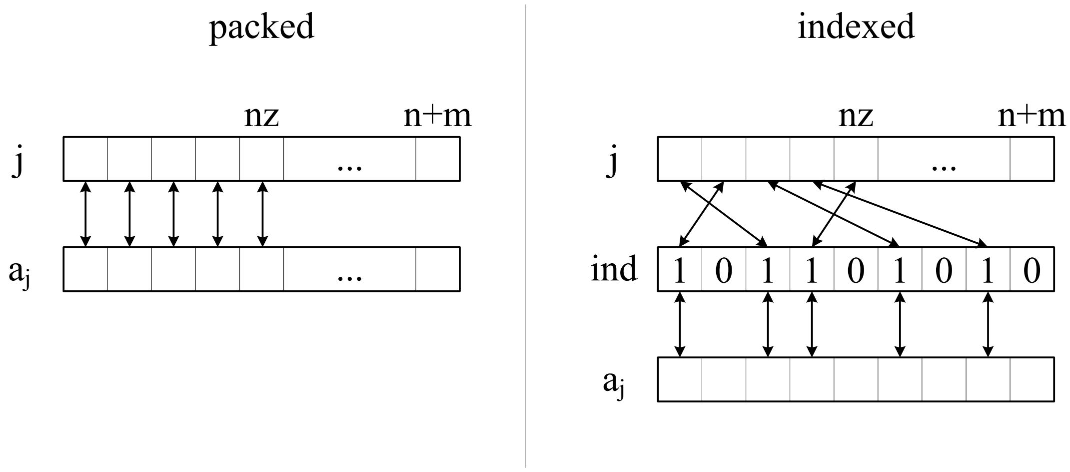

Figure 5.1.: Data structures to store vectors.

elements of the vector. The second array marks at the index position of an element whether it is non-zero or not. The third array finally stores the value of the element also at the position of its index. The indexed data structure combines advantages of the dense and the packed data structure at the cost of needing an additional array. Figure 5.1 illustrates the three different data structures.

In our implementations we use packed data structures for storing vectors such as generated cuts or reformulated rows. We use indexed data structures as intermediate data structures because some operations can be done significantly faster with them. This includes for example adding elements to a vector from a non-sorted input or adding two vectors together. In these operations the situation can occur that we want to add two non-zero elements together. In a packed data structure we would need to run through the stack to find the corresponding coefficients. In an indexed data structure we can use the indicator array to check whether an element is non-zero and then add the two values of the elements together. Using the indicator array instead of directly accessing the array with the value of the element avoids numerical problems and prevents situations where an element appears twice in the stack array.

Besides the need for an additional array, there is another important drawback of indexed data structures. In order to have them work correctly the indicator arrays have to be zeroed out in their full length, in our case $n + m$, after each use. For large problem instances this zeroing out might take a significant amount of time, especially if performed very often. Therefore we only zero them out in the full length at the beginning of the main procedure of the framework and afterwards use the stack to set the changed values back to zero after use.

5.2.3. Accuracy

As pointed out in section 5.1, accuracy is one of the key characteristics of a good cut generator. As the overall accuracy of the solver depends on the accuracy of its components it has to be made sure that the cut generators are at least as accurate as the other components of the solver. Our cut generators are not totally accurate because they use floating point arithmetic.

In cut generators several aspects of floating point arithmetic can interfere with the accuracy. One is that testing two floating point numbers for equality has to be done using a tolerance. This also includes the problem of checking whether a number is integer or not. Another problem is that subtracting two numbers of the same value can yield a result that is not exactly zero. Finally rounding or cut-off errors can occur if dealing with very small and very large numbers in the same operation.

In our implementation two measures are taken to improve the accuracy of the cut generators. The first measure is that all cut generators are implemented in a way that the cuts generated stay in the scale of the input row. This results in cuts where those coefficients that are not modified by the cut generation stay the same. Example 5.1 illustrates this using the simple MIR inequality introduced in section 2.5.

Example 5.1. *Assume we want to generate a valid inequality for the row*

$$x + 10y \geq 5 \tag{5.1}$$

using the simple MIR inequality (see section 2.5). As the right hand side is not fractional we divide the row by 10*. The result is the row* $0.1x + y \geq 0.5$ *with the simple MIR cut*

$$0.1x + 0.5y \geq 0.5. \tag{5.2}$$

This cut is not in the scale of the input row, because the coefficient of x *has changed although it is not part of the actual cut generation. If we multiply the resulting cut by 10 we get*

$$x + 5y \geq 5 \tag{5.3}$$

which is a much nicer and potentially more accurate cut than (5.2)*. In the definition of the cMIR inequalities in section 4.2 we already included this rescaling.*

The second measure to improve the accuracy is that all cuts generated by all separation algorithms are *cleaned* using the same methods. The first of these methods is to remove *quasi-zero* coefficients in the cuts. A coefficient is *quasi-zero* if it is smaller than the MOPS tolerance for elements in the matrix `xdropm` which by default is 1×10^{-6}. If a *quasi-zero* coefficient is detected, the cleaning method tries to eliminate it by substituting its lower or upper bound, i.e., relaxing the cut. If this is not possible, the cut is rejected and not added to the cut pool. Besides this, the cut cleaning also removes cuts which contain coefficients larger than $\frac{1}{\texttt{xtolin}}$. These two methods together can be seen as an approach to control the *dynamism* (as defined in [71]) of the resulting cuts. Dynamism is defined as the ratio between the smallest and the largest coefficient in a row.

5.2.4. Variable Bounds and Row Types

In our implementation the framework identifies variable bounds and decides on a set of usable input rows to be used by the cut generators. Variable bounds are mixed integer constraints of the form

$$x_j \leq u_j y_j,$$

which we call a *variable upper bound*, or

$$x_j \geq l_j y_j,$$

which we call a *variable lower bound*. In both cases we assume that $x \in \mathbb{R}$ and $y \in \mathbb{Z}$. If $y \in \{0,1\}$ in a variable lower or upper bound constraint we call it a *binary variable lower (or upper) bound*. Variable bounds are stored in a special data structure and derived directly from rows of the constraint matrix that fit their definition. Additionally, variable upper bounds are derived from rows of the form

$$\sum_{j \in N} x_j \leq uy.$$

Furthermore, the framework also identifies binary variable bounds on general integer variables like

$$z_j \leq u_j y_j$$

where $z \in \mathbb{Z}$ and $y \in \{0,1\}$. It only does so if z_j does not appear in a variable bound on any other variable. The detection of variable bounds is also used to strengthen the bounds of continuous or general integer variables. If several variable upper (or lower) bounds are identified the framework stores the one that is tightest in the current LP solution.

Rows of the constraint matrix that are variable bounds are excluded from the set of usable rows. Other reasons to exclude rows from the set of usable rows are:

1. rows that were deactivated in LP preprocessing,
2. ranged rows,
3. and rows with more than `xmxmic` variables.

By setting an upper bound on the number of variables in a row we avoid an increased runtime of the cut generators for instances with very long rows. The default value for `xmicuc` is 500, and for many instances no rows are excluded with this value. For some other instances dropping long rows is crucial for fast cut generation. The reason for this is that for the cut generators described in this thesis, the runtime directly correlates with the number of variables in the input rows. As cuts with many elements are typically not wanted, skipping these rows most of the time does not influence the performance of the solver.

5.2.5. Aggregation and Path-finding

It is easy to see that the aggregation used in the cMIR cut separation algorithm and the path-finding used in the flow path cut separation algorithm are very similar. In fact in [70] it is stated that the aggregation heuristic is essentially the same as the path-finding procedure of the flow path cut separation algorithm. Both procedures select one row after the other by first identifying a variable for elimination and then finding a row which can be used to eliminate the selected variable. The difference between the two procedures lies in the way the variable for elimination is selected. The path-finding procedure selects the variable with the largest outflow, i.e. $j \in N$ such that $a_j < 0$ and x_j^* is maximal. The aggregation heuristic chooses the variable with the largest distance to its bounds. In many cases these two methods result in the same variable to be chosen, but this is not always the case.

In our implementation we want to use one aggregation/path-finding method for all algorithms. Note that aggregated rows could also be used as an input for a flow cover cut generator but that we do not do so in the default version of our flow cover cut generator. One problem with the design decision to use the same method for cMIR and flow path cuts is that we loose some diversity. Obviously, if two different methods are used, two different paths are investigated leading to more diverse cuts. Another problem is that, although the methods are very similar, they pursue slightly different goals. The path-finding procedure's only goal is to identify fixed charge paths in the constraint matrix.

The cMIR aggregation actually pursues two goals. The first is also to identify paths, this is for example helpful for lot-sizing instances. We show this in example 5.2. The second goal of the aggregation heuristic is to make use of more complex bound structures in the generation of the mixed integer knapsacks. We illustrate this in example 5.3.

Example 5.2. *Assume the following constant capacity lot-sizing problem*

$$\begin{aligned} s_1 + x_1 - s_2 &= 2 \\ s_2 + x_2 - s_3 &= 4 \\ s_3 + x_3 - s_4 &= 5 \\ x_j &\leq 10y_j \text{ for } j = 1 \ldots 3 \\ x &\in \mathbb{R}^3_+ \\ s &\in \mathbb{R}^4_+ \\ y &\in \{0,1\}^3. \end{aligned}$$

For cutting off the fractional point $(x, s, y) = (10, 1, 0, 0, 8, 5, 0, 1, 0.1, 0)$ we can first replace all variable bounds and then try the MIR inequality for each row alone using $\delta = 10$. These inequalities are

$$s_1 + 2y_1 \geq 2 \qquad s_2 + 4y_2 \geq 4 \qquad s_3 + 5y_2 \geq 5.$$

None of these valid inequalities cuts off the point. If we aggregate the first two rows and again use $\delta = 10$ we get the valid inequality

$$s_1 + 6y_1 + 6y_2 \geq 6$$

which also is not violated. After substituting all variable bounds and aggregating all three rows we get the base row

$$s_1 + 10y_1 + 10y_2 + 10y_3 \geq 11$$

with the MIR inequality (using $\delta = 10$)

$$s_1 + y_1 + y_2 + y_3 \geq 2$$

which cuts off the fractional point. Note that in this case the variables selected for elimination s_2 and s_3 are both the largest outflow variables and the variables furthest from their bounds.

Example 5.3. *Assume we have the following structure that is part of a bigger production planning problem*

$$\begin{aligned} x_1 + x_2 + x_3 &\geq 14 \\ x_2 \leq 3 + 8y_1 + 10y_2 + x_4 & \\ x &\in \mathbb{R}^4_+ \\ y &\in \{0,1\}^2. \end{aligned}$$

The first constraint of this structure means that the sum of the production of three production lanes x_1, x_2, and x_3 has to exceed a demand of 14. The second constraint is a more complex version of a variable upper bound constraint. Two measures denoted by y_1 and y_2 can be used to increase the initial capacity of the machine, that is 3, by exactly 8 and/or 10 production units. It is also possible to increase the maximal production of the machine by a customary amount which is represented by the variable x_4. In a practical model y_1 and y_2 might stand for machines used in the production lane and x_4 might denote additional workers assigned to a lane. Note that there are different ways of modeling this situation but we assume that a user has chosen this one.

If we now want to cut off the fractional point $(x, y) = (0, 14, 0, 0, 1, 0.3)$ we see that from the first row alone no useful MIR inequality can be generated. By aggregating the two rows using x_2 as the variable to eliminate and the scaling factor -1 we can generate the reversed mixed integer knapsack

$$8y_1 + 10y_2 + x_1 + x_3 + x_4 \geq 11$$

for which the MIR inequality with $\delta = 10$ is

$$x_1 + x_3 + x_4 + y_1 + y_2 \geq 2$$

which cuts off the fractional point.

Despite the different goals of the methods, we want to use the same aggregation/path-finding method for all cut generators because of two reasons. The first is to support the fair comparison between the cut generators. The second is that the efficiency of our implementation can be increased by only generating the path once instead of separately for each cut generator.

Putting more complex bounds into a path to be used by a flow path cut separation algorithm will likely not result in the generation of good cuts. But as the cMIR cut generator in general is considered the more important cut generator one suggestion is to use its method for the aggregation/path-finding in the framework. Algorithm 2 shows how the crucial method to find the next row in the aggregation is implemented in the framework.

Algorithm 2 The Traditional Aggregation Strategy

```
procedure FIND_NEXT_ROW(aggRow)
    SORT_VARIABLES_BY_DISTANCE(aggRow)
    for j ∈ N do
        for row ∈ usableRows do
            r ← −a_jr / a_jt
            if ROW_SELECTABLE(row,r) then
                return row · r
            end if
        end for
    end for
    return ∅
end procedure
```

Note that, instead of just using the variable with the largest bound and then searching for a row with this variable, we sort the variables and then try to find a row for aggregation in the order of decreasing distance. Therefore we can find a row even if there is no selectable row for the variable with the largest distance.

To decide whether a row is selectable the following conditions are checked. First we do not want to have the same row twice in the path, this is checked by marking rows that already are in the path. Because we replace them in the bound substitution we do not want to aggregate variable bound rows, the same holds for extended bound rows defined in the next section. Furthermore we want to limit the length of the rows we work with, as it is already done when deciding on the usable rows. Therefore it is checked whether the new row added to the existing row has more variables than the parameter `xmxmic`, which by default is 500. If this is the case the row is not considered selectable. Finally we want to make sure that due to the rescaling of the cut we do not end with a row that contains bad coefficients, so we limit the rescaling factor s to be between `xdropm` and $\frac{1}{\texttt{xdropm}}$, where `xdropm` is the MOPS parameter that specifies the smallest value a coefficient of the constraint matrix is allowed to have (by default 1×10^{-7}).

The decision which of the rows to add, if there are several, is done as suggested by [48]; the first one that is selectable is used. The possible rows are identifyed efficiently using the column-wise representation of the constraint matrix. This method works quite well in identifying paths in lot-sizing instances because typically there is only one row in the original constraint matrix that contains a stock variable (see example 5.2). It is also very fast. In more complex situations, where cuts have been added and are considered as rows in the path, it might, depending on the ordering of the rows, happen that a row is selected that does not belong to the path the algorithm should find.

To overcome the drawback that the cMIR aggregation is not only tailored towards finding paths, we suggest a new method that together with the extended bound substitution described in the next section solves this problem to a certain point. We call it the path-based tightest row aggregation and it is shown in algorithm 3.

Note that this method tries to select a variable with a negative coefficient and without variable bounds. Thus its priority lies in finding fixed charge paths. If no path structure is identified it still might aggregate other rows. Because this aggregation is not as good in finding more complex bound structures using it without the extended bound substitution (described in the next section) is likely to result in a worse performance for some instances.

Another improvement of the path-based tightest row aggregation is that we do not choose the first row but the *tightest*, i.e. the row where, concerning the current LP relaxation solution, the difference between the left hand side and the right hand is minimal. The tightness of a row is represented by the *slack*. The slack is the value of a slack variable that is added to a row to make it an equality row. Note that it is very likely that we find a row with a slack of zero and therefore the procedure is speed-up very much by stopping if one of these is encountered. Also note that again the column-wise representation of the constraint matrix is used to identify candidate rows quickly. Selecting the tightest row also helps to find the paths in a lot-sizing problem because we want to select equality rows. The slacks of the rows for the current LP relaxation solution are stored in MOPS and hence do not have to be computed. In section 6.5 we perform several computational experiments to investigate the different aggregation/path-finding methods.

5.2.6. Bound Substitution

Although bound substitution is only used in one of the separation algorithms described in chapter 4 we discuss it as part of the framework because it is also used in the new mixing-based cut generators presented in section 5.6. The task of the bound substitution

Algorithm 3 The Path-based Tightest Row Aggregation Strategy

```
procedure FIND_NEXT_ROW(aggRow)
    call SORT_VARIABLES_BY_DISTANCE(aggRow)
    minSlack ← ∞
    nextRow ← ∅
    for j ∈ N  and  a_j < 0 and j has no variable bounds do
        minDual ← ∞
        for row ∈ usableRows do
            r ← −a_jr / a_jaggRow
            if t*_row < minSlack and ROW_SELECTABLE(row,r) then
                nextRow ← row · r
                minSlack ← t*_row
                if minSlack = 0  then  return nextRow
            end if
        end for
    end for
    for j ∈ N  and  a_j < 0 do
        for row ∈ usableRows do
            r ← −a_jr / a_jaggRow
            if t*_row < minSlack and ROW_SELECTABLE(row,r) then
                nextRow ← row · r
                minSlack ← t*_row
                if minSlack = 0  then  return nextRow
            end if
        end for
    end for
    for j ∈ N do
        for row ∈ usableRows do
            r ← −a_jr / a_jaggRow
            if t*_row < minSlack and ROW_SELECTABLE(row,r) then
                nextRow ← row · r
                minSlack ← t*_row
                if minSlack = 0  then  return nextRow
            end if
        end for
    end for
    return nextRow
end procedure
```

heuristic in this implementation is to transfer a general mixed integer row of the form

$$\sum_{j \in N} a_j x_j + \sum_{j \in P} g_j y_j = b \qquad x \in \mathbb{R}^{|N|}, \quad y \in \mathbb{Z}_+^{|P|}$$

into a reversed mixed integer knapsack of the form

$$\sum_{j \in I} u_j y_j + s \geq b \qquad y \in \mathbb{Z}_+^{|I|} \quad s \in \mathbb{R}_+.$$

Note that here, in contrast to other publications about the cMIR cut separation algorithm, we generate reversed mixed integer knapsack sets. The reason for this is that the cMIR cut separation algorithm in this implementation generates reversed cMIR cuts and the new mixing-based cut generators also use reversed mixed integer knapsack sets.

The simple bound substitution implemented in this framework uses variable lower ($x_j \geq l_j y_j$) and upper ($x_j \leq u_j y_j$) bounds. Based on the first rule by Marchand and Wolsey [70] it decides for each continuous variables whether it should be replaced by its lower or upper bound. The rule states that a variable is replaced by its closest bound, i.e., it uses the lower bound only if $x_j^* - l_j y_j^* \leq u_j y_j^* - x_j$. In our implementation we changed it slightly to use the lower bound only if $x_j^* - l_j y_j^* < u_j y_j^* - x_j$. The only difference is that if the distance to the lower and the upper bound is the same we use the upper bound. Note that in the case of a static bound $y_j^* = 1$ and that in the very common situation that $l_j = 0$ replacing a variable by its lower bound just means not replacing it at all. If the bound selected is in fact a variable bound, it substitutes the variable bound constraint and adds a slack variable. If the bound is a static bound, it also adds a slack variable and then modifies the right hand side. After this process all continuous variables have been replaced by slack variables and hence have a lower bound of 0. Thus, even if a continuous variable has a lower bound of less than 0, in contrast to the flow cover cut generator the row can still be used to generate a cut.

Besides this simple bound substitution we suggest an algorithmic improvement to this part of the cMIR cut separation algorithm. This improvement is connected to an observation in the previous section. There we pointed out in example 5.3 that the aggregation besides trying to find paths also is used to incorporate information about more complex bounds than the typically used variable bound constraints. Here we now define a class of constraints frequently found in practical mixed integer programming models and call this class *extended bound constraints*.

Definition 5.1. *Extended bound constraints are of the form*

$$x \begin{array}{c}\leq\\=\\\geq\end{array} b + \sum_{j \in I} u_j y_j + ks \qquad x, s \in \mathbb{R}_+, y \in \mathbb{Z}_+^{|I|}.$$

We call an extended bound constraint

- static *if* $b > 0$, $|I| = 0$, *and* $k = 0$,
- variable *if* $|I| = 1$ *and* $k = 0$,
- soft *if* $k > 0$,
- raised *if* $b > 0$,
- additive *if* $|I| > 1$,
- binary *if* $y_j \in \{0, 1\}$ *for all* $j \in I$.

Extended bound constraints appear in some mixed integer programming problem instances because they can be used to model certain real world situations. An additive extended bounds can be used, for example, to model upper bounds of production variables that depend on a set of machines to be chosen. Additive extended bounds were already studied by Atamtürk et al. in [13]. Soft extended bounds are typically used to model real world situation were it is possible to exceed a certain bound for some additional cost. Soft bounds are for example used in a practical MIP model for optimizing a semiconductor supply chain in [40].

The idea of the extended bound substitution heuristic we suggest is to still use variable bound constraints but in addition to this, store information about extended bound rows in the matrix in connection to the continuous variables x_j. This makes it possible to check after the normal bound substitution decision is made whether an extended bound constraint exists that is tighter than the variable or static bound selected. By doing this we can overcome the drawback of the path-based tightest row aggregation described in the last section and have successfully separated the two tasks of the aggregation into two steps by moving the usage of more complex bound structures into the bound substitution step. Note that a cMIR cut generator that does not use extended bound constraints still might generate cuts based on them implicitly. The method suggested here simply tries to make this process explicit and less depending on hidden decisions.

Another algorithmic improvement of the extended bound substitution is that general integer variables are also considered to be substituted. If a binary variable bound for a general integer variable exists, we treat it like a continuous variable. Thus it is possible to generate cuts for problem instances where we have a typical mixed integer programming model with integer instead of continuous variables.

Besides these large differences there is an implementation detail that we improved in the extended bound substitution. It concerns the bound substitution decision whether to replace the lower or upper bound. The rule described above makes some sense in many cases but is not very helpful in the very common situation that $x_j^* = y_j^* = 0$. We therefore extended the rule to cover this special situation separately. The rule we suggest is based on the coefficient in the objective function for the involved variables because these values give an impression of the importance of the variable in future rounds of the cut generation. The extension of the rule says that if $x_j^* = y_j^* = 0$, we replace the lower bound if $-a_j c_{vlb} < a_j c_{vub}$. Note that the $c_{vlb} = 0$ and $c_{vub} = 0$ if there is no variable bound. In section 6.4 we evaluate implementation details of the bound substitution step and compare the simple and the extended bound substitution.

5.3. The Flow Cover Cut Generator

In this section we describe our implementation of the flow cover cut separation algorithm which we presented in section 4.1. Algorithm 4 gives an overview of the program flow of the cut generator. In the following we discuss these steps in detail.

Algorithm 4 The Flow Cover Cut Generator

procedure FLOW_COVER_CUT_GENERATOR(*row*)
    if CONTAINS_VARS_LESS_THAN_ZERO(*row*) **then return** $\emptyset$
    if NO_FRACTIONAL_BINARY(*row*) **then return** $\emptyset$
    SET_REFORMULATION_STATUS(*row*,*refSta*)
    $C \leftarrow$ FIND_COVER(*row*,*refSta*)
    cut $\leftarrow$ GENERATE_CUT(*row*,*refSta*,C)
    finalCut $\leftarrow$ CLEAN_CUT(*cut*)
    return *finalCut*
end procedure

Note that this cut generator is called within the framework described in section 5.2 and therefore it is called for each usable row and the corresponding reversed row. Thus it implicitly also generates flow pack cuts. Depending on a parameter it is possible to call it for all aggregated rows or just the first row of a path. It is also possible to decide whether to call it for original rows only or for all rows including cuts generated in earlier rounds.

The first step of the cut generator is to check whether the current input row has variables with a lower bound of less than zero. If this is the case the cut generator exits. Although it would be possible to replace the variable by two new variables with a lower bound of 0 because of restrictions in our data structures we do not do so. Another aspect that is checked is whether the row contains a fractional binary variable or if one of the variable upper bound binary variables connected to the row is fractional in the current LP solution. If this is not the case it is unlikely that a violated cut can be found and the cut generator therefore does not try to. This results in an improvement of the efficiency of the cut generator and although this theoretically might lead to a worse quality of the cuts, computational experiments revealed that it typically does not.

The second step is to decide on how each variable is treated in the reformulation of the input row. Instead of performing the reformulation and storing the result in a separate data structure, in our implementation we only identify the type of reformulation to perform for a each variable and with which other variable it forms a pair. Using this information stored in the *reformulation status* ($refSta$) we compute necessary values when needed. Although the flow cover cut separation algorithm requires mixed 0-1 rows as input, we also use rows with general integer variables. These rows are relaxed by treating general integer variables as continuous variables. As pointed out in section 5.2 we also identify variable bounds for general integer variables and these are also used in the reformulation.

Besides the fact that general integer variables are treated as continuous variables the reformulation for each variable is done as described in the section 4.1. The case where either the continuous or the binary variable of a variable upper bound pair appears is straight forward. The situation where both variables of a pair appear in the same row requires some considerations. The first thing is that we only group the variables together if their coefficients have the same sign. This simplifies the decision whether a pair belongs to N^+ or N^-. The next consideration is that, if several variables share the same variable bound, because of our data structures, we can only use it in a pair with one of them. We choose the first continuous variable we find as partner for the binary variable and handle all remaining variables as if they were alone. In the section 4.1 we show how rows with *bounded* variables can be reformulated as single node flow sets. Although the precondition that the variables are bounded is necessary for this reformulation we can generate flow cover cuts from rows with unbounded variables. In the SGFCIs and LSGFCIs the upper bounds on the variables are not needed as long as they are not in the generalized cover

C or in the set L^-. Therefore, if we choose unbounded variables not to be in the cover C and not in L^-, we can use rows with unbounded variables by assuming they have a very large upper bound M. For slack variables we do the same in the sense that we simply avoid to use them in the cover or in L^- and hence relax them if they have a positive coefficient and use them in L^{--} if not. Example 5.4 shows how a non-trivial row can be reformulated.

Example 5.4. *Assume the mixed-integer row*

$$\begin{aligned} 2x_1'-x_2'+x_3'+x_4'-y_2+4y_5 &\leq 9 \\ x_1' &\leq 5y_1 \\ x_2' &\leq 4y_2 \\ x_3' &\leq 10 \\ x' &\in \mathbb{R}^4_+ \\ y &\in \mathbb{B}^3. \end{aligned}$$

If we reformulate this row into a single node flow set we get

$$\begin{aligned} x_1-x_2+x_3+x_4+x_5 &\leq 9 && \\ x_1 &\leq 10y_1 && x_1=2x_1' \\ x_2 &\leq 5y_2 && x_2=x_2'+y_2 \\ x_3 &\leq 10y_3 && y_3=1 \\ x_4 &\leq My_4 && x_4 \notin C^+, y_4=1 \\ x_5 &\leq 4y_5 && x_5=4y_5 \\ x &\in \mathbb{R}^5_+ && \\ y &\in \mathbb{B}^5. && \end{aligned}$$

The next step is to find the generalized cover C. This is the most important step of the flow cover cut generator because the quality and the speed of the cut generator greatly depend on it. As mentioned in section 4.1, the cover is found by solving the cover finding knapsack problem (see page 41). To do so we first have to transform it into the standard form for binary knapsack problems as described in [72]. This is done by reversing the constraint, reversing the objective function, and complementing variables with negative knapsack coefficients. Furthermore we also use a small number ε to get a less than instead of a less than or equal constraint. By ε we can control the smallest value we allow for λ. We use the tolerance parameter `xtolin` of MOPS for this purpose that by default is

1×10^{-5}. The result is the transformed flow cover finding knapsack problem:

$$\begin{aligned}
\max \sum_{j \in N^+} (1 - y_j^*)\bar{k_j} + \sum_{j \in N^-} y_j^* k_j \\
\sum_{j \in N^+} u_j \bar{k_j} + \sum_{j \in N^-} u_j k_j \leq b - \varepsilon + \sum_{j \in N^+} u_j \\
\bar{k_j} = 1 - k_j \\
k \in \{0,1\}^{|N|}.
\end{aligned}$$

Before using an algorithm on this problem, we can preprocess it by setting $k_j = 0$ or $\bar{k_j} = 0$ if $u_j > b - \varepsilon$. As mentioned before we deal with unbounded and slack variables by setting $k_j = 0$.

The papers about the flow cover cut generators ([93] and [51]) report that a heuristic was used to solve the cover finding knapsack problem. A simple greedy heuristic based on sorting and adding variables one by one is the typical approach for this. A description of such a primal heuristic for the 0-1 knapsack problem can be found in [80], p. 452. A different approach is to use a specialized branch-and-bound method as for example described by Martello and Toth in [72]. We use such an algorithm that is already implemented in the MOPS MIP solver and also used for the cover cut generator.

Another implementation detail connected with the cover finding is how to deal with the variables that do not have a binary variable upper bound in the flow cover finding knapsack problem. Note that in the reformulation we assume that $y_j^* = 1$ for these variables. This is not a good choice when deciding on the generalized cover because the deduction of the flow cover finding knapsack problem is based on the assumption that $x_j = u_j y_j$. Therefore using $y_j^* = \frac{x_j^*}{u_j}$ should work much better. This is confirmed in the computational results shown in section 6.3.

In the cut generation step for each variable the corresponding coefficient in the cut and, where necessary, the coefficient of the variable's upper bound variable are computed and added to an indexed data structure. This allows a fast cut generation even if several variables have the same variable upper bound. For the sets C^+ and C^{++}, computing the coefficients is straight forward. For the partition of N^- into L^- and L^{--} the rule that leads to the most violated cut is already mentioned in section 4.1. The rule is actually not clear for the very frequently happening case that $x_j^* = y_j^* = 0$. In this case it makes no difference concerning the violation of the cut whether the rule $L^- = \{j \in N^- : \lambda y_j^* < x_j^*\}$ or $L^- = \{j \in N^- : \lambda y_j^* \leq x_j^*\}$ is used. But it makes a difference for the strength of the cut in later rounds of the cut generation. Note that here we use $y_j^* = 1$ for variables without a variable upper bound. In section 6.3 we computationally compare these two versions of the rule and the result is that we use the first rule. This means that we only use the binary variable upper bound if it really results in a more violated cut.

For the set $N^+ \setminus C^+$ the LSGFCIs suggest that we have to lift all variables in this set. But actually we can decide to relax these variables before generating the cut, that means we only lift those variables $j \in N^+ \setminus C^+$ with coefficients (α_j, β_j) that improve the violation of the cut, i.e. if

$$\alpha_j x_j^* - \beta_j(1 - y_j^*) > 0.$$

For the set C^- the lifting function g has to be computed in a way that it results in the best possible cut coefficient, i.e., that g is maximal.

Accuracy is typically not a big problem for flow cover cuts. One situation where inaccuracies may occur is when λ is a very small number. This can be avoided by setting the ε in the reformulated flow cover finding knapsack problem to a value larger than the smallest value allowed as cut coefficient. This is done in our implementation where the smallest number allowed in the cut is the MOPS parameter `xdropm` (by default 1×10^{-7}) and ε is `xtolin` (by default 1×10^{-5}). Nevertheless the accuracy safeguards as described in section 5.2 are applied to the generated cut before adding it to the cut pool.

Finally, we would like to point out an observation that might help to understand why flow cover cut separation algorithms are so successful in MIP solvers. It is possible to derive the valid inequality for the simple MIR set (see section 2.5) where y is binary using the flow cover inequality. To do so we reformulate this simple MIR set

$$y - x \leq b, \qquad 0 < b < 1, \qquad x \in \mathbb{R}, y \in \{0, 1\}$$

by assuming a very large bound M on x and introducing a variable $y' = 1$ to get

$$y - x \leq b, \qquad x \leq My' \qquad y \leq 1y$$

which is a binary single node flow set. Using y as generalized flow cover resulting in $\lambda = 1 - b$ we get the SGFCI

$$y + b(1 - y) \leq b + x.$$

After rewriting, this results in the simple MIR inequality $y \leq \frac{x}{1-b}$. This means that a cutting plane procedure based on reformulation and flow cover cuts can generate some problem specific cuts for mixed 0-1 problems in the same way the cMIR procedure can for general mixed integer problems. As most practical mixed integer programming problems actually are mixed 0-1 problems, this can be seen as an explanation for their success.

5.4. The cMIR Cut Generator

In this section we describe the cut generation step of the cMIR cut generator implemented for this thesis. The aggregation and bound substitution steps are described in section 5.2 because they are also used for other cut generators. In algorithm 5 we show the program flow of our cMIR cut generator.

The cMIR cut generation procedure is called within the framework for each reversed mixed integer knapsack set generated from the usable rows, the aggregated rows, and the reverse of these rows. At the beginning it checks whether the reversed mixed integer knapsack set contains integer variables which are fractional in the current LP solution. If this is not the case, it does not try to generate a cut. This speeds up the cut generation and does not influence the quality of the generated cuts too much.

Note that the reversed mixed integer knapsack sets passed on to this separation routine might have a continuous variable s that is actually equivalent to a single slack variable for the original input row or even equal to 0. Hence the cut generator might generate pure integer cuts that are actually strengthened Chvátal-Gomory inequalities (see [62]). One result of doing this is that the cMIR cut generator now can compute cuts for the lot-sizing problem with stock upper bounds as shown in example 5.5. Note that it is also possible to pass these knapsack constraints on to a cover cut generator if all variables are binary. We do not further investigate this as it goes beyond the scope of this thesis.

Example 5.5. *Assume an instance of the constant-capacity lot-sizing problem with stock upper bounds (see [85]). In this instance we find the structure*

$$\begin{aligned} s_1 + x_1 - s_2 &= 2 \\ s_2 + x_2 - s_3 &= 4 \\ s_3 + x_3 - s_4 &= 5 \\ x_j &\leq 10 y_j \text{ for } j = 1,2,3 \\ s_1 &\leq 5 \\ s \in \mathbb{R}^4_+ \qquad x &\in \mathbb{R}^3_+ \qquad y \in \{0,1\}^3 \end{aligned}$$

Algorithm 5 The cMIR Cut Generator

procedure CMIR_CUT_GENERATOR(mik)
    if CONTAINS_INTEGER_VARS_LESS_THAN_ZERO(mik) **then return** $\emptyset$
    if NO_FRACTIONAL_INTEGER(mik) **then return** $\emptyset$
    $bestCut \leftarrow \emptyset$
    $C \leftarrow \{j \in I : y_j^* \geq \frac{u_j}{2}\}$
    for $j \in I$ **do**
        $\delta \leftarrow |g_j|$
        $cut \leftarrow$ COMPUTE_CMIR_CUT(mik,C,δ)
        if cut **better than** $bestCut$ **then**
            $bestCut \leftarrow cut$
            $\bar{\delta} \leftarrow \delta$
        end if
    end for
    if $bestCut = \emptyset$ **then return**
    $\delta^* \leftarrow \bar{\delta}$
    for $k = 1, 2, 3$ **do**
        $\delta \leftarrow \frac{\delta^*}{2^k}$
        $cut \leftarrow$ COMPUTE_CMIR_CUT(mik,C,δ)
        if cut **better than** $bestCut$ **then**
            $bestCut \leftarrow cut$
            $\bar{\delta} \leftarrow \delta$
        end if
    end for
    $T \leftarrow \{j \in I, 0 < y_j^* < \frac{u_j}{2}\}$
    sort $t \in T$ **by** $|y_t^* - \frac{u_t}{2}|$
    for $t \in T$ **do**
        $C \leftarrow C \cup t$
        $cut \leftarrow$ COMPUTE_CMIR_CUT(mik,C,$\bar{\delta}$)
        if cut **better than** $bestCut$ **then**
            $bestCut \leftarrow cut$
        else
            $C \leftarrow C \setminus t$
        end if
    end for
    $finalCut \leftarrow$ CLEAN_CUT($bestCut$)
    return $finalCut$
end procedure

From this structure we can compute the valid inequality

$$y_1 + y_2 + y_3 \geq 1$$

which is an important facet of the convex hull (see [84] and [85], p. 353). We can generate this valid inequality using an MIR inequality by first aggregating the three rows of the path and substituting the variable upper bounds for x_1, x_2, *and* x_3. *The aggregated row then looks like this:*

$$s_1 + 10y_1 + 10y_2 + 10y_3 \geq 11.$$

Now we can also substitute the simple bound of s_1. *The result is the reversed mixed integer knapsack*

$$10y_1 + 10y_2 + 10y_3 + s \geq 6$$

where we assume that $s = 0$. *The MIR inequality with* $\delta = 10$ *is the cut we are looking for.*

The basic idea of the cMIR cut generation is to do a search of the family of reversed cMIR inequalities. A reversed cMIR inequality is defined by a partition of $I = (T, C)$ and a value $\delta \in \mathbb{R}_{>0}$. Note that we use reversed cMIR inequalities instead of the normal cMIR inequalities because our bound substitution generates reversed mixed integer knapsack sets. Example 5.6 shows that by generating cuts from reversed rows we end up with exactly the same cuts as with normal cMIR inequalities. Also note that, in contrast to other implementations, our definition of the reversed cMIR inequalities includes rescaling the cut to improve its accuracy.

Example 5.6. *In this example we show how the cMIR cut from example 4.4 can be generated using the reversed cMIR inequality. The first step is to relax the single node flow set to a reversed mixed integer knapsack. To do this we first introduce a slack variable* $t_r = 17 - x_1 - x_2 - x_3 + x_4 + x_5$ *and substitute all variable bounds. Now the slack variables with negative coefficients are relaxed and the result is the reverse mixed integer knapsack set*

$$24y_1 + 20y_2 + 15y_3 - 18y_4 - 16y_5 + \underbrace{t_4 + t_5 + t_r}_{s} \geq 17.$$

Applying the reversed cMIR inequality with $C = \{1, 2, 5\}$ *and* $\delta = 24$ *yields the cut*

$$-13(1 - y_1) - 9(1 - y_2) + 13y_3 - 7y_4 + 13(1 - y_5) + t_4 + t_5 + t_r \geq 0$$

which after substituting t_4, t_5 *and* t_r *results in the same cut as in example 4.4.*

The first step of the algorithm is to decide on an initial set for C, i.e. the set of complemented variables. Our cut generator follows the rule of Marchand and Wolsey [70], i.e. for the reversed mixed integer knapsack

$$\sum_{j \in I} g_j y_j + s \geq b$$

it uses the initial set

$$C = \{j \in I : y_j^* \geq \frac{u_j}{2}\}.$$

Also as described by Marchand and Wolsey it stores a set of candidates for complementing, the set U, where

$$U = \{j \in N : 0 < y_j^* < \frac{u_j}{2}\}.$$

Then the algorithm tries certain values for δ. Marchand and Wolsey suggest to use the coefficients of the fractional integer variables in the mixed knapsack as candidates. In our implementation we use the absolute values of all coefficients of the integer variables, i.e. $|g_j|$ for all $j \in N$. To speed this up we check whether the new δ to try is the same as the last δ tried. This is especially efficient if all coefficients are the same which can be observed frequently in practical problem instances. It gets more efficient if the variables $j \in N$ are sorted but this is not in general the case in our cut generator. When trying a certain value for δ we check whether δ and the divided right hand side $\frac{b}{\delta}$ are larger than the tolerance parameter `xtolin` (by default 1×10^{-5}). This avoids numerical problems and thus improves the accuracy of the generated cuts. The computing of the cut coefficients is implemented in a very efficient way because it might be called many times in each round of the cut generation. Two indexed data structures are used to store the cut that is currently generated and the best cut generated so far. If the current cut is better than the best, instead of copying the cut, the pointers to the data structure are switched. Especially in the case that many better cuts are found this improves the efficiency of the cut generator very much. Furthermore, when a cut is generated only the coefficients of the integer variables are computed and the violation is computed using s^*, which is the sum of the LP solutions of the variables in s. Only for the final cut the coefficients for the variables in s are explicitly computed.

An important step in the cMIR cut generation is to decide whether a cut is better than a previously generated one. Comparing the violation of two cuts does not necessarily lead to the best cut. Hence we compare the cuts using their *normalized violation* (or

Euclidean distance) which in our case for a cut $\beta y + s \geq \gamma$ is defined as

$$v_n = \frac{s^* + \beta y_j^* - \gamma}{||\beta||}.$$

Note that the coefficients of the continuous variables in s are not considered when normalizing the violation as they are the same for all cuts compared. This quality measure is for example also used in [8]. Further information about comparing the quality of cuts and more references can be found in [96].

To avoid spending too much time on a row were it is not likely that a violated cut is found the cut generator does not proceed to look for a cut if in this first loop no violated cut was found. If a violated cut is found, it tries to strengthen this cut as suggested by Marchand and Wolsey by dividing δ by 2, 4, and 8. Finally, it tries to strengthen the cut by complementing the variables in U one by one. This methods keeps a variable in U complemented if this results in a better cut. The best cut finally is cleaned as described in section 5.2 and added to the cut pool.

5.5. The Flow Path Cut Generator

The flow path cut generator implemented for this thesis follows the description of the initial implementation by van Roy and Wolsey in [93]. The major difference is the path-finding method described in section 5.2. Algorithm 6 shows the program flow. Note that we call the flow path cut generation procedure only for paths with a length of at least 2 because the cMIR and the flow cover cuts typically dominate the flow path cuts for single rows. The cMIR and flow cover cut generators are called for rows and reversed rows. The flow path cut generator is called for each path and thus it is called only once for every two times the other two cut generators are called. In other words, it is not called for reversed rows or reversed paths.

The first loop runs backwards through the rows of the path. First it checks for each row that it can be reformulated. Then it removes the sets of variables that connect parts of the path, i.e. the sets R_k^+ and R_k^- as defined in section 4.3, and reformulates the row. Identifying the set R_k^+ and R_k^- constitutes a difficult task because it is necessary to check that the coefficients of the rows really negate each other. Hence accuracy safeguards similar to those used in the aggregation heuristic are needed. To search for variables in the path we again use the column-wise representation of the constraint matrix which speeds up this process very much. The sets R_k^+ and R_k^- are stored to be used again in

Algorithm 6 The Flow Path Cut Generator

```
procedure FLOW_PATH_CUT_GENERATOR(path)
    vio ← 0
    for k = |path| ... 1 do
        if CONTAINS_VARS_LESS_THAN_ZERO(path_k) then return ∅
        IDENTIFY_R_SETS(path_k, R_k^+, R_k^-)
        row ← call REMOVE_R_SETS_AND_ADD_SLACK(path_k, R_k^+, R_k^-)
        SET_REFORMULATION_STATUS(row, refSta)
        C_k^+ ← {j ∈ N_k^+ : x_j^* − (Σ_{i=k}^{K} b^+) y_j^* > 0}
        Q_k^- ← ∅
        ȳ_k ← Σ_{j∈C_k^+} y_j^*
        vio ← UPDATE_VIOLATION(vio, row, refSta, cut, C^+, Q^-)
    end for
    if vio ≤ 0 then return ∅
    cut ← ∅
    for k = |path| ... 1 do
        row ← REMOVE_R_SETS_AND_ADD_SLACK(path_k, R_k^+, R_k^-)
        SET_REFORMULATION_STATUS(row, refSta)
        C_k^+ ← {j ∈ N_k^+ : x_j^* − (Σ_{i=k}^{K} b^+) y_j^* > 0}
        Q_k^- ← {j ∈ N_k^- : x_j^* > (Σ_{i=1}^{k} ȳ_i) u_j}
        cut ← UPDATE_CUT(cut, row, refSta, C^+, Q^-)
    end for
    finalCut ← CLEAN_CUT(cut)
    return finalCut
end procedure
```

the second loop. The reformulation as a binary single node flow set is done in the same way as for the flow cover cut generator.

The actual task of the first loop is to check whether a simple network inequality based on the input path is violated and to compute the values of $\bar{y}_k = \sum_{j \in C_k^+} y_j^*$ for $k = 1 \dots K$. These values are needed to decide in the second loop which variables to use in the sets Q_k^-. To compute the simple network inequality, we need to decide on the sets C_k^+. We put a variable j into C_k^+ if it improves the violation of the cut, i.e. if $x_j^* - \sum_{i=k}^{K} b^+ y_j^* > 0$. As in the case of the flow cover cut generator we use $>$ instead of $\geq$ although this does not mean a difference in the violation of the cut.

In the second loop the cut generator produces an extended network inequality. The sets from the first loop are used in the same way as before but now we use some of the variables $j \in N_k^-$ to strengthen the cut. Whether a variable can be used to strengthen the cut can be decided directly by checking whether $x_j^* > (\sum_{i=1}^{k} \bar{y}_i) u_j$. We assume that this is what van Roy and Wolsey meant on page 52 of [93]. In section 6.5 we investigate how important it is to use the extended instead of the simple network inequality.

5.6. The Path Mixing Cut Generators

5.6.1. Path Mixing Inequalities

In this section we propose two new separation algorithms and discuss their implementation. The idea of the algorithms is to be more general substitutes for the flow path cut separation algorithm. Therefore we need valid inequalities for more general sets than fixed charge paths. We call these more general sets *mixed integer paths.*

Definition 5.2. *A mixed integer path is described by a set of K constraints of the form*

$$\begin{aligned} \sum_{j=1}^{n} a_{j1} x_j + \sum_{j=1}^{p} g_{j1} y_j &= b_1 \\ \sum_{j=1}^{n} a_{j2} x_j + \sum_{j=1}^{p} g_{j2} y_j &= b_2 \\ &\vdots \\ \sum_{j=1}^{n} a_{jK} x_j + \sum_{j=1}^{p} g_{jK} y_j &= b_K \end{aligned}$$

with $a_{jk} + a_{jk+1} = 0$ *for at least one* $j \in \{1 \ldots n\}$ *and all* $k = 1 \ldots K-1$. *Note that* $x \in \mathbb{R}^n_+, a \in \mathbb{R}^n \times \mathbb{R}^K, g \in \mathbb{R}^p \times \mathbb{R}^K, y \in \mathbb{Z}^p_+, b \in \mathbb{R}^K$ *and that* a_{jk} *and* g_{jk} *might be* 0 *for some* $(j,k), j = 1 \ldots n, k = 1 \ldots K$.

To find valid inequalities for mixed integer paths we follow the same approach used for finding the MIR inequalities. We relax the structure to a very simple set and apply valid inequalities for this simple set. The simple set we relax the mixed integer paths to is the mixing set first studied by Günlük and Pochet in [52] (see section 2.6) and we call the resulting inequalities *path mixing inequalities.*

Proposition 5.1. *Let* $\delta \in \mathbb{R}^K_{>0}$ *and* $T \subseteq \{1 \ldots K\}$, $|T| = t$. *Furthermore*

$$f_k = \frac{\sum_{i=1}^k b_i}{\delta_k} - \left\lfloor \frac{\sum_{i=1}^k b_i}{\delta_k} \right\rfloor \text{ for } k = 1 \ldots K,$$

$$h_{jk} = \frac{\sum_{i=1}^k g_{ji}}{\delta_k} - \left\lfloor \frac{\sum_{i=1}^k g_{ji}}{\delta_k} \right\rfloor \text{ for } j = 1 \ldots p,\ k = 1 \ldots K,$$

and suppose that $i_1, \ldots, i_t$ *is an ordering of* T *such that* $0 = f_{i_0} \le f_{i_1} \le \cdots \le f_{i_t}$. *Then the path mixing inequalities*

$$s \ge \sum_{\tau=1}^{t} \left(f_{i_\tau} - f_{i_{\tau-1}}\right) \left(\lceil \bar{b}_{i_\tau} \rceil - \sum_{j \in I_1} \left\lceil \frac{\sum_{i=1}^{i_\tau} g_{ji}}{\delta_k} \right\rceil y_j - \sum_{j \in I_2} \left\lfloor \frac{\sum_{i=1}^{i_\tau} g_{ji}}{\delta_k} \right\rfloor y_j \right)$$

and

$$s \ge \sum_{\tau=1}^{t} \left(f_{i_\tau} - f_{i_{\tau-1}}\right) \left(\lceil \bar{b}_{i_\tau} \rceil - \sum_{j \in I_1} \left\lceil \frac{\sum_{i=1}^{i_\tau} g_{ji}}{\delta_{i_\tau}} \right\rceil y_j - \sum_{j \in I_2} \left\lfloor \frac{\sum_{i=1}^{i_\tau} g_{ji}}{\delta_{i_\tau}} \right\rfloor y_j \right)$$
$$+ (1 - f_{i_t}) \left(\lfloor \bar{b}_{i_1} \rfloor - \sum_{j \in I_1} \left\lceil \frac{\sum_{i=1}^{i_1} g_{ji}}{\delta_{i_1}} \right\rceil y_j - \sum_{j \in I_2} \left\lfloor \frac{\sum_{i=1}^{i_1} g_{ji}}{\delta_{i_1}} \right\rfloor y_j \right)$$

where $I = \{1, \ldots, p\}$,$I = (I_1, I_2)$,

$$s = \sum_{j=1}^{n} \left(\max_{k \in T} \left\{ \frac{\sum_{i=1}^k a_{ji}}{\delta_k} \right\} \right)^+ x_j + \sum_{j \in I_2} \left(\max_{k \in T} \{h_{jk}\} \right)^+ y_j$$

and

$$\bar{b}_k = \frac{\sum_{i=1}^k b_i}{\delta_k} \text{ for } k = 1 \ldots K$$

are valid inequalities for mixed integer paths (see definition 5.2).

Proof. We sum the first k rows of the mixed integer path to get an aggregated path of the form

$$\sum_{i=1}^{k}\sum_{j=1}^{n} a_{ji}x_j + \sum_{i=1}^{k}\sum_{j=1}^{p} g_{ji}y_j = \sum_{i=1}^{k} b_i \text{ for } k \in \{i_1,\ldots,i_t\}.$$

Now we divide each row of the aggregated path by a value δ_k and split the integer variables $y_j, j \in I$ into two disjunct sets I_1 and I_2. For I_1 we relax the coefficients by rounding up and for I_2 we split the coefficient into the integer part and the fractional part h_{jk}. The result is the following aggregated path

$$\sum_{j=1}^{n} \frac{\sum_{i=1}^{k} a_{ji}}{\delta_k} x_j + \sum_{j\in I_2} h_{jk}y_j + \sum_{j\in I_1}\left\lceil \frac{\sum_{i=1}^{k} g_{ji}}{\delta_k}\right\rceil y_j + \sum_{j\in I_2}\left\lfloor \frac{\sum_{i=1}^{k} g_{ji}}{\delta_k}\right\rfloor y_j \geq \frac{\sum_{i=1}^{k} b_i}{\delta_k}$$
$$\text{for } k \in \{i_1,\ldots,i_t\}$$

which can be rewritten as a mixing set

$$s + \underbrace{\sum_{j\in I_1}\left\lceil \frac{\sum_{i=1}^{k} g_{ji}}{\delta_k}\right\rceil y_j + \sum_{j\in I_2}\left\lfloor \frac{\sum_{i=1}^{k} g_{ji}}{\delta_k}\right\rfloor y_j}_{y'_k \in \mathbb{Z}^t} \geq \frac{\sum_{i=1}^{k} b_i}{\delta_k} \text{ for } k \in \{i_1,\ldots,i_t\}.$$

This is a mixing set because

$$s \geq \sum_{j=1}^{n} \frac{\sum_{i=1}^{k} a_{ji}}{\delta_k} x_j + \sum_{j\in I_2} h_{jk}y_j \geq 0 \text{ for all } k \in \{i_1,\ldots,i_t\}.$$

Applying the mixing inequalities from section 2.6 yields the path mixing inequalities. □

These path mixing inequalities have several advantages. One is that they generalize the MIR inequality in the sense that a path mixing cut from a single row of an aggregated path is the same as an MIR cut from this row. If we assume that in the mixed integer path some variables are complemented they even generalize the cMIR inequalities. Another advantage is that, as shown in [52], several classes of problem specific cuts can be generated as path mixing cuts. This includes the *(k,l,S,I) inequalities* (see [83]) for the constant capacity lot-sizing problem, which is shown in example 5.7, as well as several other lot-sizing and network design based problem classes.

Example 5.7. *Consider an instance of the constant capacity lot-sizing problem (called LS-CC in [85])*

$$\begin{aligned}
&\min \sum_{t=1}^{n} p_t x_t + \sum_{t=0}^{n} h_t s_t + \sum_{t=1}^{n} q_t y_t \\
&s_{t-1} + x_t = d_t + s_t && \text{for } 1 \leq t \leq n \\
&x_t \leq C y_t && \text{for } 1 \leq t \leq n \\
&s \in \mathbb{R}_+^{n+1}, x \in \mathbb{R}_+^n, y \in \{0,1\}^n
\end{aligned}$$

with $n = 4$*,* $(p, h, q) = \{2, 3, 2, 1, 100, 1, 1, 1, 1, 80, 80, 80, 80\}$*,* $d = \{2, 6, 4, 5\}$ *and* $C = 10$*. By substituting the variable upper bounds for all production variables* x_j *we can generate the mixed integer path*

$$\begin{aligned}
s_0 + 10y_1 - s_1 - r_1 &= 2 \\
s_1 + 10y_2 - s_2 - r_2 &= 6 \\
s_2 + 10y_3 - s_3 - r_3 &= 4 \\
s_3 + 10y_4 - s_4 - r_4 &= 5
\end{aligned}$$

where the variables r_k*,* $k = 1 \ldots n$*, are slack variables. Applying the first path mixing inequality with* $I_2 = \emptyset$*,* $T = \{1, 4\}$*, and* $\delta_1 = \delta_4 = 10$ *yields the cut*

$$s_0 \geq 12 - 7y_1 - 5y_2 - 5y_3 - 5y_4$$

which also is a (k,l,S,I) inequality and a facet of the convex hull for this problem.

In the context of this thesis especially important is the fact that flow path cuts can also be generated using path mixing inequalities. We show this by first relaxing the rows of a fixed charge path to an appropriate mixed integer path. The first step is to relax b_k to b_k^+. Then we introduce slack variables r_k to get equality rows and substitute the variable bounds for all $j \in C_k^+ \subset N^+, k = 1 \ldots K$. The result is the mixed integer path

$$x_{k-1} - x_k + \sum_{j \in N_k^+ \setminus C_k^+} x_j - \sum_{j \in N_k^-} x_j + r_k + \sum_{j \in C_k^+} u_j y_j - \sum_{j \in C_k^+} t_j = b_k^+ \text{ for } k = 1, \ldots, K$$

where $x_0 = x_K = 0$. Now we apply the first path mixing inequality with

$$\bar{\delta} = \max \left\{ \sum_{i=1}^{K} b_i^+, \max_{j \in \bigcup_{i=1}^{K} C_i^+} u_j \right\},$$

$\delta_k = \bar{\delta}$ for all $k = 1 \dots K$, $I_1 = \bigcup_{k=1}^{K} C_k^+$, and $T = \{1 \dots K\}$. The indices in T are already ordered by increasing f_k because the right hand sides are non-negative. The resulting inequality is

$$\sum_{k=1}^{K} \sum_{j \in N_k^+ \setminus C_k^+} \frac{x_j}{\bar{\delta}} + \sum_{k=1}^{K} \frac{r_k}{\bar{\delta}} \geq \sum_{k=1}^{K} (f_k - f_{k-1}) \left(\left\lceil \frac{\sum_{i=1}^{k} b_i^+}{\bar{\delta}} \right\rceil - \sum_{i=1}^{k} \sum_{j \in C_i^+} \left\lceil \frac{u_j}{\bar{\delta}} \right\rceil y_j \right).$$

Now because $\bar{\delta}$ is large enough and $f_k - f_{k-1} = \frac{\sum_{i=1}^{k} b_i^+}{\bar{\delta}} - \frac{\sum_{i=1}^{k-1} b_i^+}{\bar{\delta}} = \frac{b_k^+}{\bar{\delta}}$ we get

$$\sum_{k=1}^{K} \sum_{j \in N_k^+ \setminus C_k^+} \frac{x_j}{\bar{\delta}} + \sum_{k=1}^{K} \frac{r_k}{\bar{\delta}} \geq \sum_{k=1}^{K} \frac{b_k^+}{\bar{\delta}} \left(1 - \sum_{i=1}^{k} \sum_{j \in C_i^+} y_j \right).$$

After multiplying by $-\bar{\delta}$ and rewriting this gets

$$-\sum_{k=1}^{K} \sum_{j \in N_k^+ \setminus C_k^+} x_j - \sum_{k=1}^{K} r_k \leq -\sum_{k=1}^{K} b_k^+ + \sum_{k=1}^{K} b_k^+ \sum_{i=1}^{k} \sum_{j \in C_i^+} y_j.$$

Substituting the slack variables and rewriting again yields the simple network inequality. To get the extended network inequality we replace the variables $j \in Q_k^- \subseteq N^-, k = 1 \dots K$ by their upper bounds u_j. The result is that

$$f_k = \frac{\sum_{i=1}^{k} (b_i^+ + \sum_{j \in Q_i^-} u_j)}{\bar{\delta}} \text{ for } k = 1 \dots K.$$

Following the same steps as above results in the extended network inequality.

Another very interesting relation is the one between cMIR and path mixing inequalities. In [37] Dash and Günlük give a proof that the MIR rank of a mixing inequality from a mixing set with K rows is at most K. Dey shows in [41] that a lower bound on the split rank of the first mixing inequality is $\lceil \log_2(k+1) \rceil$. These results suggest that it should be possible to generate path mixing inequalities from short paths with a cMIR cut separation algorithm that generates cuts out of cuts. This can actually be observed in the implementation done for this thesis, example 5.8 illustrates this. Note that Marchand showed in chapter 4 of his thesis [69] that inequalities for the capacitated lot-sizing problem can also be generated using lifting.

Example 5.8. *In the following path*

$$x_1 - 2y_1 - s_1 \leq 0$$
$$s_1 + x_2 - s_2 = 6$$
$$s_2 + x_3 - s_3 = 4$$
$$s_3 + x_4 - s_4 = 5,$$

the first row is the MIR cut generated from the flow balance constraint $s_0 + x_1 - s_1 = 2$ *of the LS-CC problem in example 5.7. By aggregating this path and after substituting all variable bounds we get the mixed integer knapsack*

$$-t_1 - t_2 - t_3 - t_4 - s_4 + 8y_1 + 10y_2 + 10y_3 + 10y_4 \leq 15,$$

where t_j *is the slack variable of the variable upper bound constraint of* j*. Applying the cMIR inequality using* $\delta = 10$ *and* $C = \emptyset$ *this results in the cut*

$$x_1 + x_2 + x_3 + x_4 - s_4 - 7y_1 - 5y_2 - 5y_3 - 5y_4 \leq 5$$

which is equivalent to the facet found in example 5.7. It is equivalent because we can substitute $x_1 + x_2 + x_3 + x_4 - s_4 = 17 - s_0$ *and multiply by* -1 *to get the same cut. We now use a slightly more difficult example. In this mixed integer path*

$$x_4 - 5y_4 - s_4 \leq 0$$
$$-s_3 - x_4 + s_4 = -5$$
$$-s_2 - x_3 + s_3 = -4$$
$$-s_1 - x_2 + s_2 = -6$$
$$-s_0 - x_1 + s_1 = -5,$$

the first row is again a simple MIR cut. The other rows have been multiplied by -1*. After adding a slack variable* p *for the first row, aggregating, multiplying by* -1 *and substituting all variable upper bounds we get the following reversed mixed integer knapsack set*

$$-t_1 - t_2 - t_3 - p + 10y_1 + 10y_2 + 10y_3 + 5y_4 \leq 17.$$

Applying the cMIR inequality using $\delta = 10$*,* $C = \emptyset$*, and substituting* p *yields*

$$x_1 + x_2 + x_3 + x_4 - s_4 - 7y_1 - 7y_2 - 7y_3 - 5y_4 \leq 3.$$

An equivalent cut can also be found by applying the path mixing inequality to the mixed integer path

$$s_0 + 10y_1 - s_1 = 2$$
$$s_1 + 10y_2 - s_2 = 6$$
$$s_2 + 10y_3 - s_3 = 4$$
$$s_3 + 10y_4 - s_4 = 5$$

with $\delta_k = 10, k = 1 \dots K$, $I_2 = \emptyset$ *and* $T = \{3, 4\}$ *and multiplying the result by* δ.

5.6.2. Two Separation Algorithms

The two path mixing cut separation algorithms we want to present here are both based on the *path mixing approach*. It is a generalization and formalization of the approach used by Günlük and Pochet in [52] to demonstrate that certain strong valid inequalities can be generated using mixing inequalities. Starting with a mixed integer path defined in the last section we go through the following steps:

1. Aggregate the first k rows of the path for $k = 1 \dots K$,
2. Use bound substitution to reformulate each of these aggregated rows into a reversed mixed integer knapsack,
3. choose a set $T \subseteq \{1 \dots K\}$,
4. choose a value $\delta_k \in \mathbb{R}_{>0}$ for each aggregated row k,
5. sort the elements in T by increasing f_k,
6. start with the path mixing cut generated from the first element in T,
7. extend the cut by running through the remaining elements in T.

The basic idea of our first path mixing separation algorithm is to imitate the flow path cut separation algorithm. We call it the *uncapacitated path mixing cut (uPMC) separation algorithm* because it generates the (l, S)-inequalities (see [19]) that are facet defining for the uncapacitated lot-sizing problem. The first step is to aggregate the first k rows for $k = 1 \dots K$ to get an aggregated path. These rows then are reformulated into reversed mixed integer knapsack sets. We choose T as the first t reversed mixed integer knapsacks for which $\sum_{i=1}^{k} b_i$, $k = 1 \dots t$ is increasing. If all b_k in the original rows are greater than or equal to 0 then $t = K$. For all δ_k we choose a value M that is arbitrarily large.

Now because $f_k = \frac{b_k}{\delta_k}$, T is already sorted and we can add the reversed mixed integer knapsacks to the cut one by one. In this one by one extension of the cut it can be decided whether to place an integer variable in the I_2 set, specified in the definition of the path mixing inequality, by checking whether $g_j < (f_k - f_{k-1})$. Each time we added a reversed mixed integer knapsack set we check whether the resulting cut is violated. If it is, this is the cut the algorithm returns. If not, the algorithm continues up to a parameter for maximum path length. This procedure in many cases generates the same cuts as a flow path cut separation algorithm. Its performance partially depends on the bound substitution heuristic used. See example 5.9 for an demonstration of this algorithm.

Example 5.9. *Assume the following mixed integer path*

$$\begin{aligned} 2x_1 + 10y_1 - x_3 &= 4 \\ x_3 - x_1 + x_2 + 2y_2 - x_4 &= 3 \\ x_4 + x_2 + 10y_3 - x_5 &= 5 \\ x \in \mathbb{R}^5_+, &y \in \mathbb{Z}^3_+. \end{aligned}$$

By aggregation and bound substitution (we assume that all continuous variables have been replaced by a static lower bound of 0*) we get the three reversed mixed integer knapsack sets*

$$\begin{aligned} 10y_1 + \underbrace{2x_1}_{s_1} &\geq 4 \\ 10y_1 + 2y_2 + \underbrace{x_1 + x_2}_{s_2} &\geq 7 \\ 10y_1 + 2y_2 + 10y_3 + \underbrace{x_1 + 2x_2}_{s_3} &\geq 12. \end{aligned}$$

We now add these reversed mixed integer knapsack sets one by one to our cut. The cut from the first reversed mixed integer knapsack set is

$$2x_1 \geq 4(1 - y_1).$$

Now we add the second reversed mixed integer knapsack set and get

$$2x_1 + x_2 \geq 4(1 - y_1) + 3(1 - y_1) - 2y_2.$$

Note that we have put y_2 into the set N_2 of the path mixing inequality and that we added x_2 to the left hand side. For x_1 nothing changed because the coefficient in the cut is larger *than the one in the reversed mixed integer knapsack. After checking that this cut is not violated, we add the third reversed mixed integer knapsack set and get*

$$2x_1 + 2x_2 + 2y_2 \geq 4(1 - y_1) + 3(1 - y_1) + 5(1 - y_1 - y_3).$$

We again have put y_2 into the set N_2 and because it had already been in the cut it was not added again. The coefficient of x_2 in the cut was increased to 2. Note that the continuous variables do not correspond to $\max_{k=\{1,2,3\}} s_k$. The maximum of the coefficient has to be taken for each variable individually. Using $s_1 + s_2 + s_3$ is also possible but would result in a weaker cut.

The second path mixing cut separation algorithm presented here tries to generate cuts that go beyond flow path cuts. We call it the *capacitated path mixing cut (cPMC) separation algorithm* because it generates the (k, l, S, I)-inequalities (see [83]) that are facet defining for the constant capacity lot-sizing problem. The idea is to use the separation algorithm for the simple mixing set described in section 2.6 in this more complex situation. Again we aggregate the rows of the path and reformulate them as reversed mixed integer knapsack sets. Then the algorithm tries to find cuts using several different sets for T. To decide on these sets, the mixed integer knapsack sets are sorted by a decreasing value of β_k, where

$$\beta_k = \left\lceil \frac{b_k}{\delta_k} \right\rceil - \sum_{j \in I} \left\lceil \frac{g_{jk}}{\delta_k} \right\rceil y_j^*.$$

For δ_k we use the maximal g_j in the reversed mixed integer knapsack set k. This definition of β_k corresponds to the one for β in the description of the separation algorithm for the simple mixing set in [85] (see section 2.6). Once the rows are sorted we use the δ_k of the first row, i.e. the row with the largest β_k, as δ_k for all k. The algorithm then tries to generate a cut from the r reversed mixed knapsack sets where β_k is largest. To do so the first r rows are sorted by increasing f_k and then the cut is generated. This is done for $r = 2 \ldots K$ until a violated cut is found. We demonstrate this algorithm in example 5.10.

Example 5.10. *We use the same set of reversed mixed integer knapsack sets as in the previous example*

$$10y_1 + \underbrace{2x_1}_{s_1} \geq 4$$
$$10y_1 + 2y_2 + \underbrace{x_1 + x_2}_{s_2} \geq 7$$
$$10y_1 + 2y_2 + 10y_3 + \underbrace{x_1 + 2x_2}_{s_3} \geq 12.$$

The current LP solution we try to cut off is the point $(x^*, y^*) = \left(2, 3, 0, 0, 0, 0, 1, \frac{1}{5}\right)$. *In this case* $\delta_k = 10$ *for all* k *and the values of* β_k *are*

$$\beta_1 = 1 - 0 = 1$$
$$\beta_2 = 1 - 0 - 1 = 0$$
$$\beta_3 = 2 - 0 - 1 - \frac{1}{5} = \frac{4}{5}.$$

We now sort by β_k *and generate the cut for the two reversed mixed integer knapsack sets with largest* β_k, *i.e.* $T = \{1, 3\}$. T *has to be sorted by* f_k, *so we get* $T = \{3, 1\}$ *and thus can compute the cut*

$$2x_1 + 2x_2 + 2y_2 \geq 2(2 - y_1 - y_3) + 2(1 - y_1)$$

where we used y_2 *in* I_2. *As this cut is not violated, the next step is to try the three reversed mixed integer knapsack sets with largest* β_k, *i.e.* $T = \{3, 1, 2\}$. *This time* T *is already sorted so we get*

$$2x_1 + 2x_2 + 2y_2 \geq 2(2 - y_1 - y_3) + 2(1 - y_1) + 3(1 - y_1)$$

for which the violation is larger than for the last cut, but still not larger than 0. *The algorithm stops without having found a violated cut.*

5.6.3. Implementation of the Path Mixing Cut Generators

For the implementation of the aforementioned path mixing cut separation algorithms we make use of the fact that many parts of the cMIR cut generator can be reused.

Aggregation and bound substitution are done in the framework for both the cMIR and the path mixing cut generators. Note that only reversed mixed integer knapsack sets generated from the (aggregated) rows, but not from reversed rows, are used. So basically the path mixing cut generators are called once for every two times the cMIR and flow cover cut generators are called.

In algorithm 7 we show the program flow of the uPMC generator. When the uPMC generator is called the first time it is initialized with $cut = \emptyset$ and $f_{last} = 0$. In the following iterations of the framework's aggregation loop the uPMC separation routine is called with the current reversed mixed integer knapsack set until it generates a violated cut or the maximum path length is reached. As a result of this way of implementing the uPMC separation algorithm, an existing cMIR cut generator can very easily be extended to also generate path mixing cuts and separating these additional cuts does not increase the runtime very much.

Algorithm 7 The uPMC Cut Generator

1: **procedure** uPMC_Generator(mik,cut,f_{last})
2:     **if** $b < f_{last}$ **then**
3:         $cut \leftarrow \emptyset$
4:         $f_{last} \leftarrow 0$
5:         **return** $\emptyset$
6:     **end if**
7:     $f \leftarrow b$
8:     $newCut \leftarrow$ Append_MIK_To_Cut(mik,cut,$f - f_{last}$,M)
9:     **if** vio($newCut$) > 0 **then**
10:         $finalCut \leftarrow$ Clean_Cut($newCut$)
11:         **return** $finalCut$
12:     **else**
13:         $cut \leftarrow newCut$
14:         $f_{last} \leftarrow f$
15:     **end if**
16: **end procedure**

The core of the uPMC cut generator is the procedure in line 8. It appends the reversed mixed integer knapsack set to the current cut generated in previous iterations. Based on the coefficients in the current cut it checks whether the violation increases more if a variable is put into set I_1 or I_2. It also checks whether a variable has been used in I_2 in a previous iteration and thus potentially does not change the value of the left hand side. Finally, the procedure in line 8 rescales the cut internally to be in the same scale as the input row (see section 5.2.3). The final cut is cleaned as described in section 5.2.3 and additionally the last aggregated row is subtracted from the resulting cut. This is done to get exactly the same cut as a flow path cut generator would and not an equivalent one (see example 5.8).

The cPMC generator is both implementationally more demanding and slower in its execution than the uPMC generator. On the other hand it potentially generates cuts not obtainable by using a flow path cut or uPMC generator. Its implementation is outlined in algorithm 8. First, the reversed mixed integer knapsack sets are stored in a linked list data structure. This list then is sorted by β_k. In the loop from line 6 to line 21 the first r reversed mixed integer knapsack sets are sorted by f_t. This step can be implemented vary fast by inserting the t-th reversed mixed integer knapsack set into the already sorted set 1 to $t-1$. Then the cut is generated using a similar procedure as in the implementation of the uPMC generator. Note that here the rescaling of the cut happens after a violated cut has been found.

Algorithm 8 The cPMC Cut Generator

1: **procedure** CPMC_CUT_GENERATOR(mik, β, δ, $mikList$, k)
2: $\quad mikList_k \leftarrow mik$
3: $\quad \delta_k \leftarrow \max_{j\in I} |g_j|$
4: $\quad \beta_k \leftarrow \left\lceil \frac{b_k}{\delta_k} \right\rceil - \sum_{j\in I} \left\lceil \frac{g_{jk}}{\delta_k} \right\rceil y_j^*$
5: $\quad$ **sort** $r \in mikList$ **by** β_r
6: $\quad$ **for** r = 2 ... k **do**
7: $\qquad f_{last} \leftarrow 0$
8: $\qquad cut \leftarrow \emptyset$
9: $\qquad \bar{\delta} = \delta_1$
10: $\qquad$ **sort** $t \in \{mikList_1, mikList_2, \dots mikList_r\}$ **by** $f_t = \frac{b_t}{\bar{\delta}} - \left\lfloor \frac{b_t}{\bar{\delta}} \right\rfloor$
11: $\qquad$ **for** t = 1 ... r **do**
12: $\qquad\quad newCut \leftarrow$ APPEND_MIK_TO_CUT($mikList_k$,cut,$f - f_{last}$)
13: $\qquad\quad cut \leftarrow newCut$
14: $\qquad\quad f_{last} \leftarrow f$
15: $\qquad$ **end for**
16: $\qquad$ **if** vio(cut) > 0 **then**
17: $\qquad\quad cut \leftarrow cut \cdot \bar{\delta}$
18: $\qquad\quad finalCut \leftarrow$ CLEAN_CUT(cut)
19: $\qquad\quad$ **return** $finalCut$
20: $\qquad$ **end if**
21: $\quad$ **end for**
22: **end procedure**

Both path mixing cut generators suffer and benefit from the fact that they use the cMIR bound substitution heuristic. On the one hand they can not generate quite as good cuts as the flow path cut generator in some situations because in the flow path cut generator the bound substitution is always done in a way that leads to the most violated cuts. On the other hand they can make use of improvements to the cMIR cut generator such as the improved bound substitution from section 5.2. See section 6.5 for a computational evaluation of the quality of these cut generators.

6. Evaluation

6.1. Evaluation Methods

6.1.1. Empirical Analysis of Algorithms

Since several decades researchers in operations research and other fields struggle with the problem of how to evaluate algorithms. The classical approach is complexity theory that looks at algorithms in a strictly formal way and proves asymptotical bounds. Unfortunately it is a known fact that worst-case and also average-case complexity results for a sophisticated algorithm usually are both hard to obtain and not very enlightening about the real runtime of an implementation of the algorithm. Therefore the approach in this thesis is to use empirical analysis of algorithms (as discussed in [53] and [74]). Empirical means in this context: experimental testing of hypotheses.

Empirical analysis of algorithms has two big advantages. The first is that it can be used to measure the real impact an implementation of an algorithm has on the overall performance of a system that uses it. In the context of this thesis this means that we can test whether the cut generators help the MIP solver to meet the expectations of the users. The user expects from an MIP solver that it solves a given problem instance fast or at least finds a reasonable good solution with a small duality gap. He also expects it to work correctly within the tolerances of the solver. The cut generators in a solver have a big influence on both of these expectations. The second advantage is that empirical studies can be used to test hypotheses. This means that experimental testing can also help researchers to a better understanding of algorithms and relations between algorithms. In this thesis most hypotheses are concerned with the performance of separation algorithms in relation to other separations algorithms or other implementations of the same algorithm.

Empirical analysis has a number of pitfalls that, if not evaded, can easily result in wrong conclusions. There are several publications that give hints and state rules one should obey when performing empirical analysis of algorithms, for example [55] and [34]. One large problem is that the runtimes of two algorithms are influenced by many factors which make a fair comparison very hard. In the following we describe the experimental setups used in this thesis and discusses their strengths and weaknesses. Another big problem is that empirical analysis depends on the problem instances used for the experiments. The next section discusses the problem instances used in this thesis and justifies their usage.

In the context of this thesis, empirical analysis of algorithms is used to compare cut generators implemented in the same framework. This supports a fair comparison between the cut generators. We do not compare the implemented cut generators to implementations in other solvers. The reason for this is that in an MIP solver a large number of components have a strong influence on the results of the solver. Even very small differences in the solvers can result in huge differences in the overall performance. Mapping differences in the results to a specific component of the solver, say a cut generator, is sometimes possible but typically they have several reasons. It is, of course, viable to compare the performance of MIP solvers to each other but only to evaluate the overall performance of the solvers, not to evaluate the performance of the cut generators. Even if one compares the dual bounds after the root node the cut generators are not the only components that influence these results. IP and LP preprocessing techniques might have a major impact on the dual bound even if not a single cut is generated. But also the way in which the cut generators are called, how many rounds of cuts are generated, when cut generation stops, and other implementation details influence these results.

In two situations we divert from not comparing to cut generators implemented outside of the framework described in chapter 5 and not even implemented in the MOPS MIP solver. The first is that we compare the results of our cut generators to the corresponding implementations currently used in the MOPS MIP solver. This is done to show the progress achieved through this thesis. The second situation is that we compare the results of our cut generators to results reported in papers about these cut generators. Although the comparison is not fair because different solvers with very difference settings are used these comparisons can be used to justify the claim that our cut generators are capable of competing with the original implementations.

6.1.2. Problem Instances

When performing experimental analysis of algorithms one has to decide on the set of test problems to use, the *test set*. In principle there are three possibilities: First, to use random generated instances, second to use public test sets, or third to use a proprietary collection of test problems. Table 6.1 lists pros and cons for these three alternatives based on a similar (but outdated) table in [34] and pitfalls pointed out in [55].

Random	Public	Proprietary
- Usually do not represent real-world behavior	+ Can consist of real-world problems	+ Can consist of real-world problems and/or problems tailored/selected towards the experiment
+ The population of the problems is known and can be controlled, statistical analysis is more reliable	- Are usually not representative and may contain problems that are not relevant to an experiment	- Removing or adding single instances may influence the results very much
- There is danger to evaluate properties of the random instances instead of properties of the algorithm	- The origin of problems sometimes is not known	- Other researchers can not compare the results with their own
- A lot of work is needed to design a good random instance generator	+ The problems and their characteristics including optimal solutions can be obtained easily from the internet	- The problems and their characteristics have to be collected

Table 6.1.: Pros and cons for random generated, public and proprietary sets of problem instances

Random generated test problems have the big advantage that, as stated by Lin and Rardin in [63], they allow statistical conclusions about all problems that can be generated by a certain random instance generator. Random instance generators come in two flavors, those that perturb the data of real-world instances or try to mimic them and those that generate completely synthetic instances. Examples for instance generators are the one for capacitated lot-sizing problems described in appendix II of [49] or the generator for small hard 0-1 problems described by Cornuéjols and Dawande in [33].

Public test sets might also contain random generated problems but usually many of the problems in these sets are real-world instances. Their big advantage is that they consist of a variety of different problems. This helps when trying to evaluate the robustness of implementations of algorithms, that means their capability to deal with many different problem types. This advantage is lost when only some of the problems in a problem library are used. On the other hand it seams a waste of computing time to work on problem instances that are not suited for a certain method. Nevertheless we claim that all instances should be used to capture situations where a method, although not meant to be used with a certain problem type, spends a lot of computation time trying to do something useful but fails. Our opinion is that leaving out instances should be considered very carefully and only used as a last resort. Another problem is that usually public problem sets are biased towards hard problems because easy to solve problems are typically not considered interesting. Fortunately, instances considered hard in the past are often easy today. So combining old and new public test problems can make up for this disadvantage.

Proprietary test sets have the advantage that they can capture new trends, for example larger problems, that are not yet present in the public test sets. They can also be used to show that there are problems where new algorithms have their strengths. The results on a proprietary test set can easily be influenced by removing or adding problems, therefore comparing averages or similar metrics for them is even more problematic. Nevertheless, they are sometimes needed in addition to public test sets for the aforementioned reasons.

The approach of this thesis is to rely on two test sets, one consisting of a combination of instances from several public test sets and the other consisting of proprietary instances. We justify this decision with the aim of this thesis to improve performance of MIP solvers on practical instances.

The public problem instances used in this thesis are those available on the websites of the public test sets MIPLIB3 [23], MIPLIB2003 [6], MITTELMANN [75] and LOTSIZELIB [20]. By adding the LOTSIZELIB problems, the test set gets slightly biased towards

	4LIB	MOPSLIB
BIN	35	0
INT	4	0
MIB	94	16
MIP	36	9
total number of problems	169	25
(min, max) variables	(18,204880)	(147,1798971)
(min, max) constraints	(6,159488)	(231,2039724)
(min, max) nonzero elements	(40,1024059)	(399,4864543)
optimum unknown	16	13

Table 6.2.: Summary of problems in 4LIB and MOPSLIB

lot-sizing problems. As path-based cut generators are an important part of this thesis and they typically work well on lot-sizing instances this increases the number of instances relevant for this thesis. As our set of public test problems consists of the problems from four public test sets it is called 4LIB. If not stated differently, it is used for all experiments.

Table C.1 on page 161 in the appendix lists the problem instances in 4LIB including the problem type. The table distinguishes four problem types: pure binary problems (BIN), pure integer problems (INT), mixed integer binary problems (MIB) and general mixed-integer problems (MIP).

The second set of problems used is a proprietary set further on called MOPSLIB. It consists of problem instances collected by the DS&OR Lab at the University of Paderborn for testing the performance of the MOPS MIP solver. The instances all have a real-world background. It is used in addition to 4LIB because it contains some very large instances that more and more often come up in industry projects. These instances have up to 1,798,971 variables and can only be solved using a 64bit architecture. So all tests with the MOPSLIB are performed using a 64bit version of the MOPS solver. Some of the instances are from project partners of the DS&OR Lab that do not want their data to be published. Therefore the instances in MOPSLIB can not be given to other researchers for experimentation. Characteristics for the instances in MOPSLIB are given in table C.2 on page 162 in the appendix. Table 6.2 lists a summary of the instances in 4LIB and MOPSLIB.

6.1.3. Computational Experiments and Performance Measures

This subsection discusses experimental setups and performance measures for testing cut generators. We start with some definitions. The tests in this thesis are designed to compare a set $\mathcal{S}$ of *solver versions*. By a solver version we mean a MOPS executable with a set of parameter settings. To perform the tests, a set $\mathcal{P}$ of test problem instances is needed. For some tests we need a solution to the problems in $\mathcal{P}$. A solution s_p in the set of solutions $\mathcal{I}_p$ for a problem p consists of an objective function value $\bar{z}_p$ and a pair of vectors $(\bar{x}_p, \bar{y}_p)$. It is called ε-optimal if $\bar{z}_p - \varepsilon < z_p^{MIP}$ where z_p^{MIP} is the objective function value of an optimal solution to p. If all constraints are violated by at most ε it is called ε-feasible and if all elements of $\bar{y}_p$ satisfy $1 - \varepsilon < |\bar{y}_p^i - \lfloor \bar{y}_p^i \rfloor| < \varepsilon$ it is called ε-integer. The optimal solution to the LP relaxation of the initial problem p is denoted by z_p^{LP}. In the following, for each test used in this thesis we describe the experimental setup and discuss advantages and disadvantages.

The k-round Test

The classical approach to test the quality of cut generators, for example used in [51], [70], and [93], is to compare the dual bound (LP bound) in the root node after adding cuts for a number of rounds. In addition to the dual bound, usually the number of cuts generated and the time spent in the rounds is reported. In this thesis we call this experimental design a *k-round test* where k is the number of rounds.

In our experimental setup for k-round tests we usually test one cut generator and deactivate all others. Nevertheless, we use all preprocessing methods such as probing and bound reduction with their default settings. An exception are the path-based cut generators that in our implementations do not generate cuts from single rows. Therefore we always test them together with the cMIR cut generator. In figure B.1 on page 155 we show a typical configuration file used in a k-round test.

The conclusions that can be drawn from a k-round test are limited. As already pointed out by Margot in [71], this test is mostly useless for measuring accuracy because invalid cuts would only become apparent if they lead to infeasibility or a dual bound worse than the optimum. Concerning the efficiency, the problem is that if more cuts are found and more rounds can be done, the separation obviously takes longer. But usually measures used in comparisons do not consider the trade-off between time and quality. For most instances they do not have to because the time spent in the cut generators is extremely small, especially compared to the time needed to resolve the LP relaxation.

Quite surprising is the fact that a k-round test can not even measure the quality of separation algorithms in all cases. There are several reasons to support this claim. Firstly, it is obvious that a k-round test does not give any insight if the optimal objective function value for the LP relaxation and the MIP are the same. Another is that after the first round of the cut generation different algorithms get different input, i.e. it might happen that a very good algorithm accidentally runs into an LP relaxation solution that can not be cut off with a cut of the family it uses. The reason described next is even more substantial. Example 6.1 shows that even if the dual bound is better this does not mean that the formulation has improved more and hence the optimal solution will be obtained faster.

Example 6.1. *We assume an instance of the constant capacity lot-sizing problem (called LS-CC in [85])*

$$
\begin{aligned}
&\min \sum_{t=1}^{n} p_t x_t + \sum_{t=0}^{n} h_t s_t + \sum_{t=1}^{n} q_t y_t \\
&s_{t-1} + x_t = d_t + s_t && \text{for } 1 \le t \le n \\
&x_t \le C y_t && \text{for } 1 \le t \le n \\
&s \in \mathbb{R}_+^{n+1}, x \in \mathbb{R}_+^n, y \in \{0,1\}^n
\end{aligned}
$$

with $n = 4$*,* $(p,h,q) = \{6,4,3,6,100,1,1,1,1,20,20,20,20\}$*,* $d = \{7,6,5,7\}$ *and* $C = 10$*. For this instance two separation algorithms generate different cuts. Algorithm 1 generates the MIR cuts*

$$
\begin{aligned}
s_0 &\ge 7 - 7y_1 \\
s_1 &\ge 6 - 6y_2.
\end{aligned}
$$

The dual bound after adding these cuts is $z_1 = 172$ *with the solution* $y = (1, 0.5, 1, 0)$*. Branching on* y_2 *results in two nodes that can not be pruned right away.*

Algorithm 2 generates one MIR cut and one mixing cut

$$
\begin{aligned}
s_1 &\ge 6 - 6y_2 \\
s_1 &\ge 11 - 8y_2 - 3y_3 - 2y_4.
\end{aligned}
$$

After adding these cuts the dual bound is $z_2 = 167$ *with the solution* $y = (0.7, 1, 1, 0)$*. In this case branching on* y_1 *results in two nodes that are integral and one is the optimal solution of* 173*. So in this example it can be seen that although* $z_1 > z_2$ *the improvement* *of the formulation by algorithm 2 is clearly better. Of course this example is artificial in the sense that two algorithms that generate exactly these cuts are not likely to be used. But the example hints at how situations like this can happen in more complex souroundings.*

Because of these reasons evaluating separation algorithms solely based on k-round tests is problematic. Nevertheless this experimental design can be used and gives important insights if the weaknesses of the method are considered. A large advantage of k-round tests is that they can be done very fast. As the solution of the LP relaxations can be restored from a saved basis file and the separation algorithms usually need at most a few seconds, many instances can be tested in less than 10 minutes. This makes this experimental design attractive for comparing and testing variants of separation algorithms. When using a 1-round test the time gets even less. It can be used very well to investigate whether two variants of a cut generator generate roughly the same cuts (as done in [17]) or to see whether changing a small detail in the algorithm changes the outcome significantly. The disadvantage of a 1-round test is that the effect of separating inequalities with a rank (see section 2.5) larger than one can not be tested. When interpreting the results of a k-round test, minor differences should not be considered. What can be considered worth an interpretation are large differences in the dual bound, the runtime, or the number of cuts. Important findings in a k-round test should be verified using other experiments.

The k-hour Test

Another classical test to evaluate cut generators is to include them in a branch-and-cut algorithm of an MIP solver and run it until a time limit is reached. This has the advantage that the quality and the efficiency of the cut generators are tested. Furthermore, diversity is also tested to a certain degree when comparing cut generator configurations with each other. Finally, accuracy is tested because it is implicitly checked whether the optimal solution is cut off.

In this thesis we call this experimental setup a k-hour test where k is the number of hours that is used as a time limit for the solver. The main performance measure reported usually is the time to solve the problem instances in the test set. The obvious advantage of this experimental design and this measure is that it reflects the situation that a user of a solver cares for. The downside of it is that the performance results in this test do not solely depend on the cut generators. They are also influenced (among other factors) by the primal heuristics used, the branching strategy, and the node selection strategy of the solver. The results are also subject to some randomness, for example, if the addition of a certain cut results in a failure of a heuristic to find a good feasible solution early in the branch-and-cut tree.

When evaluating the results, other measures than time are sometimes investigated. These measures are number of nodes, number of LP iterations, and some form of *gap* for those instances that could not be solved within the time limit. The number of nodes usually is a very bad indicator of performance especially when comparing different solvers. How much time is spent in a node strongly depends on the techniques used in the node and on the size and difficulty of the LP relaxation. As adding cuts increases the formulation of the problem adding more cuts might result in an increase of the average time needed to resolve a node in the branch-and-cut tree. So the number of nodes does not say anything about the trade-off between solving small nodes fast or large nodes slow. The number of LP iterations can give a hint of how much work the solver did but is a less direct measure than total time and excludes the effort spent in other parts of the solver.

Reporting some form of gap when the solver failed to solve an instance within the time limit does make sense because the gap is an important information for the practical use of the results. A typical gap is the *duality gap*, in a slightly different form also used in [85]. It is defined as

$$\Gamma_p^{duality} = \frac{|\bar{z}_p - \underline{z}_p|}{|\bar{z}_p| + \varepsilon}$$

where $\bar{z}_p$ is the best known primal bound and $\underline{z}_p$ is the smallest (when minimizing) dual bound of the nodes in the node list when the solver is stopped. A very small number ε is added to avoid divison by zero. Note that some solvers report other gaps during the execution of the algorithm.

In the k-hour tests for this thesis we use default settings for all solver parameters that are not related to the cut generators based on row relaxations. This means that we use all preprocessing techniques and cover, implication, clique and Gomory cuts together with the cuts we activate for the experiment. In the description of the configuration we use the *state-of-the-art (SOTA)* configuration as a reference. The state-of-the-art configuration for row relaxation-based cuts is to use flow cover, cMIR, and flow path cut generators with a traditional aggregation and bound substitution strategy. In section 6.6 we compare this to the *improved SOTA* configuration where we additionally use our new path-based tightest row aggregation and improved bound substitution. Figure B.2 on page 155 shows a typical configuration file used in a k-hour test.

A slight variation of the k-hour test is the *truncated k-hour test*. In this experimental setup the solver is given the optimal objective function value z_p^{MIP} as a primal bound.

By doing this the solver is only used to prove the optimality of this solution. This has the advantage that primal heuristics have no impact on the solution process and the influence of the branching decisions is also reduced. It emphasizes the influence a better dual bound obtained by adding cuts has on proving optimality. The effect of an improved formulation with which it is easier to find good solutions in the tree and in primal heuristics is eliminated. Truncated k-hour tests can be used in addition to normal k-hour tests to validate their results.

The ε-validity Test

This is an experimental design that can be used to test the accuracy of cut generators. The idea is to check for each cut generated in a k-round test whether it cuts-off a given integer solution $\bar{z}_p$ by more than a certain tolerance ε. We call such a cut ε-invalid. The solutions for these tests in this thesis are generated by running MOPS with all preprocessing (LP and IP) deactivated and a time limit of 10 hours. Additionally we also decrease the tolerances in MOPS to require solutions to be ζ-optimal, ζ-integer and ζ-feasible with $\zeta = 1 \times 10^{-7}$. The result of this is that the accuracy of the solutions mainly depends on the accuracy of the MOPS LP solver. Only solutions that are within the optimality tolerance of MOPS are used for the test, these are 113 out of the 4LIB test set and 8 out of the MOPSLIB test set.

As a result of this test we report the instances where ε-invalid cuts are generated. If no cut is ε-invalid for the solutions $\mathcal{I}$ to the problems in $\mathcal{P}$ we say that the implementation is ε-accurate for $(\mathcal{P}, \mathcal{I})$ in this k-round test. This way of testing for accuracy has several weaknesses like the problem that for some instances generating valid solutions is very hard. Its inherent strength is that it allows us to say that the results of a k-round test are not influenced by invalid cuts and that the implementations compared have the same minimal standard for accuracy. As it can be done relatively fast it can also be used very well for debugging. An improvement to this test would be to use several optimal or near optimal solutions. Another way of testing the accuracy of separation algorithms is described in [71]. It is called *the random dives test* and promises much better insight into the accuracy of the tested separation algorithms. Unfortunately it needs 0-feasible solutions and generating these can be very troublesome. Another drawback of this method is that it needs some implementation effort.

6.1.4. Presentation

The results of an experimental analysis of algorithms usually are many pages full of data. Although all this data is needed for an in-depth analysis, interpretation of the results is very much simplified by a sophisticated presentation of the results. Unfortunately, using a bad presentation can lead to wrong conclusions. Therefore the presentation method has to be selected very carefully.

Researchers in computational MIP apply many different presentation methods to their results. One is to use sums, arithmetic or geometric means, medians, and quartiles to aggregate the results for many problem instances into a single number (or a few numbers) for each algorithm that can easily be compared. The problem with these methods is that they tend to be influenced by single or just a few instances and that it has to be decided how to deal with instances where the algorithm fails. See [42] for a discussion of these methods. Another approach is to rank the performance of algorithms for each instance and then report average ranks. The problem with these methods is that information about the size of the difference between algorithms is lost. An example for this presentation can be found in [65].

Using statistical tests to compare algorithms is also a viable approach, it is for example used in [63]. Recently Margot brought it back to attention by using it in [71]. The problem with this approach is that much of the transparency is lost and the evaluation of the outcome of statistical tests might be hard to understand for readers who are not familiar with the topic. We choose to use the presentation techniques described below in this thesis in addition to tables with detailed results. The graphical displays for both of these presentation methods follow the design principles for the visual display of quantitative information by Tufte [90].

Gap Difference Diagrams

The *gap difference diagram* is a presentation method newly introduced in this thesis. It is based on first simplifying the interpretation and presentation of k-round tests by computing *gap closed ratios*. These ratios are designed to give an idea about how successful a separation algorithm was in closing the gap between the initial solution of the LP relaxation and the IP optimal solution. Reporting just the dual bound has the problem that we can not compare two instances in the same scale. We consider two slightly different ratios for this purpose. The first is the *absolute gap closed*:

$$\varrho = \frac{z^* - z_{LP}}{z_{MIP} - z_{LP} + \varepsilon}$$

where z_{LP} is the initial solution of the LP relaxation, z^* is the dual bound after k rounds of cuts and z_{MIP} is an optimal solution to the problem instance or the best known primal bound. The very small number ε is added to avoid a division by zero. The other ratio is the *relative gap closed*:

$$\zeta = \frac{z^* - z_{LP}}{z_{best} - z_{LP} + \varepsilon}$$

where the same notation as above is used and z_{best} is the best dual bound that any of the compared algorithms achieved. The advantage is that for ϱ_{rel} no optimal solution is needed and the best algorithm(s) in a comparison for one instance can easily be identified because their ratio is 1.00. For the presentation in this thesis we use the *absolute gap closed* because we know the optimal solutions or reasonably good bounds for all problems in our test sets.

The *gap difference diagram* is a visual display that helps to compare the results of two k-round tests performed with two solver versions A and B. It shows the *gap closed difference* $\Delta_s = \varrho_s^A - \varrho_s^B$ for each instance s in the test set using a bar chart. A positive value of Δ_s means that algorithm A closed more of the gap than algorithm B for instance s whereas a negative Δ_s implies that algorithm B closed more of the gap. To improve the readability, the instances are sorted by Δ_s and labels are added to the instances where the first time $\Delta_s \leq 0.001$ and $\Delta_s \leq -0.001$. The result is a diagram as shown in figure 6.1.

An valid interpretation of the example in figure 6.1 is that algorithm A overall performs better than algorithm B because for many instances more of the gap is closed. For one instance the difference goes up to more than 80%. On the other side of the diagram we see that for a few instances worse bounds are achieved. To make a final conclusion whether to use algorithm A or B it is advisable to check (using an 1-hour test) that the differences for these instances do not result in a situation were an instance can not be solved within reasonable time. The numbers in the diagram can be interpreted in the way that for 92% of the instances algorithm A closes more or the same amount of the gap and it is only worse for 8% of the instances.

A major drawback of this visual display is that only two solver versions can be compared. But for comparing two solver versions it gives a very good impression of how different two algorithms are and for how many instances the results of one algorithm are better than the results of another one. It also gives information whether they differ by a small

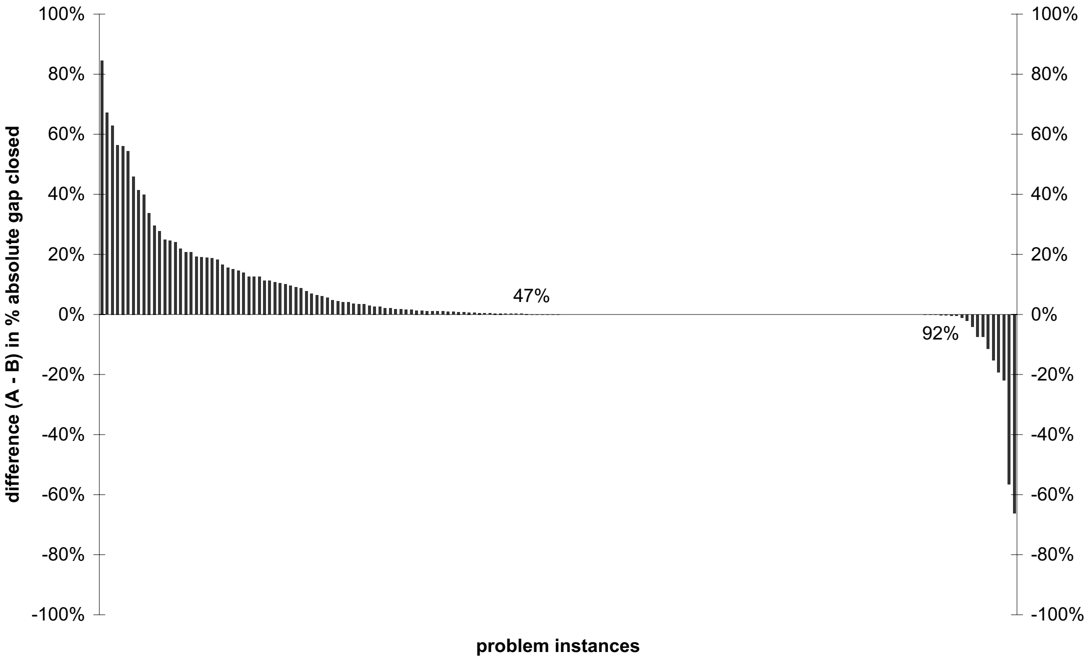

Figure 6.1.: An example of a gap difference diagram comparing the results of a 10-round test for two solver versions A and B.

or large amount. Together with a table sorted by Δ_s, as we show in the appendix of this thesis, it can help to identify instances where improvements are needed.

Performance Profiles

In this thesis *performance profiles* as introduced by Dolan and Moré in [42] are used to present the results of k-hour tests. The performance profile of an MIP solver is based on a performance ratio:

$$r_{ps} = \frac{t_{ps}}{\min_{i \in \mathcal{S}}\{t_{pi}\}}$$

Here t_{ps} is the time needed to solve problem p with solver version s. We define that $r_{ps} \in [1, r_M]$ and that $r_{ps} = r_M$ only if problem p is not solved by solver s within the time limit. What r_{ps} actually says is that if for example $r_{ps} = 4$ then solver version s solves problem instance p four times slower than the fastest solver version in this comparison. The performance profile of a solver is the cumulative distribution function for a performance metric, so as we use r_{ps} as performance metric, the performance profile used in this thesis is

$$\rho_s(\tau) = \frac{1}{|\mathcal{P}|}|\{p \in \mathcal{P} : r_{ps} \leq \tau\}|.$$

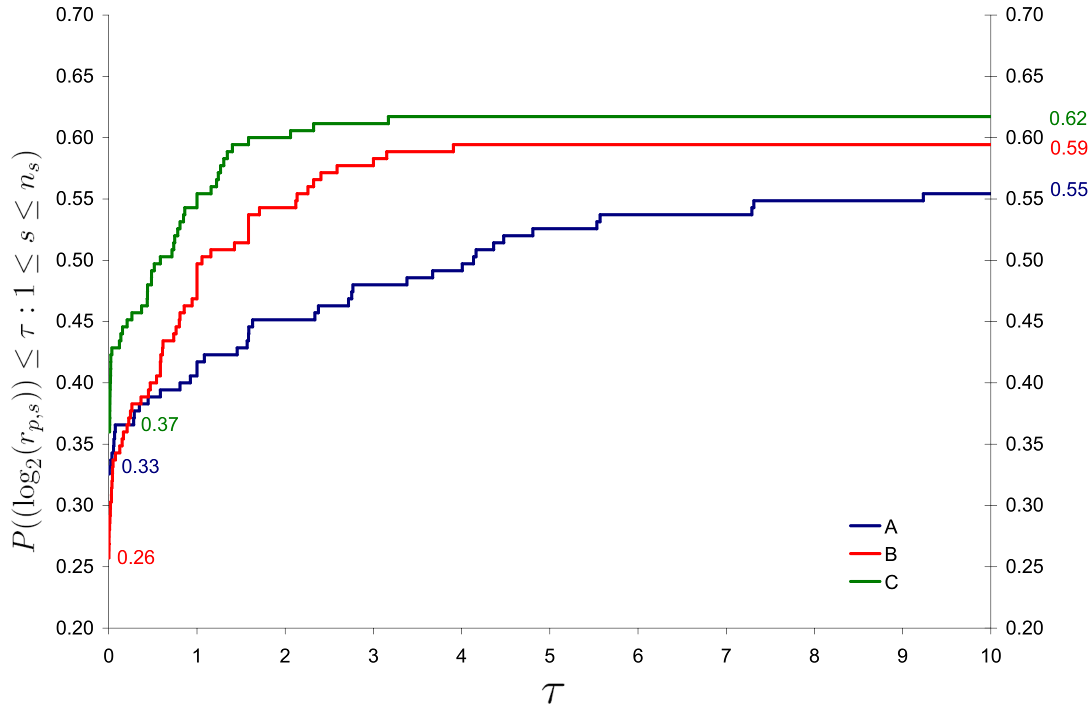

Figure 6.2.: An example of a diagram showing performance profiles for three solver versions A, B and C.

The performance profile of a solver version depends on the set $\mathcal{S}$ of solver versions it is compared with and on the set $\mathcal{P}$ of problem instances. We choose to show the performance profiles on a logarithmic scale as suggested in [42]. The performance profiles are displayed in a digram as shown in figure 6.2.

The small numbers in the diagram show the values for $\tau = 0$ and $\tau = r_M$ (or the appropriate logarithmic value). This simplifies observing how many instances were solved fastest and how many instances were solved at all. For the further interpretation of a diagram with performance profiles the general rule is that a solver version is better if its profile is further up and to the left of the diagram. So from the example in figure 6.2 the conclusion can be drawn that solver version C performs best. From the numbers we can see that it solves about 37% of the instances fastest. Furthermore they indicate that is solves about 62% of the instances within the time limit. For the solver versions A and B we can not clearly say which one is better. A solves more instances fastest and B solves more instances within the time limit. An interpretation of this could be that A is faster for some instances but B is better for solving hard instances. Note that we only show a part of the y-axis to focus on the interesting part of the diagram.

The advantages of using performance profiles are explained in detail in [42]. One worth mentioning is that they give the same weight to each instance in the test set so that the interpretation is not influenced by a small number of instances as it is the case when using averages. In the last few years performance profiles enjoyed great popularity in the field of computational optimization and are used in articles of major journals (see for example [64] and [60]).

In addition to a diagram with the performance profiles we show tables with detailed results for each solver version tested. These tables can be found in the appendix. Note that some solutions reported optimal by the solver differ from the optimal solutions listed in table C.1 and table C.2. We attribute this to the default values of the tolerances in the MOPS MIP solver.

6.1.5. The Test Environment

All computational experiments for this thesis are performed on a personal computer (PC) with an Intel Core 2, 2.40 Ghz, CPU and 8 GB random access memory (RAM). The operating system of this machine is Windows XP Professional x64. The code of the described cut generators, the framework, and a new version of the cut pool for MOPS 10.0 is linked to a MOPS version 9.19 library (LIB) and compiled using release settings of the Intel Fortran Compiler 10.0.026. For tests using the 4LIB test set a 32-bit binary is generated as this is the usual way MOPS is distributed. For experiments with the MOPSLIB test set a 64-bit binary is generated because some of the instances need to address more memory than the 32-bit version can allocate. If not stated differently, the MOPS parameters are at their default settings that can be found in [77] and [78]. Performance measures are obtained from the MOPS statistic files and these, as well as MOPS message and option files, are archived by the author.

6.2. Accuracy Evaluation

In this section we evaluate the accuracy of the implemented cut generators. For this purpose we do ε-validity, 10-round tests with $\varepsilon \in \{1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}, 1 \times 10^{-7}\}$ for each cut generator. As mentioned in section 5.6, our path-based cut generators do not generate cuts from single rows. Therefore in the tests for flow path cut, uPMC and cPMC generators the cMIR cuts are also activated.

	1×10^{-4}	1×10^{-5}	1×10^{-6}	1×10^{-7}
flow cover	-	-	-	-
cMIR	-	-	b4-12b rgn	b4-12b rgn
flow path	-	-	rgn	b4-12 b4-12b rgn
uPMC	-	-	rgn	b4-12 b4-12b rgn
cPMC	-	-	rgn	b4-12 b4-12b rgn
	ε			

Table 6.3.: Results of the accuracy tests. The table shows the names of the instances for which invalid cuts were generated.

In section 6.1 we explain the ε-validity test and mentioned that the solutions for this test are generated with an accuracy of 1×10^{-7}. Therefore testing for smaller numbers of ε does not make sense. The primal tolerance of the MOPS LP/MIP solver is by default 1×10^{-4}. Hence we require that all cuts generated are at least 1×10^{-4} accurate, that means that for no instance of our accuracy test set a cut is generated that violates our accurate optimal solution by more than 1×10^{-4}. As one can see in table 6.3, which lists the instances where ε-invalid cuts are generated, this is the case for all of the cut generators.

The results for smaller values of ε indicate that numerical issues lead to slightly violated cuts for a very small number of the 113 instances. Whether the reason for this lies in the LP solver or the cut generation can not be said from this experiment. Overall this test shows that the implementations are accurate enough to be used in a commercial MIP solver. Nevertheless note that this test only uses a subset of our test problems because

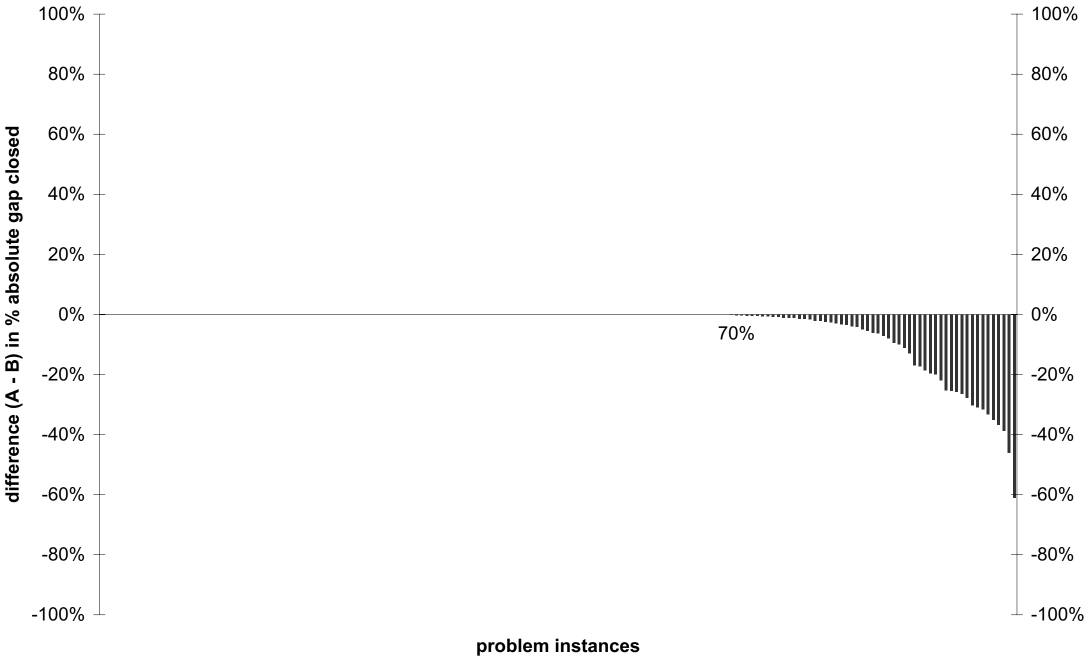

Figure 6.3.: Comparison between the default version of the flow cover cut generator that does not generate cuts from aggregated rows (A) and a version that does (B).

we were not able to generate accurate optimal solutions for the other problem instances. Among the other instances are numerically difficult instances that cause a lot of problems for LP solvers and cut generators. For these instances this test could not be used but it was checked that no obviously invalid cuts are generated.

6.3. Evaluation of the Flow Cover Cut Generator

6.3.1. Implementation Details

In this section we show how much impact the implementation details described in section 5.3 have on the performance of a flow cover cut generator. The first thing we want to investigate is whether generating flow cover cuts from aggregated rows is advantageous. In figure 6.3 we see a gap difference diagram for the comparison of the default version of the implemented flow cover cut generator that does not generate flow cover cuts for aggregated rows and a version of the cut generator that does.

The gap difference diagram indicates that generating flow cover cuts from aggregated rows results in better dual bounds for about 30% of the problem instances in the 4LIB test

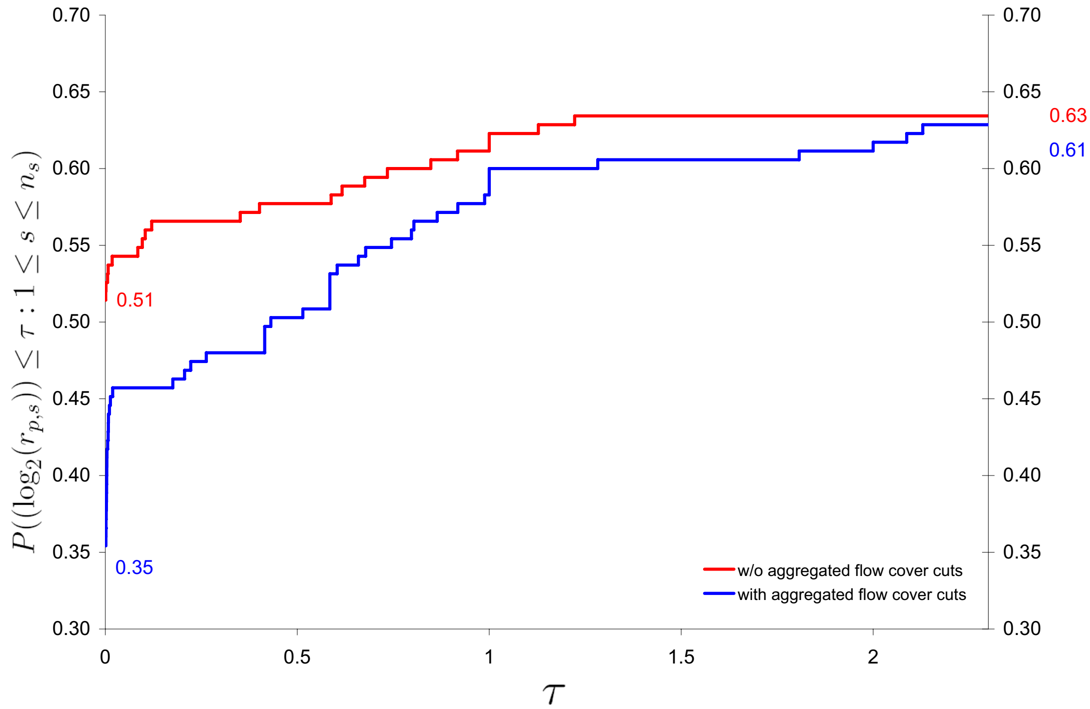

Figure 6.4.: Performance profiles for two solver versions, both using an improved SOTA configuration, one with aggregation for the flow cover cuts, the other without.

set. The detailed results in table D.1 on page 164 show that getting these improved dual bounds needs much more computation time. The sum over the time spent in supernode processing for all instances (not only those listed in table D.1) is 924.99 seconds without aggregation and 1545.40 seconds with aggregation. Furthermore we assume that the flow cover cuts found through aggregation can also be found using cMIR cuts. Therefore we show the results of two 1-hour tests where we compare the improved SOTA solver version with aggregated and without aggregated flow cover cuts. The results of these tests are shown in the performance profiles in figure 6.4.

From the performance profiles we can see that the version without aggregated flow cover cuts performs clearly better than the one with aggregated flow cover cuts. One reason for this is that although better bounds are achieved in the direct comparison of the two flow cover cut generators alone, in combination with the cMIR cut generator that uses aggregation, the additional time spent to get aggregated flow cover cuts does not pay off.

The detailed results in table D.21 (page 187) and table D.22 (page 190) reveal that with aggregated flow cover cuts the instance `bc1` can be solved. Without aggregated flow cover

Figure 6.5.: Comparison between the default version of the flow cover cut generator that generates cuts out of cuts (A) and a version that does not (B).

cuts this instance is not solved but the two instances `b4-10` and `tr12-30` are. We assume that this is not the effect of a better or worse cut generator but the result of differences in the way the branch-and-bound tree is searched. More important is the fact that for a large number of instances the solution time is much larger because of the time spent in the cut generation. One reason for this might be the exact solution of the flow cover finding knapsack problem. We discuss this later in this section. As a result of these experiments we do not use aggregation for flow cover cuts in the default version of the flow cover cut generator. We do include an parameter to activate it to solve instances where every bit of improvement in the dual bound is needed.

Another aspect concerning the input rows of a flow cover cut generator is whether to generate cuts out of cuts. The gap difference diagram in figure 6.5 shows a comparison of two 10-round tests with different flow cover cut generator versions. In version A, the default version, cuts are generated out of cuts added in previous iterations of the cut generation. In version B only original rows of the constraint matrix are considered as input rows. Note that in neither case aggregation is used.

The diagram shows that the impact of generating cuts out of cuts is fairly small. Only for a few instances better or worse dual bounds are achieved. The computation times

Figure 6.6.: Comparison between the default version of the flow cover cut generator that uses binary variable bounds on integer variables (A) and a version that does not (B).

shown in the detailed results in the appendix (table D.2, page 165) do not increase very much. Therefore we generate cuts out of cuts in the default version of our flow cover cut generator despite its small impact.

Concerning the reformulation of rows we now investigate whether using variable upper bounds on integer variables influences the performance of the flow cover cut generator. To do so we perform two 10-round tests with two versions of the flow cover cut generator, one that does use binary variable bounds on integer variables and one that does not. The results are shown in a gap difference diagram in figure 6.6.

These results indicate that only for three instances the use of binary variable bounds on integer variables yields clearly better dual bounds. See table D.3 on page 165 for detailed results. For the three instances, `ches3`, `ches5`, and `neos671048`, the improvement is large. An inspection of other results in this thesis reveals that the `ches3` and ches5 instances a always solved within seconds and that `neos671048` is also not a very problematic instance. Therefore the result of this experiment is that using binary variable bounds on integer variables in a flow cover cut generator can improve the dual bound of some MIP problem instances but in our test set this extension of the original reformulation approach does not lead to an improved performance.

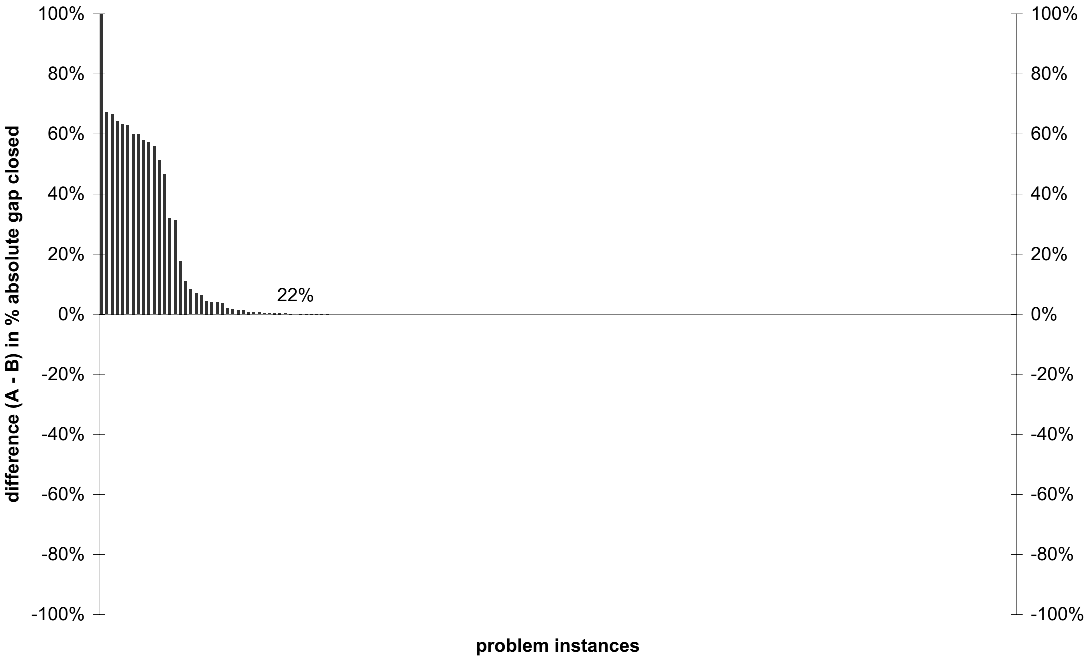

Figure 6.7.: Comparison between the default flow cover cut generator using $y_j^* = \frac{x_j^*}{u_j}$ (A) for variables without a variable upper bound and a version that uses $y_j^* = 1$ (B) for these.

As mentioned in section 5.3 the flow cover finding is the most important aspect of a flow cover cut generator. There we also point out that in the objective function of the flow cover finding knapsack problem using

$$y_j^* = \frac{x_j^*}{u_j}$$

for variables that do not have a variable upper bound is a better choice than using

$$y_j^* = 1$$

as it is implied by the reformulation. We test this by comparing two versions of the cut generator where we used the default version with the first rule mentioned (version A) and a version with the rule implied by the reformulation (version B). We show the results in the gap difference diagram in figure 6.7. Detailed results can be found in table D.4 on page 166.

As expected using version A yields significantly better results for many instances. Surprisingly, the dual bound is the same or better for all but one instance for which the

Figure 6.8.: Comparison between the default version of the flow cover cut generator that uses an exact algorithm to solve the flow cover finding knapsack problem (A) and a version that uses a heuristic (B).

difference in the bound is very small. This is an indication that this implementation detail is very important for a good flow cover cut generator.

Another implementation detail that is connected with the flow cover finding of the flow cover cut generator is how to solve the flow cover finding knapsack problem. In figure 6.8 we show the results of the comparison of two versions of the flow cover cut generator. The first, version A, uses an exact branch-and-bound-based method for solving the flow cover finding knapsack problem. The second, version B, uses a simple greedy heuristic (described in [80]).

The gap difference diagram shows that for some instances using the exact method results in much better dual bounds. For a few instances the dual bounds are worse. A comparison of the runtime of the two cut generator versions reveals that for all 175 test problem instances version A spends 924.99 seconds in the supernode processing phase and version B 935.07 seconds. For most of the instances the runtime is the same leading to the conclusion that, although a similar amount of time is spent, solving the flow cover finding knapsack problem exactly improves the dual bounds obtained. Note that there is a node limit of 100000 on the branch-and-bound method to avoid getting stuck in a very hard instance.

Figure 6.9.: Comparison between the default version of the flow cover cut generator that uses the rule $L^- = \{j \in N^- : \lambda y_j^* < x_j^*\}$ (A) and a version that uses the rule $L^- = \{j \in N^- : \lambda y_j^* \leq x_j^*\}$ (B).

Another decision that has to be made in the flow cover cut generator is which variables to put into the set L^-. As mentioned in section 5.3, $L^- = \{j \in N^- : \lambda y_j^* < x_j^*\}$ is a reasonably good rule for this as it maximizes the violation of the cut and thus increases the chance to find a violated inequality. If we use the same rule except that we use *less than or equal* ($\leq$) instead of *less than* ($<$) this would also maximize the violation. We compare these two versions in the gap difference diagram shown in figure 6.9.

These results indicate that there is a measurable difference between the two solver versions. They also indicate that using the rule with *less than* leads to better results, but not for all instances. Therefore we choose to use the rule with *less than* for the default version of the flow cover cut generator.

Finally we want to investigate how important the lifting is for the quality of the flow cover inequalities. To do so we compare the results of two 10-round tests. In the first version of the flow cover cut generator we generate LSGFCIs (version A) and in the second just SGFCIs, i.e. we do not use lifting (version B). The results are shown in figure 6.10 and table D.7 on page 169.

For some instances lifting makes a difference but in general it does not. A reason for this

Figure 6.10.: Comparison between the default version of the flow cover cut generator that uses lifting (A) and a version that does not (B).

might be that the flow cover finding does not consider the lifting process. Nevertheless, as the lifting is sequence independent it can be done very quickly and does not increase the runtime very much. Thus we use it in the default version of our flow cover cut generator.

6.3.2. Comparison to the Previous Flow Cover Cut Generator

In this section we compare the described flow cover cut generator to the one that is currently used in the MOPS solver. It is an implementation by the author of this thesis but done 3 years ago and not in the framework described in this thesis. We compare the results of two 10-round tests, one with the new flow cover cut generator (without aggregation) (A) and one with the old flow cover cut generator (B). Figure 6.11 shows the results in a gap difference diagram and table D.8 on page 170 lists the actual numbers in a sorted table.

The gap difference diagram shows that for almost all instances the new cut generator performs at least as good as the old one. For some instances the improvement is extremely large. The table with the detailed results shows that these instances are pure binary problems and that the old cut generator did not generate cuts at all. The reason for this

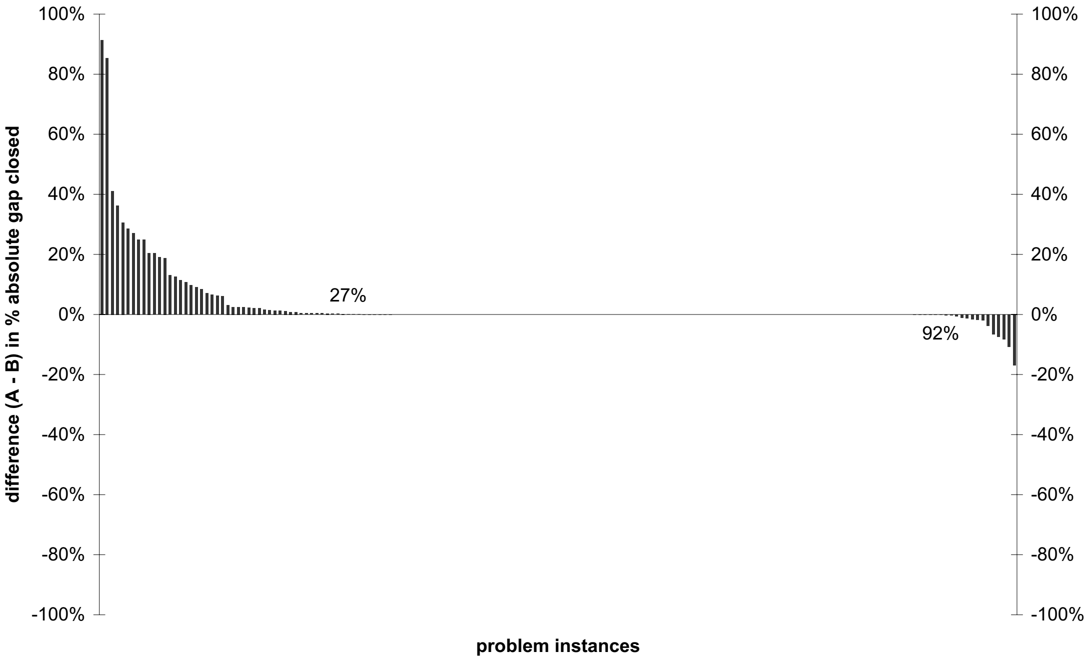

Figure 6.11.: Comparison between the new (A) and the old (B) flow cover cut separation algorithm of MOPS .

is that the old cut generator did not use pure binary rows as input. This makes sense from a theoretical point of view but the practical results indicate that, although flow cover inequalities are not intended to generate cuts for pure binary rows, our implementation generates them quite successfully. Most of the implementation details described in this thesis were actually identified when the old cut generator was implemented or from the comparison of the old and the new cut generator.

6.3.3. Comparison to Published Results

Here we compare the results obtained with the described flow cover cut separation algorithm to the results reported by Gu, Nemhauser and Savelsbergh in [51]. In table 6.4 we list those problem instances of their test set publically available, the results they reported, and our results. Unfortunately the comparison is not fair because different solvers are used and we do not know the exact settings in which their results were obtained. Nevertheless these results can give a hint whether the implementation by Gu, Nemhauser, and Savelsbergh does something totally different than our implementation. To ease the identification of the best result for each instance we print it in bold face. For MOPS , the column *cuts gen.* lists the cuts generated, the column *cuts sel.* lists the cuts selected by the cut pool.

	Gu et al. 1998		MOPS		
name	cuts	XLP	cuts gen.	cuts sel.	XLP
egout	14	556.4	46	17	**565.9**
fiber	360	381837.8	156	89	**382576.8**
fixnet3	83	**51880.8**	44	17	51611.3
fixnet4	190	8307.8	257	57	**8405.8**
fixnet6	169	3507.4	267	52	**3564.3**
khbo05250	122	106608880	331	88	**106724316.4**
mod013	54	267.3	80	31	**269.6**
modglob	366	20662084	282	80	**20675896.1**
rentacar	60	**29219168**	29	12	29017833.3
rgn	74	**64.6**	29	20	48.8
set1al	400	**15867.2**	269	180	15464.5
set1cl	400	**6484.2**	269	180	5996.5

Table 6.4.: Comparison to the results from [51].

The results show that for more than half of the instances our cut generator achieved a better dual bound than the one by Gu, Nemhauser and Savelsbergh described in [51]. The reason for this is not necessarily that the cut generator is better, differences between the underlying solvers, for example in IP and LP preprocessing, might as well be the reason. For five of the instances the Gu, Nemhauser, Savelsberg cut generator achieves better results, again the exact reasons are not clear. We know from experimentation that using a different configuration of the cut pool can lead to much better dual bounds with our cut generator.

Based on these results we claim that the cut generator described in this thesis is able to recreate or improve upon the results in [51]. Obtaining these results is only possible with close attention to the implementation details not mentioned in any publication before but described for the first time in section 5.3 of this thesis.

6.4. Evaluation of the cMIR Cut Generator

6.4.1. Implementation Details and Algorithmic Improvements

In this section we show how much impact the implementation details and algorithmic improvements presented in section 5.4 have on the performance of the cMIR cut generator. The first implementation detail we want to investigate is whether generating cuts out of cuts improves the performance of a cMIR cut generator. Figure 6.12 shows a gap

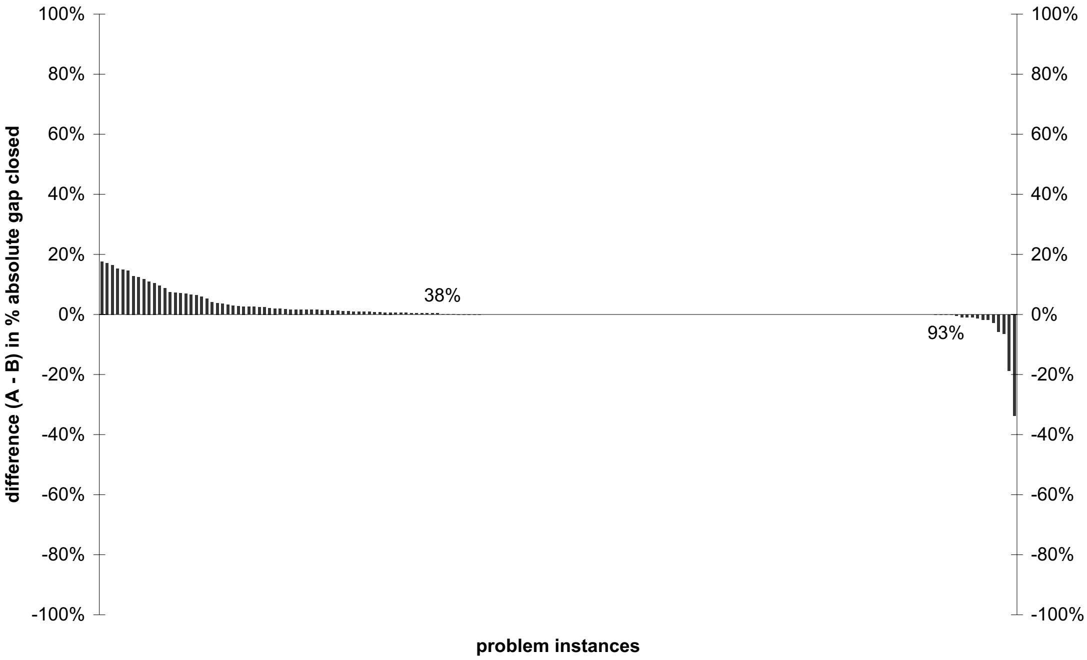

Figure 6.12.: Comparison between the default version of the cMIR cut generator (A) and a version that does not generate cuts out of cuts (B).

difference diagram for a computational experiment where we compare a version of the cMIR cut generator that generates cuts out of cuts (A) with one that does not (B).

The diagram shows that for many instances slightly better dual bounds are obtained by generating cuts out of cuts. One reason for this is, as discussed in section 5.6, that some path inequalities can be generated by aggregating cuts generated in previous rounds with constraints of the original constraint matrix. The detailed results in table D.9 on page 172 reveal that some of the instances where the difference is large are lot-sizing instances, which supports this observation. An important question is whether there are higher-rank cMIR inequalities that we can not generate using a path-based cut generator but which are needed to solve some of the MIP problems in our test set. To investigate this we compare two cut generator configurations for which we perform a 1-hour test. In both cut generator configurations we use the flow cover, the flow path, and the cMIR cut generators. In one configuration all cut generators generate cuts out of cuts in the other they do not. The performance profiles for the two cut generator configurations are shown in figure 6.13.

These results show that the use of cuts out of cuts slows down the solver for some instances. On the other hand it allows to solve much more instances within one hour.

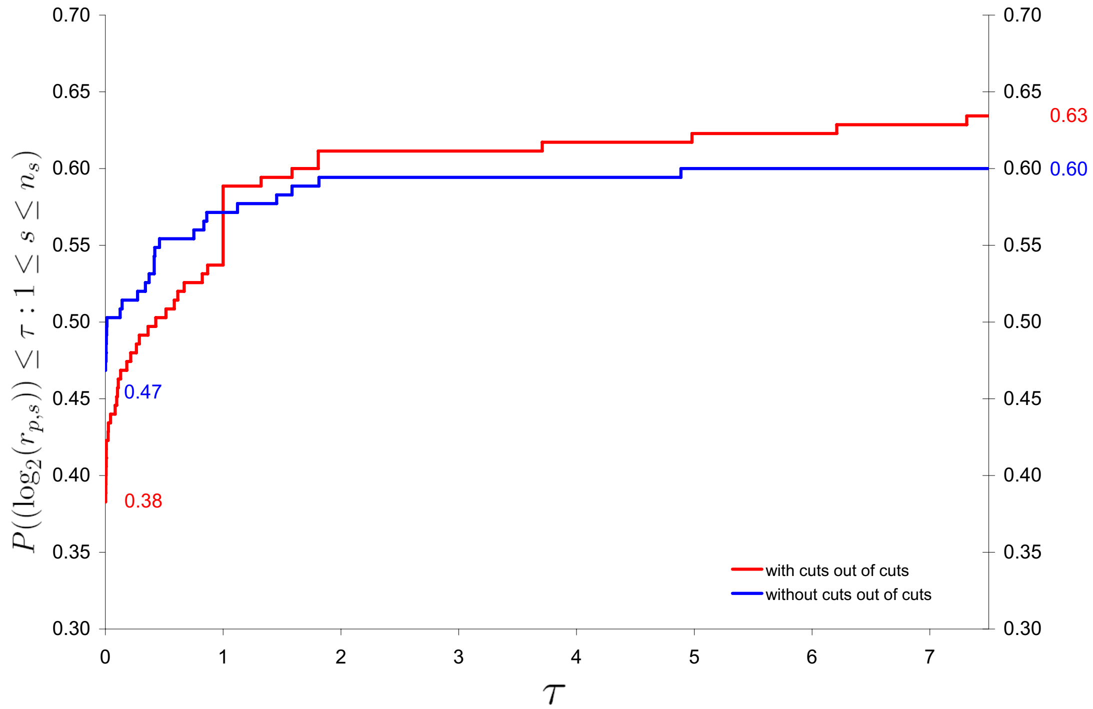

Figure 6.13.: Comparison between two cut generator versions, one that generates cuts out of cuts and one that does not.

Although the version that does not generate cuts out of cuts also used flow path cuts some lot-sizing instances were not solved. In these cases the fact that some path-based cuts can not be generated as flow path cuts but can be generated as higher-rank cMIR cuts might play a role. Overall we consider the possibility to solve more problems more important and thus use cuts out of cuts in our default setting.

In section 5.2 we suggest algorithmic improvements for the cMIR cut separation algorithm that involve changes to the aggregation and bound substitution strategies. The results are called path-based tightest row aggregation and improved bound substitution. We now compare the combination of these two strategies to the traditional strategies suggested by Marchand and Wolsey in [70]. These traditional strategies were also implemented by Gonçalves and Ladanyi in [48]. For this comparison we perform two 10-round tests for different cMIR cut generator versions. The first uses the new path-based tightest row aggregation and the extended bound substitution (version A) and the other the traditional strategies (version B). Figure 6.14 shows the corresponding gap difference diagram.

These results show that for 92% of the instances in our test set the improved version of the cMIR cut generator results in equal or better dual bounds. The detailed results can

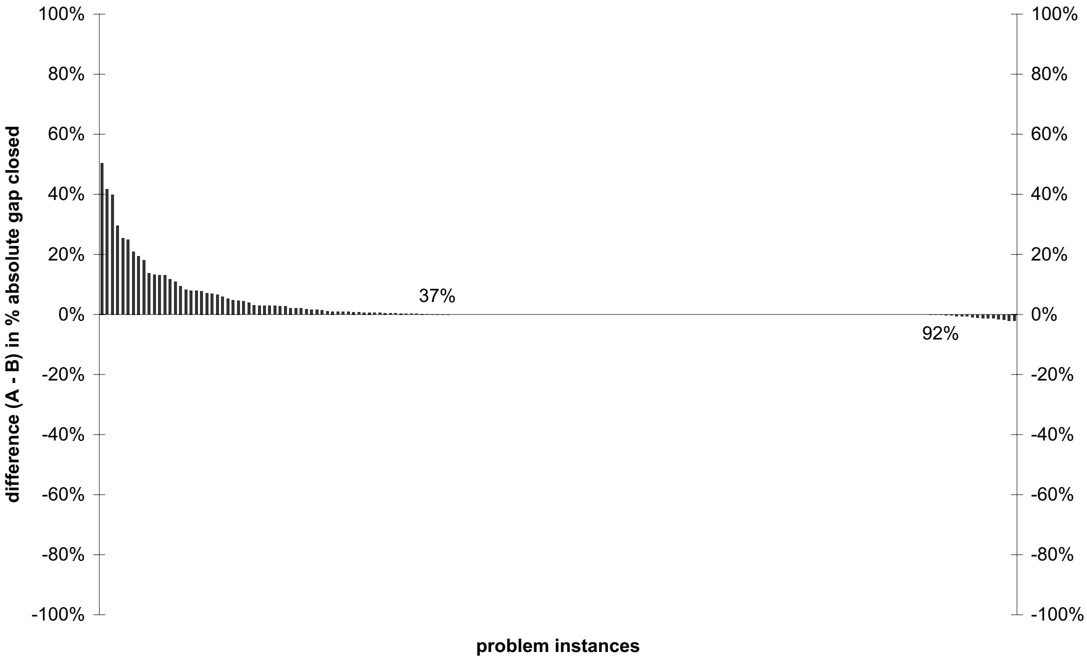

Figure 6.14.: Comparison between a cut generator version using path-based tightest row aggregation and extended bound substitution (A) and version using traditional aggregation and bound substitution (B).

be found in table D.10 on page 173. Only for a few instances the traditional aggregation and bound substitution methods obtain better results.

We now look at the two algorithmic improvements separately. First we inspect the path-based tightest row aggregation. The path-based tightest row aggregation improves the aggregation heuristic by Marchand and Wolsey in [70] in two points. First, as indicated in section 5.2, we change the selection of the aggregation variable to emphasize finding path structures. Second, we specify that in the row selection of the aggregation heuristic the tightest row is used. Marchand and Wolsey did not specify which row to use but Gonçalves and Ladanyi [48] used the first row they found. In figure 6.15 we show a gap difference diagram for a comparison of two 10-round tests for two cMIR cut generator versions. The first version uses the path-based tightest row aggregation (version A) the second uses the traditional aggregation heuristic from [70] and [48], i.e. choosing the variable farthest from its bounds and the first row found (version B). Both versions use the extended bound substitution method.

The diagram shows that the path-based tightest row aggregation contributes significantly to the good results of our improved cMIR cut generator. Table D.11 on page 177 lists the detailed results. Another important aspect of the path-based tightest row aggregation

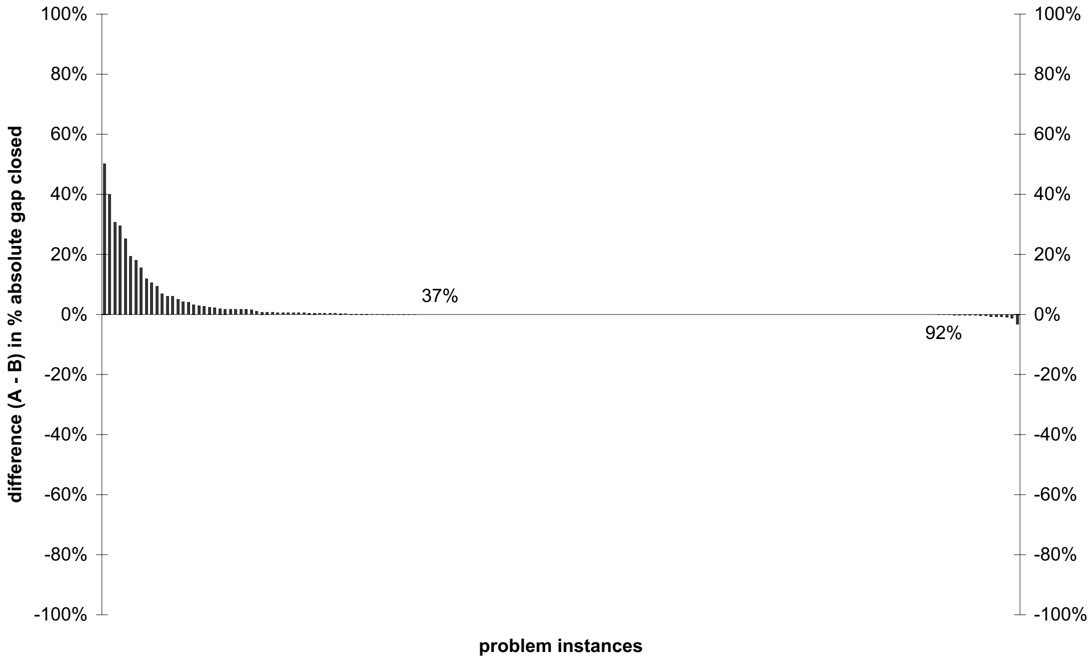

Figure 6.15.: Comparison between a cMIR cut generator version that uses path-based tightest row aggregation (A) and one that uses the traditional aggregation strategy (B).

is that it leads to better results for the path-based cut generators. This is evaluated in section 6.5.

Now we investigate how large the impact of the improved bound substitution method on the performance of our cMIR cut generator is. The largest difference between the traditional and the improved bound substitution is the use of extended bounds. The idea of using extended bounds is to make up for the fact that the path-based tightest row aggregation less frequently incorporates information about complex bound structures. Other differences are the use of variable bounds on integer variables and an improved bound substitution rule. Again we show a gap difference diagram (figure 6.16) for two versions of the cMIR cut generator. The first version (A) uses the improved bound substitution, the second (B) the traditional one. In both cases we use the path-based tightest row aggregation.

These results show that the impact of the improved bound substitution is also significant but it seems to be less influential than the aggregation strategy. We also see that the impact of combining both methods is higher than of the individual methods.

The impact of using binary variable bounds on integer variables in the bound substitution

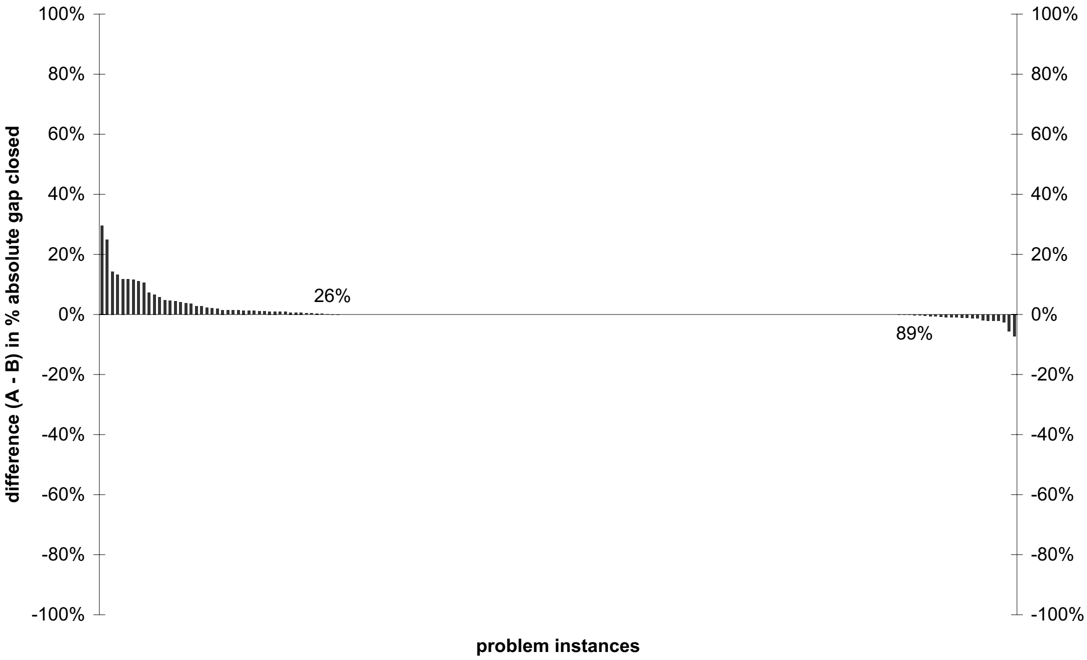

Figure 6.16.: Comparison between a version of the cMIR cut generator that uses the improved bound substitution (A) and one that does not (B).

is shown by the gap difference diagram in figure 6.17. These results are obtained by running two 10-round tests with two cMIR cut generator versions. The first uses binary variable bounds for integer variables (A), the second does not (B).

As observed in a similar experiment for the flow cover cuts using binary variable bounds on integer variables only influences very few instances. See table D.13 on page 179 for details. Note that by using variable bounds on integer variables we eliminate one of the advantages of the flow cover cut generator over the cMIR cut generator.

6.4.2. Comparison to the Previous cMIR Cut Generator

In this section we compare the new cMIR cut generator to the old one. This old cMIR cut generator was implemented by Wesselmann as part of his diploma thesis [95] and includes some of the improvements also used in the new cMIR cut generator. In figure 6.18 we show the gap difference diagram for the comparison of the new cMIR cut generator (A) and the old one (B).

The results show that for almost half of the instances using the new cMIR cut generator results in better dual bounds. For a few instances the old cut generator performs better.

Figure 6.17.: Comparison between a version of the cMIR cut generator that uses binary variable bounds on integer variables (A) and one that does not (B).

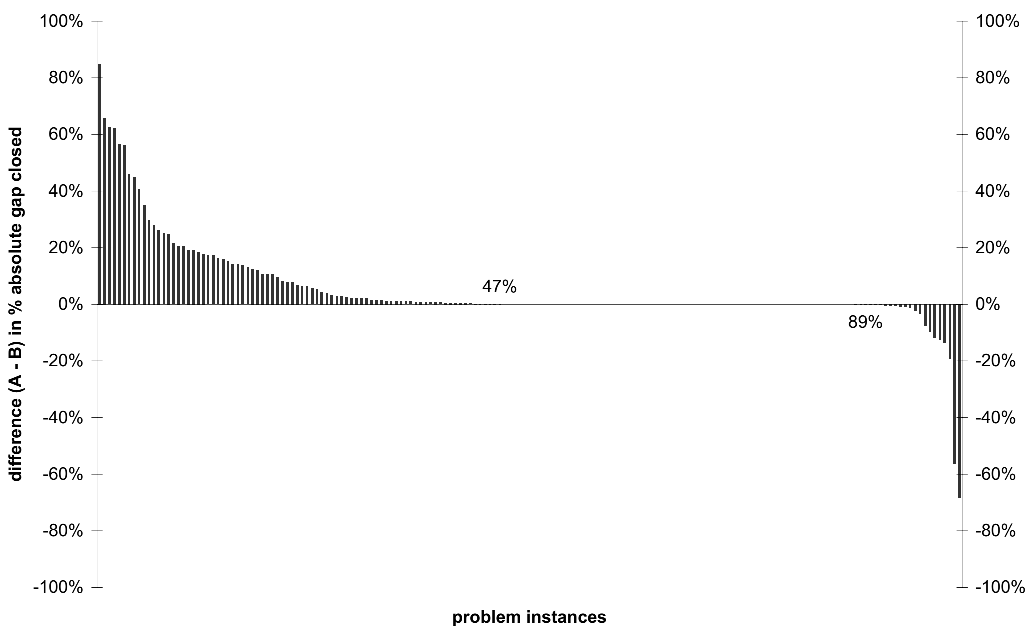

Figure 6.18.: Comparison between the new cMIR cut generator (A) and the old one (B).

A closer inspection of the detailed results, shown in table D.14 on page 175, reveals that the instances for which the negative difference is large are easily solvable or unsolvable for either version and therefore the advantage of the old cMIR cut generator does not influence the overall performance very much. The largest difference between the old and the new cut generator is that the old one did not use mixed integer knapsack sets without continuous variables. Hence among the largest positive differences in the dual bounds obtained are some pure 0-1 instances. Concerning efficiency, the new cMIR cut generator needs much more time for some instances. These instances are mainly very hard ones so that we assume that spending more time on them does not have an impact on the overall performance of the solver. In section 6.6 we compare the overall performance of the solver using the old and the new cut generators to get a picture of the overall improvement of the performance.

6.4.3. Comparison to Published Results

In this section we compare the results of our cMIR cut generator with the results published by Marchand and Wolsey in [70]. As pointed out in section 6.1, comparing cut generators implemented in different solvers gives only limited insights. To make this comparison as fair as possible we compare the results in [70] with our results where we do not generate cuts out of cuts (see table 6.5).

The comparison of the results indicates that for about half of the instances our results are the same or better. For the other half the results by Marchand and Wolsey are better. The only large difference is observed for the instance `khb05250`. When investigating this instance more closely we see that if we deactivate the MOPS LP preprocessing, the dual bound after 10-rounds of cuts is 106825740.22, so even better than the result by Marchand and Wolsey. This is one symptom of the many important differences in the underlying solvers. One such a difference is the MOPS cut pool, which leads to much smaller number of cuts even if the same or a better dual bound is obtained. Overall we claim that it is viable to conclude from these results that our implementation is capable of competing with the original one by Marchand and Wolsey.

6.4.4. Comparison between the Flow Cover and the cMIR Cut Generator

In this section we want to check whether our cMIR cut generator renders our flow cover cut generator obsolete. In section 4.2 we show that the cMIR approach can be used to generate SGFCIs and even LSGFCIs in some cases. In the evaluation of the flow cover

	Marchand and Wolsey 2001		MOPS		
name	cuts	root	cuts gen.	cuts sel.	root
egout	213	561.00	64	14	**566.94**
fixnet6	1788	**3832.00**	436	66	3646.79
modglob	530	**20726103.00**	608	106	20679686.42
pp08a	593	7123.00	914	164	**7161.51**
pp08aCUTS	1071	**7219.00**	1346	117	7189.71
qiu	0	**-931.64**	0	0	**-931.64**
rgn	151	**82.20**	234	60	81.80
set1ch	1328	**51814.00**	2263	402	51762.61
vpm1	468	**20.00**	90	40	**20.00**
vpm2	652	12.69	470	129	**12.93**
gen	42	112313.00	114	37	**112313.36**
khb05250	259	**106857080.00**	164	16	96070104.35
dcmulti	64	184283.00	920	96	**186264.76**
flugpl	1	**1167875.00**	0	0	1167185.73
rentacar	175	**29449924.00**	350	17	29274325.20
misc06	0	12841.00	64	7	**12841.69**
mod011	922	**-5844969.00**	3298	317	-59493500.53
bell3a	13	**873351.00**	48	15	870793.00
arki001	176	7579798.00	1314	155	**7579814.00**
bell5	18	8621775.00	110	20	**8926029.48**
danoint	863	**62.72**	1007	105	62.69
blend2	693	**7.17**	25	8	7.04
pk1	9	**0.00**	0	0	**0.00**
noswot	335	**-43.00**	12	6	**-43.00**
rout	1444	**982.64**	28	15	981.86

Table 6.5.: Results from [70] and results for a version of the cMIR cut generator that does not generate cuts out of cuts. The best dual bound in each row is marked bold, *cuts gen.* are the cuts generated by MOPS and *cuts sel.* are the cuts selected by the cut pool.

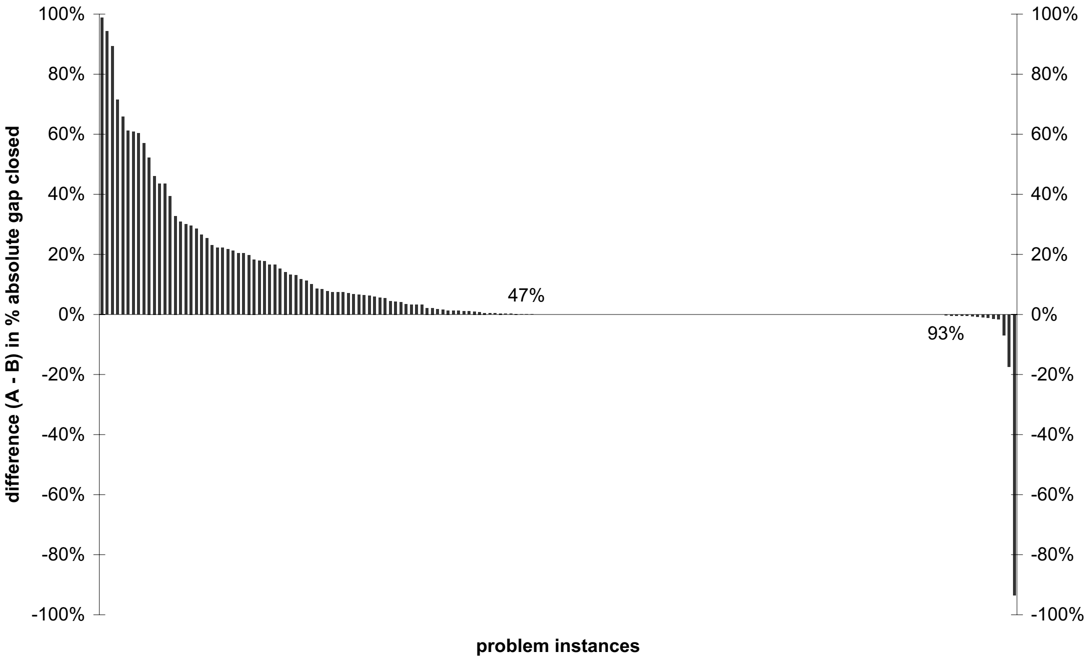

Figure 6.19.: Gap difference diagram for a comparison between the cMIR cut generator (A) and the flow cover cut generator (B). In both cases aggregation is activated.

inequalities in section 6.3 we observed that lifting does not improve the flow cover cut generator very much. So the idea that a cMIR cut generator implemented in certain way can make up for the existence of a flow cover cut generator comes into mind.

In a first experiment we compare the cMIR cut generator and the flow cover cut generator in a 10-round test. To do so, the flow cover cut generator is used with aggregation. The results are shown in the gap difference diagram in figure 6.19. Detailed results can be found in table D.15 on page D.15.

The results show that for almost half of the instances the cMIR cut generator moves the bound more than the flow cover cut generator. The actual differences between the two cut generators are the way in which the set of complemented variables, i.e. the cover, is chosen, the bound substitution rule, and the fact that several values for δ are tried in the cMIR cut generator. That the flow cover cut generator uses lifting is also a difference but it does not seem very important. One instance where the flow cover cut generator is better is `khb05250`. Note that this is not an instances for which we found that lifting improves the dual bound. Our experience is that for some pure integer problems trying different values for δ results in better dual bounds and that in some instances the flow cover cut generator is better because of its superior way of handling the bound substitution step.

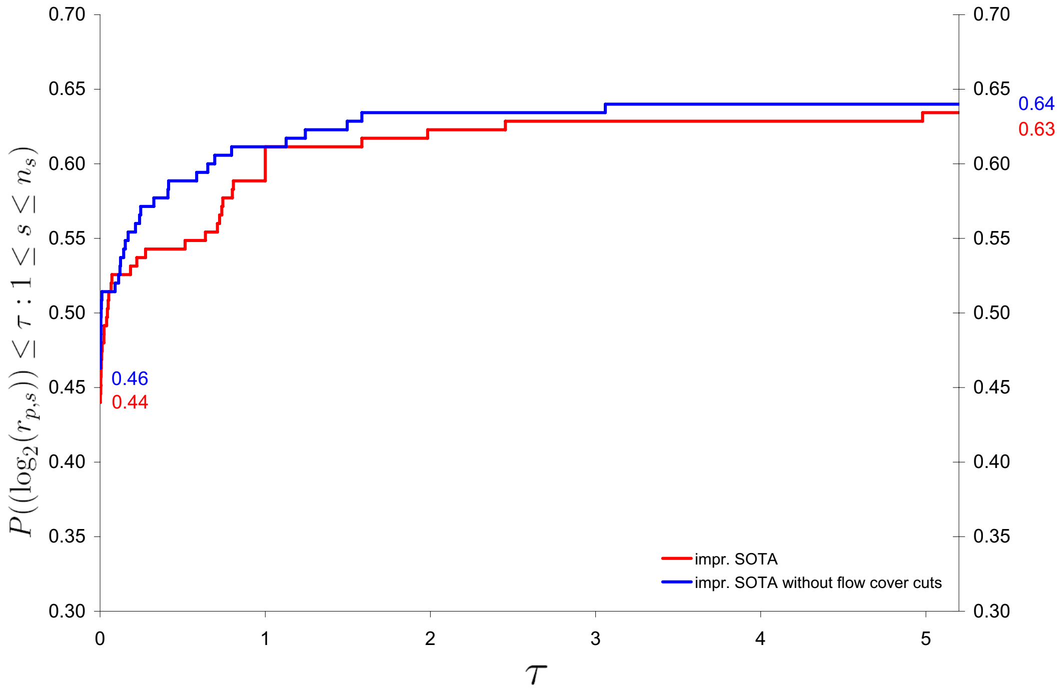

Figure 6.20.: Performance Profiles for the comparison between a state-of-the-art cut configuration with and without flow cover cuts.

We now check how much impact the flow cover cut generator has on the overall performance of the solver by testing the improved SOTA cut configuration with the flow cover cut generator and without it. The results are shown in the performance profiles in figure 6.20.

The performance profiles indicate that the version with the flow cover cut generator shows a slightly worse performance on our test set than the one without it. The only problem instance that is not solved without flow cover cuts but with the other version is `m20-75-4` (see table D.21 on page 187 and table D.24 on page 195). On the other hand the problem instances `bc1` and `prod1` are only solved without flow cover cuts.

From these results we conclude that there are instances where the flow cover cut generator can contribute cuts that are needed to solver certain problem instances. For many other problem instances this is not the case, the performance profiles even suggest that the performance might increase by not using flow cover cuts. We think that using a flow cover cut generator together with a cMIR cut generator adds to the diversity of the generated cuts which in some cases is just what is needed to solve certain problem instances. Maybe it is possible to achieve similar results by using two cMIR cut generators with different configurations. It is also possible that the reason why the flow cover cut generator still

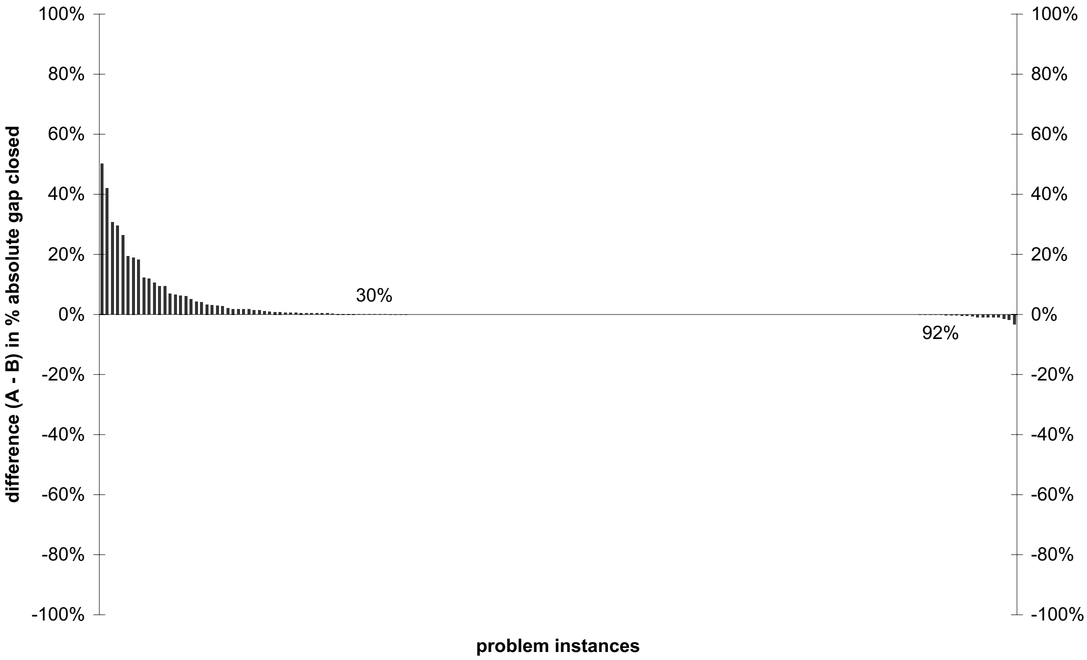

Figure 6.21.: Comparison between the solver version that uses path-based tightest row aggregation (A) and a version that uses the traditional cMIR aggregation (B).

sometimes is needed lies in the fact that the heuristics for bound substitution, cover finding, and cut generation in the cMIR cut generator are not good enough.

6.5. Evaluation of the Path-based Cut Generators

6.5.1. Implementation Details of the Flow Path Cut Generator

We now briefly evaluate the impact of some implementation details on the performance of the flow path cut generator. The first of these implementation details is the aggregation/ path-finding strategy. We evaluate it by comparing the results of 10-round tests for two solver versions where the cMIR and flow path cut generators are activated. In one version we use the path-based tightest row aggregation (version A) and in the other we use the traditional cMIR aggregation strategy (version B). A description of these strategies can be found in section 5.2.5 of this thesis. The results are shown in the gap difference diagram in figure 6.21. Note that we include the cMIR cut generator to also generate cuts from single rows.

Figure 6.22.: Comparison between a solver version that uses extended network inequalities (A) and a version that uses simple network inequalities (B).

The gap difference diagram shows that the dual bounds obtained with the path-based tightest row aggregation are much better than with the traditional aggreagtion strategy. Some of these improvements can be attributed to the improved separation of the cMIR cut generator. Overall the results are not surprising because in the flow path cut separation algorithm the most important step is to identify a path structure in the problem. The other parts of the flow path cut generator just follow simple rules to get the most violated cut out of a path.

A second implementation detail we want to investigate is how important the use of extended network inequalities in contrast to using simple network inequalities is. For this purpose we compare the results of 10-round tests for two solver versions with cMIR and flow path cut generators activated. In the first version (A), we use extended network inequalities, in the second only simple network inequalities (B). The results are shown in the gap difference digram in figure 6.22.

These results show that using extended network inequalities only has an impact on very few instances. Measured in the absolute gap that is closed in the root node even on these instances the impact is not very high. At the same time the additional implementation effort needed to generate extended network inequalities is quite large. Nevertheless we use them in the default version because every little improvement might count in certain situations. One explanation for the small improvement is that the flow path cut separation algorithm only tries to generate an extended network inequality if the corresponding simple network inequality is violated. Thus the extended network inequality is only used to improve upon cuts found, not to generate more cuts.

6.5.2. Comparison of Path-based Cut Generators

In this section we compare the cut generators for path-based cuts. In the tests described in the following we again always use the cMIR cut generator in addition to a path-based cut generator to also generate cuts from single rows. We start with a number of 10-round tests to compare four solver versions that generate path-based cuts with a solver version that does not. The version that does not generate path-based cuts uses the cMIR cut generator that does not generate cuts out of cuts. We compare it to the cMIR cut generator that does generate cuts out of cuts, i.e. we also generate cuts with an cMIR rank larger than one. We call this generating *higher-rank* cMIR cuts. As shown in example 5.8 on page 90 it is possible to generate path mixing cuts as higher-rank cMIR cuts. The corresponding gap difference diagram is also used in evaluating implementation details of the cMIR cut generators and therefore can be found in figure 6.12 on page 125. The sorted results are listed in table D.9 on page 172.

From the digram we see that generating cuts out of cuts results in better dual bounds for about 38% of the problems in our test set. For about 7% of the instances slightly worse dual bounds are obtained. We attribute this to the random behaviour of adding rounds of cuts and to the influence of the MOPS cut pool.

In the next tests we want to show how much adding flow path cuts improves the dual bounds of our problem instances. To concentrate on the actual improvement through flow path cuts we do not generate cuts out of cuts. The results are shown in figure 6.23 and in table D.18 in the appendix.

These results indicate that generating flow path cuts makes a smaller difference than generating higher ranking cMIR cuts. We conclude from this that in the previous test higher ranking cMIR cuts were generated that the flow path cut generator did not generate and probably not even are path-based cuts. From the detailed results in table D.18 (page 182) we see that the problem instances with the largest difference in the dual bound are the `tr*-*` instances and `set1ch`. These instances are lot-sizing problems from LOTSIZELIB [20] and, as expected, the flow path cut generator works very well for them.

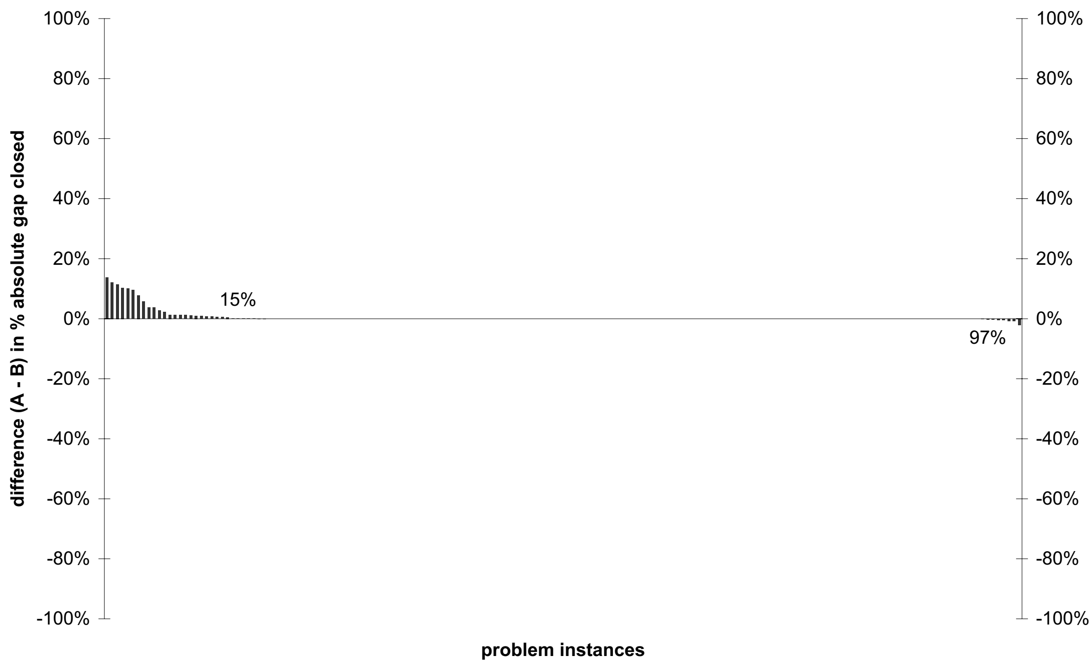

Figure 6.23.: Comparison between a solver version where the flow path cut generator and the cMIR cut generator are used (A) and a version where only the cMIR cut generator is used (B). In both cases no cuts are generated out of cuts.

By comparing these results to the one in table D.9 (page 172) from generating higher-rank cMIR cuts we notice that the dual bounds obtained by using the flow path cut generator are better than the one obtained by using higher-rank cMIR cuts. We assume the reason for this is that the flow path cut generator uses the lot-sizing structure more directly and also is able to generate path-based cuts in earlier rounds than the cMIR cut generator. The cMIR cut generator first needs to generate the cuts with rank one before it can obtain a path-based cut. When using the flow path cut generator the path-based cuts are generated in addition to cMIR cuts and not instead of them.

We now perform the same test for the uPMC generator. Again we do not generate cuts out of cuts. The results are shown in figure 6.24 and table D.19.

Except for two outliers the results look similar to the results for the flow path cut generator. As the uPMC generator is designed to imitate the flow path cut generator this is not surprising. From the detailed results in table D.19 (page 183) we can see that the outliers are `liu`, with a much better dual bound, and `clorox`, where the dual bound is much worse. Both instances are not very interesting because `liu` is not solved by MOPS within one hour regardless of the dual bound obtained and `clorox` is solved within seconds by all solver versions we tested. Another observation from the sorted table is that

Figure 6.24.: Comparison between a solver version where the uPMC generator and the cMIR cut generator are used (A) and a version where only the cMIR cut generator is used (B). In both cases no cuts are generated out of cuts.

the lot-sizing problems, where the flow path cut generator improved the dual bounds significantly, are also among the instances where the uPMC generator improves the bounds most. Comparing the detailed results we see that the dual bounds produced using the flow path cut generator are slightly better. We assume that the main advantage of the flow path cut generator is that it does the decision which variables to put into the set C^+ in the best possible way. In the uPMC generator the equivalent to this step is done heuristically in the bound substitution procedure.

Finally, we compare a solver version with the cMIR cut and cPMC generators with a version using just the cMIR cut generator. Again we do not generate cuts out of cuts. The results are shown in figure 6.25 and details are shown in table D.20.

The results indicate that using the cPMC generator improves the dual bounds of more problem instances than the other path-based cut generators but still less than generating higher rank cMIR cuts. Again, as expected, the lot-sizing instances are those with the largest increase in the dual bound. The dual bounds for these instances are most of the time better than the ones obtained using the uPMC generator but typically not better than the ones from using the flow path cut generator. This is quite surprising because the cPMC cut generator can generate path mixing cuts that are facets of the constant

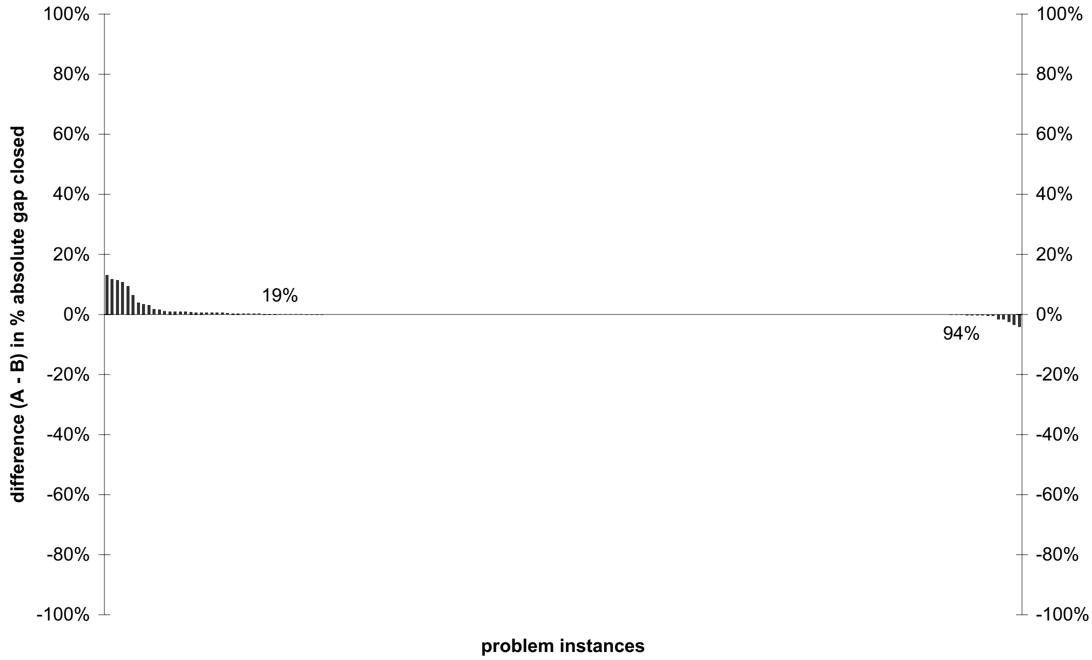

Figure 6.25.: Comparison between a solver version where the cPMC generator and the cMIR cut generator are used (A) and a version where only the cMIR cut generator is used (B). In both cases no cuts are generated out of cuts.

capacity lot-sizing problem and can not be obtained using a flow path cut generator. The same cuts can be generated as higher-rank cMIR cuts. It seems as if these additional cuts are not very successful for the instances in our testset. It might also be that they are not necessary for practically solving MIP problems.

To confirm the results we obtained from the 10-round tests we compare the four different cut generators that can generate path-based cuts in a 1-hour test. This time we generate cuts out of cuts in all cut generators to get a more realistic comparison of the improvements obtained by adding path-based cut generators. The performance profiles for these tests are shown in figure 6.26. The detailed results for the four solver versions can be found in table D.24, table D.25, table D.26 and table D.27.

The performance profiles show that surprisingly the cMIR cut generator performs very good in this comparison. It solves about 43% of the instances fastest and the 64% of the instances within the time limit. The only other solver version that solves as many instances is the one using the flow path cut generator. The performance profile of the flow path cut generator is very close to the one of the cMIR cut generator and sometimes better. The versions with the uPMC generator or the cPMC generator are not as successful as expected. Using the uPMC generator, MOPS solves as many instances

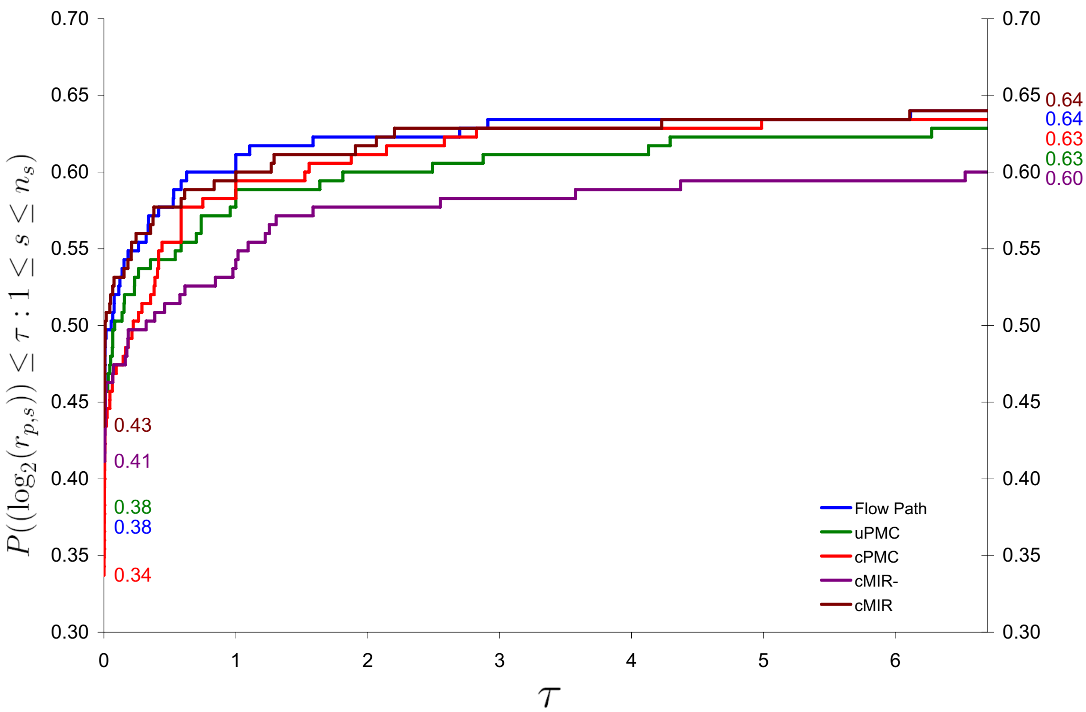

Figure 6.26.: Performance profiles for the four cut generators that generate path-based cuts and the cMIR cut generator that does not generate cuts out of cuts (cMIR-). In all other configurations cuts are generated out of cuts.

fastest as the flow path cut generator but does not solve as many instances within the time limit. Using the cPMC generator, it solves even less instances fastest but only a few less within the time limit. We include a performance profile for a version that uses just a cMIR cut generator without generating cuts out of cuts (MIR-) as a reference. This solver version solves significantly less problems than the others. We investigate this further in the next section.

We assume that one reason why the path mixing cut generators do not perform as well as the flow path cut generator is the earlier mentioned dependence of our new cut generators on the heuristic bound substitution step. Another reason might be that the combination of the cMIR cut generator that generates cuts out of cuts and the flow path cut generator results in more diverse cuts than using the cMIR cut generator with a path mixing cut generator. In the latter case both cut generators us the same reversed mixed integer knapsack sets whereas in the first case more different cuts might be found. Future improvements of the bound substitution process or adjusting implementation details might lead to versions of the path mixing cut generators that are more successful than the ones described here.

The cMIR cut generator performs very well alone but looses some of its power if combined with one of the path mixing cut generators. We assume that the cMIR cut generator only works well if it is not disturbed by another cut generator. To generate path-based cuts with a cMIR cut generator in a later round the exactly right cuts have to be added in earlier rounds. Thus this process is not very reliable but it works fast and is very successful.

6.5.3. Evaluation of the Need for a Path-based Cut Generator

In this section we investigate whether our best path-based cut generator, the flow path cut generator, improves the results of an MIP solver. To do so we run 1-hour tests for two solver versions. The first solver version uses the improved state-of-the-art (SOTA) setting, i.e. flow cover, cMIR, and flow path cuts with a path-based tightest row aggregation and improved bound substitution. The second solver version is identical except that the flow path cut generator is deactivated. The results are shown in figure 6.27, table D.21, and table D.29.

The results of this comparison show that the version with the flow path cut generator performs clearly better than the one without. The one without solves some instances faster but not very many and not very much. The version with the flow path cuts solves

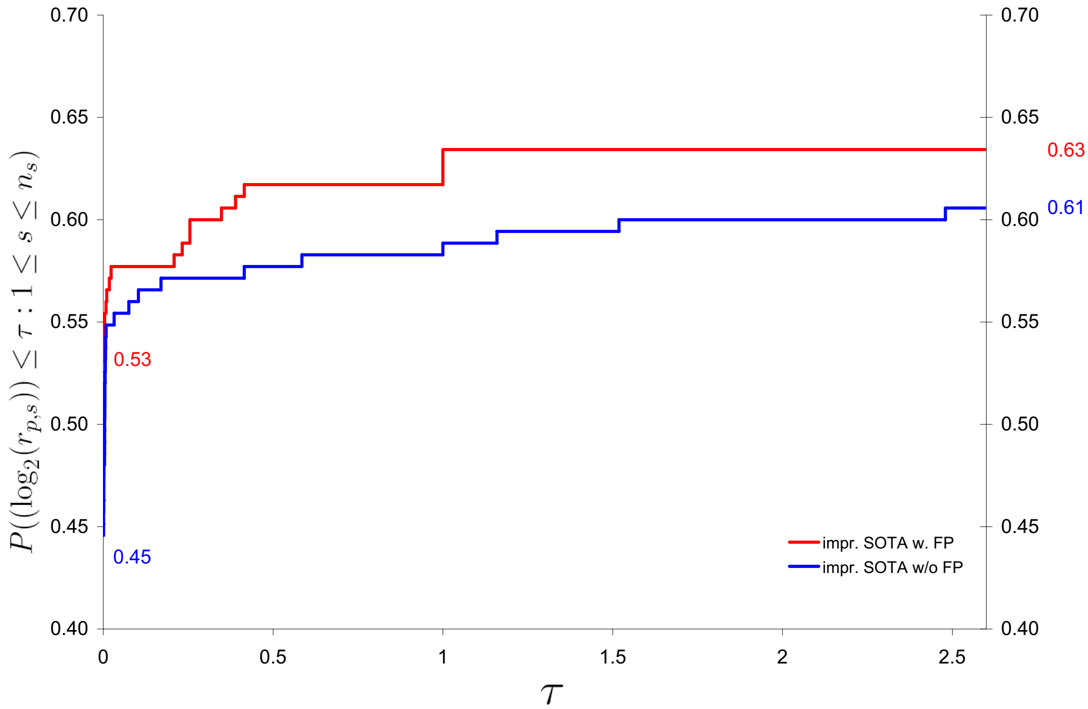

Figure 6.27.: Performance profiles for the improved SOTA solver version with and without flow path cuts.

significantly more instances within the time limit. Looking into table D.21 on page 187 reveals that these are mainly the lot-sizing instances already mentioned in the previous experiments. It seems as if including the flow cover cut generator disturbs the cMIR cut generator in a way that it can not reliably generate path-based cuts anymore.

6.6. Comparison of Cut Configurations

In this section we compare four different configurations of row relaxation-based cut generators to show the improvement through our new implementation. A second purpose of these experiments is to find out which of the configurations is a good default configuration. Two of the configurations tested are the SOTA and the improved SOTA configuration described in section 6.1. The third configuration is the old configuration of the MOPS solver (called OLD). This configuration uses the old cMIR and flow cover generators. It does not use a flow path cut generator. The fourth configuration is the improved SOTA configuration were the flow cover cuts have been deactivated. For these four settings we first perform 1-hour tests with the 4LIB test set. The results are shown in the performance profiles displayed in figure 6.28.

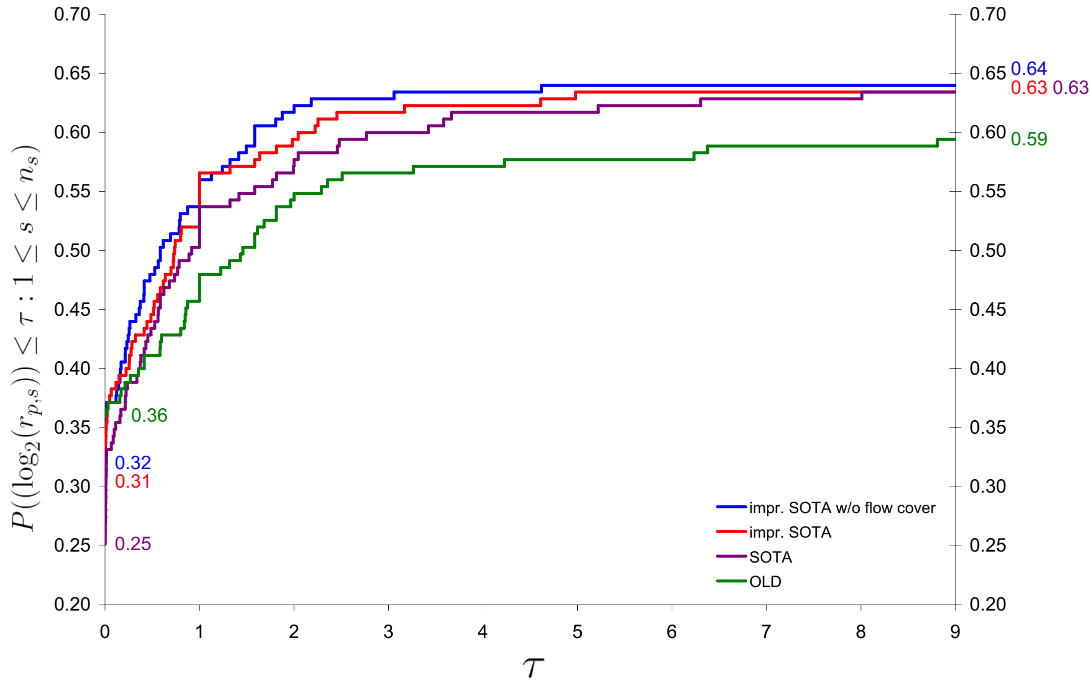

Figure 6.28.: Performance profiles for the four different cut configurations.

The performance profiles indicate that our new implementations perform significantly better than the OLD configuration. The only point were the OLD setting is slightly better is the number of instances that are solved fastest. But for theses instances the difference is not large enough to move the performance profile on top of the others. Furthermore we see again that the improved SOTA configuration without flow cover cuts performs slightly better than the one with them. We discussed this already in section 6.4.

The difference between the SOTA and the improved SOTA configuration is also noticeable. Both configurations solve the same number of instances but the improved version does so slightly faster. We conclude from this that our algorithmic improvements, i.e. path-based tightest row aggregation and improved bound substitution, influence the performance of both the cMIR and flow path cut generator positively. For more detailed results we refer to the tables in the appendix (impr. SOTA: page 187, SOTA: page 206, OLD: page 211, impr. SOTA w/o flow cover: page 195).

We would like to point out that all three configurations that use the new implementations successfully solve the instance `tr12-30` within one hour. This instance is part of the MIPLIB2003 test set and typical state-of-the-art MIP solvers struggle to solve this instance to optimality within a few hours. We assume that out improved implementations of the flow path and the cMIR cut generators lead to this result.

name	impr. SOTA	SOTA	OLD	impr. SOTA w/o FC
mopsMIB001	10.85%	**6.54%**	6.86%	10.85%
mopsMIB002	-	-	-	-
mopsMIB003	26.66%	26.74%	**21.76%**	26.84%
mopsMIB004	-	-	**26.87%**	-
mopsMIB005	149.99%	151.84%	155.09%	**149.98%**
mopsMIB006	2.21%	2.12%	**0.49%**	2.21%
mopsMIB007	**0.01%**	**0.01%**	*	**0.01%**
mopsMIB008	59.61	**53.24**	-	57.57
mopsMIB009	-	-	-	-
mopsMIB010	-	-	-	-
mopsMIB011	-	-	-	-
mopsMIB012	-	-	-	-
mopsMIB013	**0.21%**	0.22%	0.23%	**0.21%**
mopsMIB014	0.02%	0.02%	**0.01%**	0.02%
mopsMIB015	8.75%	**8.27%**	9.21%	8.48%
mopsMIB016	-	-	42.7%	**31.77%**
mopsMIP001	0.90	**0.63**	0.94	0.90
mopsMIP002	**15.5%**	15.56%	18.41%	**15.5%**
mopsMIP003	1.28	**0.49**	0.79	1.30
mopsMIP004	**9.76%**	**9.76%**	9.01%	9.34%
mopsMIP005	37.99	41.52	35.75	**31.17**
mopsMIP006	0.52	0.61	1.62	**0.48**
mopsMIP007	10.75%	34.38%	**9.99%**	10.75%
mopsMIP008	67.02%	**66.04%**	68.73%	67.02%
mopsMIP009	84.9%	**64.84%**	76.09%	76.09%

Table 6.6.: Results for the MOPSLOB test set, time in minutes or duality gap after 1-hour. The * indicates that MOPS reported a wrong solution.

Now we also investigate the results for the MOPSLIB test set. The same configurations as above are tested. We show the results of the 1-hour tests in table 6.6 instead of showing performance profiles because only a few of the instances are solved within the time limit. Values in % are the duality gap when the solver terminated. A '-' indicates that no duality gap could be report as no primal bound was found. The best values in each row are marked bold.

The results for this test set with many very large and difficult instances show that there are instances for which the conclusions obtained through the previous experiments do not necessarily hold. By comparing the values in the table we can not clearly say which configuration works best. The duality gap results depend very much on the primal heuristics of MOPS and these results indicate that they need to be improved, especially for large and difficult instances. The detailed results can be found in the appendix (impr. SOTA: page 212, SOTA: page 213, OLD: page 213, impr. SOTA w/o flow cover: page 212).

The problem instance `mopsMIB008` is solved by all configurations except OLD. The problem with this instance is that it contains many rows with hundreds of elements for which the cMIR cut generation can take a very long time. Only because our new implementation limits the maximal number of elements in a row, this instance can be solved with MOPS within one hour.

Overall we see that finding a good default configuration for an MIP solver is very hard and depends on the test set. Nevertheless does this also show that our improved SOTA configuration with and without flow cover cuts can compete with other configurations and does not results in generally worse solution behaviour.

7. Conclusions and Outlook

7.1. Conclusions

In this thesis we discuss several aspects of five separation algorithms for cutting planes based on mixed integer row relaxations. The most important aspect is the implementation of separation algorithms in the context of an MIP solver. Besides outlining the general implementation of the five separation algorithms we also point out implementation details that have not been mentioned in previous publications. Furthermore these implementations, i.e. the five cut generators, are evaluated to identify the importance of some implementation details and to compare them to each other. This leads us to insights about how these separation algorithms should be implemented and which type of cut generators are actually needed in an MIP solver.

Concerning the flow cover cut separation algorithm we describe its implementation in more detail than any other publication. Especially our results on how to exactly formulate the flow cover finding knapsack problem are vital for implementing a competitive flow cover cut generator. By implementing it in a framework with the cMIR cut generator we are able to evaluate its importance in comparison to this other very important row relaxation based cut generator.

A result of this evaluation is that in most cases the flow cover cut generator is not needed as long as a cMIR cut generator is used. This cMIR cut generator needs to be implemented in a certain way in order to be a good substitute. For a few instances (only one in our test set) the flow cover cut generator helps to solve otherwise not solvable instances. Reasons for this might be that the flow cover cut generator handles the bound substitution step more carefully or that it sometimes generates different cuts than the cMIR cut generator and thus increases the overall diversity of the cuts. On the other hand using it together with the cMIR cut generator can also lead to not solving some instances. The reasons for this can be manyfold and include some randomness.

Concerning the cMIR cut generator the result of this thesis is a detailed description of its implementation as well as a number of algorithmic improvements. One algorithmic improvement is the way in which the aggregation step in the cMIR cut separation algorithm is done. We introduce the path-based tightest row aggregation strategy that focusses on finding path structures in MIP problems. Using this aggregation strategy improves the performance of the cMIR cut generator and allows us to use the same strategy for a path-based cut generator. A drawback of this strategy is that it mostly does not consider extended bound structures for aggregation. Therefore we also introduce an improved bound substitution step for the cMIR cut separation algorithm. This improved bound substitution step not only considers extended bound structures, it also makes use of binary variable bounds on integer variables and refines the decision whether to substitute the upper or the lower bound. An evaluation of these algorithmic improvements reveals that they largely improve the performance of the cMIR cut generator on our test set.

Concerning path-based separation algorithms our first result is the description of a cut generator for flow path cuts. Again this description is more detailed than any in the current literature and mentions implementation aspects not considered anywhere else. We also show that it is possible to use the same aggregation/path-finding method for both a cMIR cut generator and a flow path cut generator.

An important result of this thesis is the definition of the path mixing inequalities and the description of two new path-based separation algorithms. The path mixing inequalities are important because they are a superset of the flow path inequalities. Cut generators for flow path cuts are very limited in their use and can not be extended easily. Path mixing inequalities are based on the more general mixing procedure. As we show, this can be used to suggest an algorithm and implement a cut generator, called the uPMC generator, that uses parts of the cMIR cut generator to generate flow path cuts implicitly. A further strength of this cut generator is that it can be implemented easily and efficiently.

The implementation of a second separation algorithm described for path mixing cuts results in the cPMC generator. It is aimed at generating path-based cuts that go beyond flow path cuts, as it is designed to generate facet-defining cuts for the constant capacity lot-sizing problem. It is less easy to implement but also uses parts of the cMIR cut generator and potentially generates cuts not obtainable with a flow path cut generator.

Another result of this thesis concerning path-based cuts is that, if implemented in a certain way, a cMIR cut generator can generate path mixing cuts implicitly. This only works if it generates cuts out of cuts, i.e. if it combines cuts generated in a previous round and original rows of the constraint matrix in the aggregation step. Although the results of this method are subject to random influences we show that it works very well.

The evaluation chapter of this thesis also includes experiments with configurations of cut generators. We show that configurations using our newly implemented cut generators outperform the existing ones of the MOPS MIP solver. Furthermore we show that our algorithmic improvements to the cMIR cut generator result in an improved overall performance of the state-of-the-art configuration for row relaxation-based cut generators. These results also include solving the very difficult MIP problem instance `tr12-30` that with our new cut generators can be solved in about 40 minutes. Other commercial MIP solvers usually do not solve this instance to optimality within one hour. Our work on generating cuts for paths also improves the solution times for other lot-sizing based MIP problems.

The computational experiments in this thesis are conducted on a large set of problem instances from publically available problem libraries. By not leaving out any problem instance we obtained results for a wide spectrum of problems. We evaluate the results using well-established and newly developed methods, i.e. performance profiles and gap difference diagrams. These experiments and their evaluation also illustrate the difficulties one has to face when evaluating the performance of cut generators and MIP solvers. Many components of an MIP solver interfere with each other and small changes in one part of the solver can influence other parts of the solver very much. Only extensive testing and clearly defined performance measures can lead to usable information. From the results for the MOPSLIB test set we see that a different set of problem instances implies new challenges for solvers and their components.

7.2. Outlook

The field of computational mixed integer programming holds a vast number of challenges. Parts of this thesis are merely a starting point for future research. In this section we want to point out possible research opportunities arising from our work and from current topics in MIP.

Concerning the cMIR cut generator it is apparent that the current bound substitution procedures are not leading to the best possible results. The goal of future research could be to use optimization methods to find a way of generating a mixed integer knapsack set such that it leads to a violated cut. One way of doing this is to formulate the non-linear separation problem for MIR cuts as an MIP using linearization techniques, as described in [38]. As this is currently not efficient, a smart way of solving this problem, or only parts of it, has to be found. Another way of improving upon the cMIR cut generator is to use its principles with other families of valid inequalities such as the mingling [12] or two-step MIR inequalities [36]. First steps in this direction are already done but it is not clear whether this approach really leads to better computational results.

The path mixing inequalities and separation algorithms proposed in this thesis are meant as a starting point for future developments. In direct comparison to the well tested and refined flow path cut generator they currently are not competitive. Nevertheless they have some nice properties that future research can build upon. Their current weakness is the bound substitution process that, although improved by the new method in this thesis, still is not good enough to outperform the flow path cut generator. A different way of implementing a path mixing cut separation algorithm may overcome this drawback. Alternatively, research on the cMIR cut generator could lead to a new bound substitution method that would also improve the path mixing cut generators. The closeness to the cMIR cut generator might also be used to implement a cut generator that generates both cMIR and path mixing cuts directly.

In our opinion, the general process of generating cuts in an MIP solver needs more attention from researchers. As we see in some of our experiments, other parts of the solver such as LP preprocessing have an influence on how cut generators perform. There also are non trivial interactions when several cut generators are used together. Finally, we see in our experiments that a cut pool can have a large influence on the performance of an MIP solver. All these topics need intensive computational studying in order to understand better how MIP solvers can be improved.

Other important topics in MIP solver development also demand attention. In mixed integer programming, one topic that is discussed in recent publications is how to handle symmetries in MIP problems. Applying the results from this research into state-of-the-art MIP solvers will probably lead to large performance improvements. Another research direction is called *MIPing* (see [43] for an overview). MIPing means modeling and solving problems occurring in an MIP solver, such as finding feasible solutions or cut generation, as MIP problems. Currently, this approach is successfully used for some components of the solver but is not efficient for others. Future developments might change this. An area that needs more attention are semi-continuous and semi-integer variables. When these modeling methods become more widely used, MIP solvers might have to deal with them in a more efficient way and adjust many of their components to them.

Since quite a while publications in MIP discuss paralellization of MIP solvers. Now that multi-core CPUs are the standard for desktop computers, this topic becomes increasingly interesting. A problem with parallelization is its non-deterministic behaviour that users of MIP solvers do not expect. Therefore the leading solver companies provide parallel MIP solvers with a deterministic behaviour. Research in this area is also likely to have a huge influence on the performance of MIP solvers in the future.

A. Notation

Mathematical Notation

MIP	Mixed integer programming
LP	Linear programming
MIR	Mixed integer rounding
cMIR	complemented mixed integer rounding
uPMC	uncapacitated path mixing cut
cPMC	capacitated path mixing cut
$\mathbb{R}^n$	the set of n-dimensional real numbers
$\mathbb{R}^n_+$	the set of n-dimensional real numbers ≥ 0
$\mathbb{R}^n_{>0}$	the set of n-dimensional real numbers > 0
$\mathbb{Z}^n$	the set of n-dimensional integer numbers
$\mathbb{Z}^n_+$	the set of n-dimensional integer numbers ≥ 0
x^*_j	the current LP solution of a continuous variable j
y^*_j	the current LP solution of a 0–1 or general integer variable j
z^*_j	the current LP solution of a general integer variable j
$(a)^+$	$\max\{0, a\}$
$\lvert a \rvert$	absolute value of a, i.e. a if $a \geq 0$ and $-a$ if $a < 0$
$\lfloor a \rfloor$	the largest integer number $\leq a$
$\lceil a \rceil$	the smallest integer number $\geq a$
$\emptyset$	the empty set, is also used to indicate empty data structures
$A = (B, C)$	If B and C are sets, (B, C) is a partition of the set A, i.e. $B \cup C = A$, $B \cap C = \emptyset$ If B and C are matrices, then A is the matrix where C is placed next to B

Pseudo-code Notation

row, $aggRow$, etc.	data structure to store rows of the form: $\sum_{j\in N} a_j x_j + \sum_{j\in P} g_j y_j = b, \quad x \in \mathbb{R}^{\|N\|}, \quad y \in \mathbb{Z}_+^{\|P\|}$
$refSta$	data structure to store the reformulation status
$usableRows$	data structure to store a list of usable rows
$path$	data structure to store an ordered list of rows
cut, $bestCut$, etc.	data structure to store a cut
mik	data structure to store a reversed mixed integer knapsack set: $\sum_{j\in I} g_j y_j + s \geq b, \quad y_j \leq u_j$ for $j \in I, \quad y \in \mathbb{Z}_+^{\|I\|}, \quad s \in \mathbb{R}_+^1$

B. Example Configuration Files

```
xmxdsk = 5000 xoutsl = 0 xmxmin = 60 xmxmic = 500
xstart = 3
xmxnod = 1
xheutp = 0
xgomct = 0
xcovct = 0
xclict = 1
ximpli = 1
xmxagg = 6
xaggst = 4
xmicuc = 1
xextbs = 1
xnwmic = 1
xflwct = 0
xmirct = 1
xflwpa = 0
xmingc = 0
xmixct = 0
xmxpsu = 10
```

Figure B.1.: Configuration file for a 10-round test with the cMIR cut generator.

```
xmxdsk = 5000 xoutsl = 0 xmxmin = 60 xmxmic = 500
xmxagg = 6
xaggst = 4
xmicuc = 1
xextbs = 1
xnwmic = 1
xflwct = 1
xmirct = 1
xflwpa = 2
xmingc = 0
xmixct = 0
```

Figure B.2.: Configuration file for a 1-hour test with the improved SOTA configuration.

C. Test Sets

NAME	CON	INT	BIN	M	NZ	IP [1]		TYP	S [2]
10teams	225	0	1800	230	11437	924.00		MIB	1 3 4
30_05_100	1	0	10771	12050	45879	9.00		BIN	4
30_95_100	1	0	10975	12526	46657	3.00		BIN	4
30_95_98	1	0	10989	12471	46365	12.00		BIN	4
a1c1s1	3456	0	192	3312	12917	11503.44		MIB	3
acc0	0	0	1620	1737	7304	0.00		BIN	4
acc1	0	0	1620	2286	13047	0.00		BIN	4
acc2	0	0	1620	2520	15478	0.00		BIN	4
acc3	0	0	1620	3249	16766	0.00		BIN	4
acc4	0	0	1620	3285	16955	0.00		BIN	4
acc5	0	0	1339	3052	15792	0.00		BIN	4
aflow30a	421	0	421	479	13211	1158.00		MIB	3
aflow40b	1364	0	1364	1442	35695	1168.00		MIB	3
air03	0	0	10757	124	91210	340160.00		BIN	1
air04	0	0	8904	823	48359	56137.00		BIN	1 3 4
air05	0	0	7195	426	36460	26374.00		BIN	1 3 4
arki001	850	96	442	1048	19107	7580813.05		MIP	1 3
atlanta-ip	1965	106	46667	21732	184445	90.01		MIP	3
b4-10	700	0	480	1509	6017	14050.84		MIB	2
b4-10b	700	0	480	2871	13321	14050.84		MIB	2
b4-12	840	0	576	1823	7397	16103.88		MIB	2
b4-12b	840	0	576	3857	22806	16103.88		MIB	2
b4-20b	1560	0	1080	9707	104534	23358.21	*	MIB	2
BASF6-10	6350	0	1300	3610	21774	21267.57		MIB	2
BASF6-5	3175	0	650	1805	10784	12071.58		MIB	2
bc1	1499	0	252	1913	188174	3.34		MIB	4
bell3a	62	32	39	123	191	878430.32		MIP	1
bell5	46	28	30	91	348	8966406.49		MIP	1
bienst1	477	0	28	576	3384	46.75		MIB	4
bienst2	470	0	35	576	3729	54.60		MIB	4
binkar10_1	2128	0	170	1026	4798	6742.20		MIB	4
blend2	89	25	239	274	2581	7.60		MIP	1
cap6000	0	0	6000	2176	15556	-2451377.00		BIN	1 3 4
ches1	84	32	114	212	702	74.34		MIP	2
ches2	287	81	800	516	2762	-2889.85	*	MIP	2

continued on the next page

NAME	CON	INT	BIN	M	NZ	IP [1]		TYP	S [2]
ches3	330	33	363	234	1836	-1303896.92		MIP	2
ches4	66	22	242	145	667	-647402.75		MIP	2
ches5	432	54	486	366	2376	-7342.82		MIP	2
clorox	345	0	75	737	5121	21217.81		MIB	2
Con-12	360	0	120	554	2402	7593.07		MIB	2
con-24	720	0	240	1118	4516	25804.96	*	MIB	2
dano3_3	13804	0	69	3202	112192	576.34		MIB	4
dano3_4	13781	0	92	3202	115184	576.44		MIB	4
dano3_5	13758	0	115	3202	115207	576.92		MIB	4
dano3mip	13321	0	552	3202	120299	688.90	*	MIB	1 3
danoint	465	0	56	664	4527	65.67		MIB	1 3
dcmulti	473	0	75	290	2754	188182.00		MIB	1
disctom	0	0	10000	399	29674	-5000.00		BIN	3
dlsp	543	0	177	954	2499	613.00		MIB	2
ds	0	0	67732	656	1024059	180.00	*	BIN	3
dsbmip	1694	0	192	1854	7668	-305.20		MIB	1
egout	86	0	55	98	165	568.10		MIB	1
enigma	0	0	100	21	373	0.00		BIN	1
fast0507	0	0	63009	507	416441	174.00		BIN	1 3
fiber	44	0	1254	363	2822	405935.18		MIB	1 3
fixnet6	500	0	378	478	5042	3983.00		MIB	1 3
flugpl	7	11	0	18	40	1201500.00		MIP	1
gen	720	6	144	780	1064	112313.36		MIP	1
gesa2	816	168	240	1392	5319	25779856.37		MIP	1 3
gesa2_o	504	336	384	1248	5637	25779856.37		MIP	1 3
gesa3	768	168	216	1368	3623	27991042.65		MIP	1
gesa3_o	480	336	336	1224	3524	27991042.65		MIP	1
glass4	20	0	302	396	2884	1200012600.00		MIB	3
gt2	0	164	24	29	604	21166.00		INT	1
harp2	0	0	2993	112	6870	-73899798.00		BIN	1 3
khb05250	1326	0	24	101	6376	106940226.00		MIB	1
l152lav	0	0	1989	97	9240	4722.00		BIN	1
liu	67	0	1089	2178	11627	1104.00	*	MIB	3
lrn	4798	0	2455	8701	34460	44479487.02		MIB	4
lseu	0	0	89	28	916	1120.00		BIN	1
m20-75-1	20	425	75	445	6839	-50322.00		MIP	4
m20-75-2	20	425	75	445	8821	-50322.00		MIP	4
m20-75-3	20	425	75	445	9331	-51158.00		MIP	4
m20-75-4	20	425	75	445	7454	-52752.00		MIP	4
m20-75-5	20	425	75	445	6659	-51349.00		MIP	4
manna81	0	3303	18	6480	12960	-13164.00		INT	3
markshare1	12	0	50	6	404	1.00		MIB	1 3
markshare1_1	17	0	45	6	425	0.00		MIB	4

continued on the next page

NAME	CON	INT	BIN	M	NZ	IP [1]		TYP	S [2]
markshare2	14	0	60	7	513	1.00		MIB	1 3
markshare2_1	20	0	54	7	612	0.00	*	MIB	4
mas74	1	0	150	13	4702	11801.19		MIB	1 3 4
mas76	1	0	150	12	2595	40005.05		MIB	1 3 4
misc03	1	0	159	96	1824	3360.00		MIB	1
misc06	1696	0	112	820	3731	12850.86		MIB	1
misc07	1	0	259	212	8260	2810.00		MIB	1 3 4
mitre	0	0	10724	2054	50031	115155.00		BIN	1
mkc	2	0	5323	3411	30176	-563.85		MIB	1 3
mod008	0	0	319	6	3462	307.00		BIN	1
mod010	0	0	2655	146	1603	6548.00		BIN	1
mod011	10862	0	96	4480	34050	-54558535.00		MIB	1 3 4
modglob	324	0	98	291	2348	20740508.10		MIB	1 3
momentum1	2825	0	2349	42680	64386	109143.49		MIB	3
momentum2	1923	1	1808	24237	172786	12314.22		MIP	3
momentum3	6933	1	6598	56822	563233	264954.00	*	MIP	3
msc98-ip	853	53	20237	15850	83152	19839497.01		MIP	3
multiA	1440	0	480	972	3482	3774.76		MIB	2
multiB	1440	0	480	972	4129	3964.90		MIB	2
multiC	1440	0	480	972	4105	2083.29		MIB	2
multiD	1440	0	480	972	5583	6089.07	*	MIB	2
multiE	720	0	240	492	3390	2710.59	*	MIB	2
multiF	540	0	180	372	2728	2428.90		MIB	2
mzzv11	0	251	9989	9499	133833	-21718.00		INT	3 4
mzzv42z	0	235	11482	10460	144136	-20540.00		INT	3 4
neos1	0	0	2112	5020	10601	19.00		BIN	4
neos10	0	5	23484	46793	140413	-1135.00		MIP	4
neos11	320	0	900	2706	9506	9.00		MIB	4
neos12	847	0	3136	8317	20045	13.00		MIB	4
neos13	12	0	1815	20852	215701	-95.47		MIB	4
neos2	1061	0	1040	1103	11157	454.86		MIB	4
neos20	198	30	937	2446	5200	-434.00		MIP	4
neos21	1	0	613	1085	12391	7.00		MIB	4
neos22	2786	0	454	5208	12402	779715.00		MIB	4
neos23	245	0	232	1568	3845	137.00		MIB	4
neos3	1387	0	1360	1442	18122	368.84		MIB	4
neos4	5712	0	17172	38577	23530	-48603440751.00		MIB	4
neos5	10	0	53	63	2016	15.00		MIB	4
neos6	446	0	8340	1036	252166	83.00		MIB	4
neos648910	66	0	748	1491	3838	32.00		MIB	4
neos671048	13	880	2695	23762	64054	5001.00		MIP	4
neos7	1102	20	434	1994	5608	721934.00		MIP	4
neos8	0	4	23224	46324	138343	-3719.00		MIP	4

continued on the next page

NAME	CON	INT	BIN	M	NZ	IP [1]		TYP	S [2]
neos9	79309	0	2099	31600	245910	798.00		MIB	4
net12	12512	0	1603	14021	69440	214.00		MIB	3
noswot	28	25	75	182	765	-41.00		MIP	1 3
nsrand-ipx	1	0	6620	735	137144	51200.00		MIB	3
nug08	0	0	1632	912	5936	214.00		BIN	4
nw04	0	0	87482	36	636666	16862.00		BIN	1 3 4
opt1217	1	0	768	64	2976	-16.00		MIB	3
p0033	0	0	33	16	225	3089.00		BIN	1
p0201	0	0	201	133	1930	7615.00		BIN	1
p0282	0	0	282	241	1584	258411.00		BIN	1
p0548	0	0	548	176	2454	8691.00		BIN	1
p2756	0	0	2756	755	9113	3124.00		BIN	1 3
pk1	31	0	55	45	915	11.00		MIB	1 3 4
pp08a	176	0	64	136	1886	7350.00		MIB	1 2 3
pp08aCUTS	176	0	64	246	2214	7350.00		MIB	1 3
prod1	101	0	149	208	4596	-56.00		MIB	4
prod2	101	0	200	211	10490	-62.00		MIB	4
protfold	0	0	1835	2112	21776	-31.00		BIN	3
qap10	0	0	4150	1820	15480	340.00		BIN	4
qiu	792	0	48	1192	3696	-132.87		MIB	1 3 4
qnet1	124	129	1288	503	5977	16029.69		MIP	1
qnet1_o	124	129	1288	456	4643	16029.69		MIP	1
ran10x26	260	0	260	296	5366	4270.00		MIB	4
ran12x21	252	0	252	285	5080	3664.00		MIB	4
ran13x13	169	0	169	195	3248	3252.00		MIB	4
rd-rplusc-21	165	0	457	125899	308457	165395.28		MIB	3
rentacar	9502	0	55	6803	21869	30356760.98		MIB	1
rgn	80	0	100	24	1336	82.20		MIB	1
rgna	100	0	20	84	150	82.20		MIB	2
roll3000	428	492	246	2295	28828	12890.00		MIP	3
rout	241	15	300	291	3998	1077.56		MIP	1 3
set1ch	472	0	240	492	3478	54537.75		MIB	1 2 3
seymour	0	0	1372	4944	33536	423.00		BIN	1 3
seymour1	921	0	451	4944	33243	410.76		MIB	4
sp97ar	0	0	14101	1761	307920	661984000.00	*	BIN	3
stein27	0	0	27	118	398	18.00		BIN	1
stein45	0	0	45	331	1034	30.00		BIN	1
stp3d	0	0	204880	159488	598863	500.74	*	BIN	3
swath	81	0	6724	884	28736	467.41		MIB	3
swath2	4399	0	2406	884	27447	385.20		MIB	4
swath3	4099	0	2706	884	27080	397.76		MIB	4
t1717	0	0	73885	551	325689	201736.00	*	BIN	3
timtab1	226	94	77	171	1995	764772.00		MIP	3

continued on the next page

NAME	CON	INT	BIN	M	NZ	IP [1]		TYP	S [2]
timtab2	381	164	130	294	2708	1132271.00	*	MIP	3
tr12-15	360	0	180	375	2690	74634.00		MIB	2
tr12-30	720	0	360	750	7065	130596.00		MIB	2 3
tr24-15	720	0	360	735	6091	136509.00		MIB	2
tr24-30	1440	0	720	1470	7891	288424.00	*	MIB	2
tr6-15	180	0	90	195	1685	37721.00		MIB	2
tr6-30	360	0	180	390	2747	61746.00	*	MIB	2
vpm1	210	0	168	234	1364	20.00		MIB	1
vpm2	210	0	168	234	1865	13.75		MIB	1 3
vpm2a	210	0	168	234	1547	13.75		MIB	2
vpm5	328	0	512	928	4131	3003.20		MIB	2

Table C.1.: Instances of the 4LIB test set.

[1] a * indicates that this solution is not proven to be in the optimality tolerance of MOPS

[2] Source of the instance, 1: MIPLIB3, 2: LOTSIZELIB, 3: MIPLIB2003, 4: MITTELMANN

NAME	CON	INT	BIN	M	NZ	IP [1]		TYPE
mopsMIB001	45800	0	229	50029	229860	106006.00	*	MIB
mopsMIB002	1922	0	1907	62696	192028	2058282.50	*	MIB
mopsMIB003	968	0	961	22862	37821	2507914.50		MIB
mopsMIB004	1922	0	1914	62753	105846	4815524.00	*	MIB
mopsMIB005	1467392	0	13	225347	1175596	748880.61		MIB
mopsMIB006	876801	0	2115	441143	1245281	4716275376.94	*	MIB
mopsMIB007	140267	0	4117	66863	413621	71713031.88		MIB
mopsMIB008	1342207	0	456764	2039724	4864543	-58327.46		MIB
mopsMIB009	12168	0	6500	56732	100464	76826357.30	*	MIB
mopsMIB010	24336	0	13000	113128	178693	58714511.02	*	MIB
mopsMIB011	6084	0	3250	28542	61912	46176589.81	*	MIB
mopsMIB012	342586	0	35136	245952	809591	-60671181.71	*	MIB
mopsMIB013	6562	0	672	4704	14873	-959206.58	*	MIB
mopsMIB014	6562	0	672	4704	14806	-961513.04		MIB
mopsMIB015	2731	0	1365	2730	6825	6344.86	*	MIB
mopsMIB016	1	0	118738	41517	416674	410.02		MIB
mopsMIP001	34	0	113	296	399	20675.00		MIP
mopsMIP002	1197	0	257	1831	3632	6341.44	*	MIP
mopsMIP003	306	0	47	271	1408	2726.80		MIP
mopsMIP004	5	46	1018	231	12764	1842.00		MIP
mopsMIP005	2416	90	2955	9496	81762	2536.81		MIP
mopsMIP006	2356	91	2865	9256	73341	208.68		MIP
mopsMIP007	357	0	73	535	1518	693.84		MIP
mopsMIP008	1629	0	402	2525	8610	4153.53	*	MIP
mopsMIP009	930	90	615	1692	11248	21188.11	*	MIP

Table C.2.: Instances of the MOPSLIB test set.

[1] a * indicates that this solution is not proven to be in the optimality tolerance of MOPS

D. Test Results

name	without aggregation				with aggregation				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
neos22	6	777291.43	0.99	0.38	6	777191.43	0.99	4.50	0.000273
...									
momentum1	0	79203.00	0.18	7.67	29	79203.04	0.18	12.36	-0.000001
atlanta-ip	254	81.28	0.00	15.59	394	81.28	0.00	20.64	-0.000033
momentum2	79	10696.29	0.68	68.83	179	10696.59	0.68	101.03	-0.000059
dano3mip	3	576.23	0.00	1.30	32	576.24	0.00	59.53	-0.000110
fiber	89	382576.78	0.91	0.22	93	382638.99	0.91	0.23	-0.000249
ches5	34	-7372.07	0.73	0.08	37	-7372.00	0.73	0.08	-0.000662
momentum3	697	92033.22	0.00	215.61	978	92326.39	0.00	570.41	-0.001695
lrn	224	44301814.15	0.88	4.03	342	44306047.77	0.88	8.62	-0.002911
neos3	0	-6111.38	0.07	1.31	14	-6082.65	0.07	4.91	-0.004140
multiD	17	3716.49	0.02	0.08	20	3727.38	0.02	0.09	-0.004506
neos2	0	-4360.27	0.07	0.89	13	-4336.13	0.07	3.89	-0.004666
danoint	0	62.64	0.00	0.01	11	62.65	0.01	0.22	-0.005650
timtab2	54	100112.69	0.02	0.06	93	106236.65	0.02	0.19	-0.005840
dano3_5	1	576.23	0.00	0.94	12	576.24	0.01	32.72	-0.006988
gesa2_o	58	25586469.19	0.36	0.12	74	25588610.23	0.37	0.12	-0.007058
ran10x26	93	4019.63	0.39	0.09	108	4023.88	0.40	0.11	-0.010282
a1c1s1	397	3365.38	0.23	1.03	497	3475.07	0.24	1.64	-0.010441
egout	17	565.88	0.99	0.02	22	568.10	1.00	0.05	-0.011061
fixnet6	52	3564.30	0.85	0.11	74	3601.85	0.86	0.16	-0.013497
gesa3_o	26	27849140.37	0.10	0.06	35	27851274.98	0.11	0.11	-0.013561
gesa2	45	25586549.38	0.36	0.14	58	25591166.32	0.38	0.16	-0.015219
timtab1	18	34269.53	0.01	0.01	30	49751.13	0.03	0.06	-0.021033
BASF6-10	84	20906.74	0.17	1.05	172	20915.93	0.19	1.88	-0.021179
BASF6-5	80	11791.59	0.16	0.67	157	11799.69	0.19	1.09	-0.024184
dano3_4	2	576.23	0.00	0.94	17	576.24	0.03	40.62	-0.026305
ran13x13	87	2929.39	0.42	0.06	104	2945.17	0.45	0.09	-0.028149
mas74	0	10482.80	0.00	0.01	13	10526.33	0.03	0.09	-0.033024
aflow30a	77	1032.00	0.28	0.44	148	1038.00	0.31	0.64	-0.034319
mas76	0	38893.90	0.00	0.02	12	38936.52	0.04	0.08	-0.038356
dano3_3	2	576.23	0.01	0.91	18	576.24	0.05	45.92	-0.040965
multiB	9	3609.56	0.03	0.05	19	3627.72	0.08	0.49	-0.049661
vpm2	65	11.47	0.41	0.05	106	11.68	0.46	0.09	-0.054200
vpm5	0	3001.95	0.00	0.03	21	3002.02	0.06	0.17	-0.060049
multiC	7	1447.49	0.02	0.03	47	1487.38	0.08	1.30	-0.061710
vpm2a	30	11.47	0.23	0.03	57	11.68	0.30	0.06	-0.070762
b4-10	14	12997.04	0.09	0.11	127	13088.64	0.17	0.61	-0.079268
rentacar	12	29017833.29	0.06	0.17	15	29151329.73	0.16	0.47	-0.093460
modglob	80	20675896.08	0.79	0.08	112	20706399.26	0.89	0.11	-0.098537
b4-12	20	14291.34	0.06	0.19	199	14503.76	0.17	0.86	-0.110261
aflow40b	148	1054.00	0.30	3.66	310	1075.00	0.43	7.53	-0.129362
b4-20b	10	22093.99	0.03	2.52	146	22313.58	0.20	13.26	-0.168172
multiA	23	3512.67	0.04	0.11	51	3559.78	0.21	1.34	-0.172175
Con-12	14	3230.33	0.30	0.03	104	4378.27	0.48	0.24	-0.185481
tr24-30	695	186022.74	0.61	0.50	984	237735.59	0.81	0.77	-0.196063
tr6-15	82	31252.53	0.79	0.05	222	37237.48	0.98	0.22	-0.198477
vpm1	11	17.00	0.35	0.00	22	18.00	0.56	0.03	-0.218182
clorox	148	12183.95	0.56	0.31	191	17360.22	0.81	0.52	-0.251801
b4-10b	3	13735.45	0.12	0.23	45	13826.25	0.37	1.48	-0.254235
mod011	55	-61787185.56	0.04	1.03	345	-59836429.49	0.30	1.86	-0.257919
dcmulti	9	184522.72	0.13	0.03	89	185634.93	0.39	0.26	-0.264404
con-24	12	12690.11	0.13	0.05	170	16845.27	0.40	0.53	-0.276603
tr6-30	172	45919.74	0.68	0.05	369	60973.69	0.98	0.47	-0.302019
tr24-15	336	98886.65	0.69	0.22	787	136111.83	1.00	0.98	-0.308198
multiF	27	1832.07	0.11	0.02	109	2043.54	0.43	0.12	-0.314699
tr12-15	159	54554.69	0.65	0.05	423	73789.76	0.99	0.42	-0.332308
pp08a	69	5576.53	0.61	0.05	179	7189.99	0.97	0.14	-0.350626
b4-12b	6	15571.01	0.07	0.39	77	15781.85	0.44	2.97	-0.367313
tr12-30	350	86236.45	0.61	0.17	897	129852.81	0.99	1.12	-0.387798
set1ch	153	42196.72	0.45	0.16	469	52560.13	0.91	0.31	-0.459982
pp08aCUTS	28	6048.22	0.30	0.06	129	7188.53	0.91	0.19	-0.609989

Table D.1.: Comparison between a flow cover cut generator that does not generate aggregated flow cover cuts and one that does.

name	with cuts out of cuts cuts	xLP	gap closed	time (s)	without cuts out of cuts cuts	xLP	gap closed	time (s)	Δ
mitre	850	114963.00	0.54	4.03	754	114909.00	0.41	3.88	0.130283
a1c1s1	397	3365.38	0.23	1.03	201	2192.94	0.11	0.61	0.111598
gen	53	112286.02	0.85	0.09	38	112276.38	0.80	0.06	0.052583
ran12x21	117	3361.72	0.40	0.11	103	3335.15	0.35	0.09	0.052437
binkar10_1	55	6656.92	0.19	0.26	45	6653.90	0.16	0.23	0.028750
fixnet6	52	3564.30	0.85	0.11	43	3499.03	0.83	0.11	0.023459
net12	139	72.00	0.28	10.55	89	69.00	0.26	8.55	0.015248
aflow30a	77	1032.00	0.28	0.44	83	1030.00	0.27	0.38	0.011440
tr6-15	82	31252.53	0.79	0.05	82	30920.70	0.77	0.03	0.011005
ran13x13	87	2929.39	0.42	0.06	85	2923.45	0.41	0.06	0.010596
lseu	29	1006.00	0.60	0.03	26	1003.00	0.59	0.03	0.010515
modglob	80	20675896.08	0.79	0.08	74	20672764.29	0.78	0.06	0.010117
p0282	49	254545.00	0.95	0.09	49	254164.00	0.95	0.08	0.004672
egout	17	565.88	0.99	0.02	17	565.13	0.99	0.00	0.003731
BASF6-5	80	11791.59	0.16	0.67	79	11791.13	0.16	0.69	0.001360
lrn	224	44301814.15	0.88	4.03	254	44300215.97	0.88	6.45	0.001099
gesa3_o	26	27849140.37	0.10	0.06	25	27849063.75	0.10	0.05	0.000487
gesa2	45	25586549.38	0.36	0.14	40	25586413.48	0.36	0.11	0.000448
momentum3	697	92033.22	0.00	215.61	535	91987.89	0.00	167.49	0.000262
multiA	23	3512.67	0.04	0.11	20	3512.62	0.04	0.06	0.000150
atlanta-ip	254	81.28	0.00	15.59	220	81.28	0.00	12.78	0.000105
mod011	55	-61787185.56	0.04	1.03	43	-61787962.93	0.04	0.59	0.000103
gesa2_o	58	25586469.19	0.36	0.12	58	25586449.27	0.36	0.09	0.000066
khb05250	88	106724316.42	0.98	0.17	86	106723801.31	0.98	0.16	0.000047
roll3000	85	11099.34	0.00	1.41	82	11099.28	0.00	1.22	0.000036
prod2	86	-85.31	0.39	0.50	87	-85.31	0.39	0.62	0.000009
...									
momentum2	79	10696.29	0.68	68.83	71	10696.32	0.68	68.58	-0.000005
prod1	63	-81.47	0.42	0.23	62	-81.47	0.42	0.22	-0.000059
fiber	89	382576.78	0.91	0.22	88	382624.88	0.91	0.25	-0.000192
vpm2	65	11.47	0.41	0.05	66	11.47	0.41	0.03	-0.001589
BASF6-10	84	20906.74	0.17	1.05	94	20907.45	0.17	1.38	-0.001634
p0548	162	7967.00	0.91	0.16	158	7983.00	0.91	0.14	-0.001937
ran10x26	93	4019.63	0.39	0.09	79	4020.96	0.40	0.08	-0.003215
aflow40b	148	1054.00	0.30	3.66	149	1060.00	0.33	3.78	-0.036961
mod008	24	295.00	0.25	0.11	27	296.00	0.32	0.11	-0.062232

Table D.2.: Comparison between a flow cover cut generator that does generate cuts out of cuts and one that does not.

name	with integer var. bounds cuts	xLP	gap closed	time (s)	without integer var. bounds cuts	xLP	gap closed	time (s)	Δ
ches3	19	-1303896.92	1.00	0.03	0	-1303932.92	0.00	0.02	1.000000
ches5	34	-7372.07	0.73	0.08	0	-7421.26	0.28	0.06	0.452388
neos671048	7	2999.00	0.50	3.14	0	2001.00	0.25	1.80	0.249750
roll3000	85	11099.34	0.00	1.41	82	11099.28	0.00	1.25	0.000036
...									

Table D.3.: Comparison between a flow cover cut generator that does use binary variable bounds on integer variables and one that does not.

	with improved knapsack				with normal knapsack				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
ches3	19	-1303896.92	1.00	0.03	20	-1303932.92	0.00	0.01	1.000000
m20-75-1	74	-53226.01	0.67	0.39	0	-59156.76	0.00	0.31	0.671297
tr24-15	336	98886.65	0.69	0.22	5	18429.26	0.02	0.06	0.666130
tr12-15	159	54554.69	0.65	0.05	6	17381.99	0.01	0.06	0.642200
tr6-30	172	45919.74	0.68	0.05	1	14315.61	0.05	0.03	0.634057
tr6-15	82	31252.53	0.79	0.05	2	12244.04	0.16	0.02	0.630372
tr24-30	695	186022.74	0.61	0.50	5	27948.60	0.01	0.16	0.599318
m20-75-5	72	-55037.62	0.60	0.39	0	-60527.44	0.00	0.33	0.598122
tr12-30	350	86236.45	0.61	0.17	4	20912.38	0.02	0.06	0.580804
m20-75-2	75	-54436.00	0.57	0.39	0	-59987.20	0.00	0.36	0.574349
m20-75-4	75	-56669.00	0.56	0.44	0	-61651.23	0.00	0.34	0.559849
m20-75-3	73	-55795.88	0.51	0.41	0	-60670.37	0.00	0.33	0.512437
fiber	89	382576.78	0.91	0.22	39	265936.97	0.44	0.19	0.466834
ches5	34	-7372.07	0.73	0.08	29	-7406.98	0.41	0.05	0.321003
set1ch	153	42196.72	0.45	0.16	0	35118.11	0.14	0.01	0.314186
ches1	2	69.12	0.18	0.01	0	67.99	0.00	0.01	0.178064
a1c1s1	397	3365.38	0.23	1.03	244	2194.16	0.11	0.66	0.111483
fixnet6	52	3564.30	0.85	0.11	37	3335.57	0.77	0.09	0.082216
lrn	224	44301814.15	0.88	4.03	119	44198527.78	0.81	2.76	0.071030
rentacar	12	29017833.29	0.06	0.17	2	28928379.62	0.00	0.11	0.062626
b4-10	14	12997.04	0.09	0.11	5	12947.21	0.04	0.08	0.043125
gesa2_o	58	25586469.19	0.36	0.12	36	25573940.58	0.32	0.11	0.041299
b4-12	20	14291.34	0.06	0.19	6	14213.83	0.02	0.09	0.040233
gesa3_o	26	27849140.37	0.10	0.06	14	27843412.69	0.06	0.05	0.036387
mod011	55	-61787185.56	0.04	1.03	3	-61941416.54	0.02	0.22	0.020392
net12	139	72.00	0.28	10.55	0	69.00	0.26	3.59	0.015248
timtab2	54	100112.69	0.02	0.06	1	84523.00	0.00	0.03	0.014866
gesa3	12	27851274.98	0.11	0.08	4	27849063.75	0.10	0.06	0.014048
gesa2	45	25586549.38	0.36	0.14	28	25584194.25	0.36	0.14	0.007763
timtab1	18	34269.53	0.01	0.01	0	28694.00	0.00	0.01	0.007575
dano3_3	2	576.23	0.01	0.91	1	576.23	0.00	1.53	0.005684
vpm2a	30	11.47	0.23	0.03	24	11.46	0.22	0.03	0.004318
multiD	17	3716.49	0.02	0.08	19	3707.15	0.01	0.08	0.003869
khb05250	88	106724316.42	0.98	0.17	88	106688575.21	0.98	0.19	0.003243
dano3_4	2	576.23	0.00	0.94	1	576.23	0.00	1.89	0.003155
vpm2	65	11.47	0.41	0.05	66	11.46	0.41	0.05	0.002105
multiA	23	3512.67	0.04	0.11	19	3512.22	0.04	0.05	0.001639
dano3_5	1	576.23	0.00	0.94	1	576.23	0.00	2.66	0.000804
neos22	6	777291.43	0.99	0.38	6	777191.43	0.99	0.42	0.000273
bc1	1	2.19	0.55	7.19	0	2.19	0.55	7.34	0.000200
b4-12b	6	15571.01	0.07	0.39	5	15570.99	0.07	0.39	0.000050
roll3000	85	11099.34	0.00	1.41	82	11099.26	0.00	1.30	0.000049
b4-20b	10	22093.99	0.03	2.52	6	22093.93	0.03	1.80	0.000047
dano3mip	3	576.23	0.00	1.30	1	576.23	0.00	1.12	0.000005
...									
atlanta-ip	254	81.28	0.00	15.59	297	81.28	0.00	19.22	-0.000108

Table D.4.: Comparison between the default flow cover cut generator and a version that uses $y_j^* = 1$ for variables without variable upper bound in the objective function of the flow cover finding knapsack problem.

	exact knapsack				heuristic knapsack				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
ches3	19	-1303896.92	1.00	0.03	18	-1303932.92	0.00	0.02	1.000000
m20-75-1	74	-53226.01	0.67	0.39	0	-59156.76	0.00	0.30	0.671297
m20-75-5	72	-55037.62	0.60	0.39	0	-60527.44	0.00	0.33	0.598122
m20-75-2	75	-54436.00	0.57	0.39	1	-59987.14	0.00	0.41	0.574343
m20-75-4	75	-56669.00	0.56	0.44	0	-61651.23	0.00	0.31	0.559849
m20-75-3	73	-55795.88	0.51	0.41	0	-60670.37	0.00	0.30	0.512437
tr6-30	172	45919.74	0.68	0.05	44	22393.90	0.21	0.03	0.471987
fiber	89	382576.78	0.91	0.22	19	279819.65	0.50	0.11	0.411271
tr6-15	82	31252.53	0.79	0.05	44	21513.62	0.46	0.03	0.322968
tr12-30	350	86236.45	0.61	0.17	190	54367.61	0.32	0.12	0.283350
mod010	7	6535.00	0.18	0.83	2	6533.00	0.06	0.44	0.125654
a1c1s1	397	3365.38	0.23	1.03	314	2397.88	0.13	0.88	0.092092
lrn	224	44301814.15	0.88	4.03	199	44225571.09	0.83	3.99	0.052433
ches5	34	-7372.07	0.73	0.08	35	-7377.68	0.68	0.05	0.051560
b4-10	14	12997.04	0.09	0.11	12	12967.48	0.06	0.11	0.025580
tr24-30	695	186022.74	0.61	0.50	677	181868.22	0.60	0.48	0.015751
b4-12	20	14291.34	0.06	0.19	19	14261.78	0.04	0.19	0.015343
ran13x13	87	2929.39	0.42	0.06	85	2921.77	0.41	0.06	0.013598
gesa3	12	27851274.98	0.11	0.08	8	27850058.93	0.10	0.12	0.007725
fixnet6	52	3564.30	0.85	0.11	54	3553.47	0.85	0.11	0.003891
gesa2	45	25586549.38	0.36	0.14	41	25585499.80	0.36	0.14	0.003460
gesa2_o	58	25586469.19	0.36	0.12	58	25585535.60	0.36	0.11	0.003077
tr12-15	159	54554.69	0.65	0.05	158	54438.49	0.65	0.05	0.002008
msc98-ip	233	19553706.29	0.10	16.23	235	19553341.12	0.10	14.16	0.001146
atlanta-ip	254	81.28	0.00	15.59	125	81.27	0.00	12.52	0.000545
vpm2	65	11.47	0.41	0.05	66	11.47	0.41	0.03	0.000514
binkar10_1	55	6656.92	0.19	0.26	51	6656.89	0.19	0.27	0.000246
roll3000	85	11099.34	0.00	1.41	107	11099.34	0.00	1.23	0.000005
momentum2	79	10696.29	0.68	68.83	73	10696.27	0.68	60.02	0.000004
...									
momentum3	697	92033.22	0.00	215.61	674	92033.32	0.00	243.38	-0.000001
danoint	0	62.64	0.00	0.01	1	62.64	0.00	0.03	-0.000004
modglob	80	20675896.08	0.79	0.08	85	20675899.93	0.79	0.06	-0.000012
set1ch	153	42196.72	0.45	0.16	155	42198.77	0.45	0.14	-0.000091
vpm2a	30	11.47	0.23	0.03	34	11.47	0.23	0.03	-0.000092
pp08a	69	5576.53	0.61	0.05	72	5577.73	0.61	0.05	-0.000261
multiD	17	3716.49	0.02	0.08	20	3717.78	0.02	0.09	-0.000532
p0282	49	254545.00	0.95	0.09	58	254589.00	0.95	0.08	-0.000540
p0548	162	7967.00	0.91	0.16	142	7978.00	0.91	0.16	-0.001332
BASF6-5	80	11791.59	0.16	0.67	90	11792.09	0.17	0.75	-0.001505
prod1	63	-81.47	0.42	0.23	105	-81.40	0.42	0.28	-0.001620
prod2	86	-85.31	0.39	0.50	166	-85.24	0.39	0.95	-0.001793
mitre	850	114963.00	0.54	4.03	764	114964.00	0.54	3.64	-0.002413
BASF6-10	84	20906.74	0.17	1.05	99	20910.05	0.18	1.30	-0.007641
sp97ar	4	652568542.94	0.00	3.28	13	652645381.10	0.01	3.33	-0.008154
ran10x26	93	4019.63	0.39	0.09	86	4023.84	0.40	0.09	-0.010185
dcmulti	9	184522.72	0.13	0.03	14	184567.50	0.14	0.01	-0.010646
aflow30a	77	1032.00	0.28	0.44	65	1035.00	0.30	0.36	-0.017159
pp08aCUTS	28	6048.22	0.30	0.06	30	6089.24	0.33	0.06	-0.021943
lseu	29	1006.00	0.60	0.03	34	1013.00	0.62	0.03	-0.024534
aflow40b	148	1054.00	0.30	3.66	126	1059.00	0.33	3.92	-0.030800
mod008	24	295.00	0.25	0.11	28	296.00	0.32	0.11	-0.062232
ran12x21	117	3361.72	0.40	0.11	130	3398.03	0.48	0.11	-0.071674
mod011	55	-61787185.56	0.04	1.03	266	-59712010.24	0.32	1.22	-0.274369

Table D.5.: Comparison between the default flow cover cut generator that uses an exact method for solving the flow cover finding knapsack problem and a version that uses a heuristic.

	less than				less than or equal				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
fiber	89	382576.78	0.91	0.22	51	193492.98	0.15	0.19	0.756781
khb05250	88	106724316.42	0.98	0.17	45	104126076.70	0.74	0.14	0.235759
modglob	80	20675896.08	0.79	0.08	49	20608767.91	0.57	0.05	0.216850
lseu	29	1006.00	0.60	0.03	8	948.00	0.40	0.00	0.203282
p0282	49	254545.00	0.95	0.09	86	242243.00	0.80	0.11	0.150864
mod010	7	6535.00	0.18	0.83	3	6533.00	0.06	0.44	0.125654
m20-75-2	75	-54436.00	0.57	0.39	59	-55580.79	0.46	0.42	0.118444
m20-75-1	74	-53226.01	0.67	0.39	63	-54035.61	0.58	0.53	0.091638
m20-75-5	72	-55037.62	0.60	0.39	65	-55748.86	0.52	0.44	0.077490
aflow30a	77	1032.00	0.28	0.44	44	1023.00	0.23	0.22	0.051478
p0548	162	7967.00	0.91	0.16	164	7625.00	0.87	0.16	0.041398
m20-75-4	75	-56669.00	0.56	0.44	70	-57004.19	0.52	0.48	0.037665
m20-75-3	73	-55795.88	0.51	0.41	69	-56106.81	0.48	0.48	0.032687
fixnet6	52	3564.30	0.85	0.11	60	3475.23	0.82	0.12	0.032015
binkar10_1	55	6656.92	0.19	0.26	37	6654.73	0.17	0.20	0.020846
net12	139	72.00	0.28	10.55	117	69.00	0.26	8.66	0.015248
tr6-15	82	31252.53	0.79	0.05	78	30874.75	0.77	0.03	0.012528
mitre	850	114963.00	0.54	4.03	1440	114958.00	0.52	6.56	0.012063
gesa2	45	25586549.38	0.36	0.14	45	25583187.88	0.35	0.17	0.011081
BASF6-10	84	20906.74	0.17	1.05	164	20903.44	0.16	1.62	0.007600
gesa2_o	58	25586469.19	0.36	0.12	70	25584764.94	0.36	0.20	0.005618
dano3_3	2	576.23	0.01	0.91	2	576.23	0.00	0.89	0.004867
ran10x26	93	4019.63	0.39	0.09	76	4018.28	0.39	0.08	0.003272
ran13x13	87	2929.39	0.42	0.06	66	2927.69	0.42	0.06	0.003028
BASF6-5	80	11791.59	0.16	0.67	126	11790.58	0.16	0.89	0.003015
dano3_4	2	576.23	0.00	0.94	2	576.23	0.00	0.91	0.002701
vpm2	65	11.47	0.41	0.05	71	11.46	0.41	0.03	0.002560
p2756	260	3065.00	0.86	1.03	399	3064.00	0.86	1.33	0.002353
prod1	63	-81.47	0.42	0.23	51	-81.56	0.42	0.20	0.002002
nsrand-ipx	83	49832.00	0.41	1.94	90	49829.00	0.41	2.11	0.001293
atlanta-ip	254	81.28	0.00	15.59	305	81.27	0.00	19.03	0.001043
set1ch	153	42196.72	0.45	0.16	141	42173.48	0.45	0.09	0.001031
dano3_5	1	576.23	0.00	0.94	1	576.23	0.00	0.55	0.000804
gesa3	12	27851274.98	0.11	0.08	12	27851190.35	0.11	0.08	0.000538
gesa3_o	26	27849140.37	0.10	0.06	21	27849060.37	0.10	0.08	0.000508
momentum3	697	92033.22	0.00	215.61	447	91971.43	0.00	114.00	0.000357
gen	53	112286.02	0.85	0.09	39	112285.99	0.85	0.08	0.000180
roll3000	85	11099.34	0.00	1.41	98	11099.27	0.00	1.61	0.000040
momentum2	79	10696.29	0.68	68.83	78	10696.27	0.68	61.74	0.000005
dano3mip	3	576.23	0.00	1.30	2	576.23	0.00	0.73	0.000005
bc1	1	2.19	0.55	7.19	3	2.19	0.55	10.84	0.000005
...									
prod2	86	-85.31	0.39	0.50	89	-85.31	0.39	0.61	-0.000032
pp08a	69	5576.53	0.61	0.05	66	5577.47	0.61	0.05	-0.000204
lrn	224	44301814.15	0.88	4.03	384	44321742.90	0.89	5.06	-0.013705
aflow40b	148	1054.00	0.30	3.66	95	1060.00	0.33	2.80	-0.036961
clorox	148	12183.95	0.56	0.31	144	13499.84	0.62	0.31	-0.064012
ran12x21	117	3361.72	0.40	0.11	90	3419.31	0.52	0.09	-0.113670
rgn	20	48.80	0.00	0.02	60	64.60	0.47	0.03	-0.473054

Table D.6.: Comparison between the default flow cover cut generator that uses $L^- = \{j \in N^- : \lambda y_j^* < x_j^*\}$ and a version that uses $L^- = \{j \in N^- : \lambda y_j^* \leq x_j^*\}$.

	with lifting				without lifting				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
lseu	29	1006.00	0.60	0.03	7	948.00	0.40	0.01	0.203282
mod008	24	295.00	0.25	0.11	21	292.00	0.07	0.11	0.186696
mod010	7	6535.00	0.18	0.83	4	6533.00	0.06	0.58	0.125654
binkar10_1	55	6656.92	0.19	0.26	56	6651.83	0.14	0.25	0.048419
ran12x21	117	3361.72	0.40	0.11	118	3341.70	0.36	0.09	0.039522
ran13x13	87	2929.39	0.42	0.06	77	2911.79	0.39	0.06	0.031402
nsrand-ipx	83	49832.00	0.41	1.94	80	49768.00	0.38	1.92	0.027586
ran10x26	93	4019.63	0.39	0.09	75	4008.48	0.37	0.08	0.027020
clorox	148	12183.95	0.56	0.31	147	11801.13	0.54	0.33	0.018622
mitre	850	114963.00	0.54	4.03	1309	114956.00	0.52	5.09	0.016889
p0282	49	254545.00	0.95	0.09	48	253881.00	0.94	0.08	0.008143
pp08aCUTS	28	6048.22	0.30	0.06	30	6035.06	0.30	0.06	0.007036
fiber	89	382576.78	0.91	0.22	92	381942.47	0.90	0.22	0.002539
lrn	224	44301814.15	0.88	4.03	205	44300543.58	0.88	3.36	0.000874
dano3mip	3	576.23	0.00	1.30	4	576.23	0.00	1.24	0.000004
...									
roll3000	85	11099.34	0.00	1.41	97	11099.36	0.00	1.75	-0.000011
modglob	80	20675896.08	0.79	0.08	84	20675924.27	0.79	0.08	-0.000091
BASF6-5	80	11791.59	0.16	0.67	77	11791.62	0.16	0.67	-0.000098
gesa2_o	58	25586469.19	0.36	0.12	55	25586515.03	0.36	0.11	-0.000151
atlanta-ip	254	81.28	0.00	15.59	307	81.28	0.00	18.12	-0.000153
multiD	17	3716.49	0.02	0.08	18	3717.78	0.02	0.09	-0.000532
m20-75-3	73	-55795.88	0.51	0.41	75	-55789.00	0.51	0.41	-0.000723
dano3_5	1	576.23	0.00	0.94	7	576.23	0.00	1.22	-0.000822
a1c1s1	397	3365.38	0.23	1.03	417	3374.36	0.23	1.00	-0.000854
m20-75-1	74	-53226.01	0.67	0.39	75	-53218.00	0.67	0.39	-0.000906
gesa2	45	25586549.38	0.36	0.14	43	25587124.30	0.36	0.14	-0.001895
msc98-ip	233	19553706.29	0.10	16.23	357	19554347.01	0.10	17.45	-0.002011
vpm2a	30	11.47	0.23	0.03	33	11.48	0.23	0.03	-0.002276
BASF6-10	84	20906.74	0.17	1.05	91	20907.91	0.17	1.28	-0.002701
vpm2	65	11.47	0.41	0.05	74	11.48	0.41	0.05	-0.003809
m20-75-5	72	-55037.62	0.60	0.39	75	-54992.00	0.60	0.39	-0.004970
dano3_4	2	576.23	0.00	0.94	9	576.23	0.01	1.98	-0.004972
dano3_3	2	576.23	0.01	0.91	6	576.23	0.01	2.03	-0.007302
aflow30a	77	1032.00	0.28	0.44	91	1034.00	0.29	0.34	-0.011440
p0548	162	7967.00	0.91	0.16	173	8079.00	0.93	0.16	-0.013557
set1ch	153	42196.72	0.45	0.16	158	42609.30	0.47	0.14	-0.018313
fixnet6	52	3564.30	0.85	0.11	73	3615.69	0.87	0.12	-0.018472
aflow40b	148	1054.00	0.30	3.66	128	1057.00	0.32	3.14	-0.018480
bienst1	3	11.72	0.00	0.02	76	14.05	0.07	0.25	-0.066393
bienst2	4	11.72	0.00	0.02	100	14.92	0.07	0.31	-0.074490
timtab2	54	100112.69	0.02	0.06	142	228646.61	0.14	0.08	-0.122567
timtab1	18	34269.53	0.01	0.01	68	176889.67	0.20	0.03	-0.193757

Table D.7.: Comparison between the default version of the flow cover cut generator that generates LSGFCIs and a version that generates SGFCIs.

	new				old				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
p0282	49	254545.00	0.95	0.09	0	180001.00	0.04	0.00	0.914162
p2756	260	3065.00	0.86	1.03	0	2702.00	0.01	0.23	0.854012
p0548	162	7967.00	0.91	0.16	0	4571.00	0.50	0.01	0.411072
gesa2_o	58	25586469.19	0.36	0.12	0	25476489.68	0.00	0.02	0.362530
khb05250	88	106724316.42	0.98	0.17	60	103353448.27	0.67	0.11	0.305865
gen	53	112286.02	0.85	0.09	0	112233.78	0.57	0.01	0.284958
ches5	34	-7372.07	0.73	0.08	45	-7401.51	0.46	0.03	0.270719
neos671048	7	2999.00	0.50	3.14	0	2001.00	0.25	1.25	0.249750
mod008	24	295.00	0.25	0.11	0	291.00	0.00	0.02	0.248928
lrn	224	44301814.15	0.88	4.03	484	44004362.80	0.67	5.49	0.204558
lseu	29	1006.00	0.60	0.03	0	948.00	0.40	0.00	0.203282
fiber	89	382576.78	0.91	0.22	96	334836.04	0.72	0.20	0.191076
binkar10_1	55	6656.92	0.19	0.26	0	6637.19	0.00	0.02	0.187870
BASF6-10	84	20906.74	0.17	1.05	33	20849.88	0.04	0.62	0.130937
mod010	7	6535.00	0.18	0.83	0	6533.00	0.06	0.16	0.125654
BASF6-5	80	11791.59	0.16	0.67	32	11753.32	0.05	0.30	0.114290
p0033	21	2890.00	0.65	0.00	0	2829.00	0.54	0.00	0.107313
gesa3_o	26	27849140.37	0.10	0.06	3	27833697.15	0.00	0.03	0.098108
vpm2a	30	11.47	0.23	0.03	3	11.21	0.14	0.00	0.090018
vpm2	65	11.47	0.41	0.05	31	11.14	0.32	0.02	0.084205
nsrand-ipx	83	49832.00	0.41	1.94	0	49668.00	0.34	0.91	0.070690
msc98-ip	233	19553706.29	0.10	16.23	266	19532747.49	0.04	17.17	0.065798
rentacar	12	29017833.29	0.06	0.17	2	28928379.62	0.00	0.06	0.062626
gesa2	45	25586549.38	0.36	0.14	18	25568019.75	0.30	0.06	0.061080
ran12x21	117	3361.72	0.40	0.11	117	3345.76	0.37	0.08	0.031502
ran13x13	87	2929.39	0.42	0.06	70	2915.36	0.40	0.05	0.025032
ran10x26	93	4019.63	0.39	0.09	80	4009.41	0.37	0.06	0.024764
tr12-30	350	86236.45	0.61	0.17	342	83576.78	0.58	0.09	0.023647
tr6-15	82	31252.53	0.79	0.05	80	30559.87	0.76	0.03	0.022970
tr24-15	336	98886.65	0.69	0.22	329	96257.85	0.67	0.14	0.021765
mitre	850	114963.00	0.54	4.03	0	114954.00	0.52	4.06	0.021714
net12	139	72.00	0.28	10.55	47	69.00	0.26	3.44	0.015248
b4-12	20	14291.34	0.06	0.19	6	14263.36	0.04	0.06	0.014522
b4-10	14	12997.04	0.09	0.11	6	12981.92	0.07	0.06	0.013084
tr24-30	695	186022.74	0.61	0.50	687	182817.02	0.60	0.47	0.012154
pp08aCUTS	28	6048.22	0.30	0.06	17	6027.18	0.29	0.02	0.011254
prod1	63	-81.47	0.42	0.23	0	-81.84	0.41	0.03	0.008341
tr12-15	159	54554.69	0.65	0.05	160	54091.98	0.65	0.05	0.007994
pp08a	69	5576.53	0.61	0.05	58	5556.14	0.61	0.01	0.004430
b4-10b	3	13735.45	0.12	0.23	4	13733.88	0.11	0.16	0.004403
prod2	86	-85.31	0.39	0.50	0	-85.45	0.38	0.09	0.003818
b4-20b	10	22093.99	0.03	2.52	1	22089.07	0.03	1.41	0.003764
tr6-30	172	45919.74	0.68	0.05	171	45749.58	0.68	0.05	0.003414
atlanta-ip	254	81.28	0.00	15.59	58	81.25	0.00	4.20	0.002806
dcmulti	9	184522.72	0.13	0.03	6	184514.12	0.13	0.00	0.002044
b4-12b	6	15571.01	0.07	0.39	3	15569.86	0.07	0.31	0.002004
multiA	23	3512.67	0.04	0.11	7	3512.24	0.04	0.03	0.001546
sp97ar	4	652568542.94	0.00	3.28	0	652560391.11	0.00	2.11	0.000865
momentum3	697	92033.22	0.00	215.61	38	91953.22	0.00	23.23	0.000462
multiD	17	3716.49	0.02	0.08	12	3715.69	0.02	0.08	0.000334
neos22	6	777291.43	0.99	0.38	6	777191.43	0.99	0.28	0.000273
gesa3	12	27851274.98	0.11	0.08	9	27851253.46	0.11	0.06	0.000137
roll3000	85	11099.34	0.00	1.41	69	11099.29	0.00	2.56	0.000032
momentum2	79	10696.29	0.68	68.83	39	10696.25	0.68	24.70	0.000008
...									
modglob	80	20675896.08	0.79	0.08	77	20675924.27	0.79	0.05	-0.000091
dano3_5	1	576.23	0.00	0.94	6	576.23	0.00	0.78	-0.000715
m20-75-3	73	-55795.88	0.51	0.41	75	-55789.00	0.51	0.44	-0.000723
egout	17	565.88	0.99	0.02	20	566.05	0.99	0.01	-0.000836
a1c1s1	397	3365.38	0.23	1.03	398	3374.90	0.23	0.50	-0.000906
m20-75-1	74	-53226.01	0.67	0.39	75	-53218.00	0.67	0.42	-0.000906
neos4	0	-49463016984.65	0.55	1.75	11	-49460660286.31	0.55	4.50	-0.001238
dano3_4	2	576.23	0.00	0.94	5	576.23	0.00	0.67	-0.001708
dano3_3	2	576.23	0.01	0.91	3	576.23	0.01	0.69	-0.002345
m20-75-5	72	-55037.62	0.60	0.39	75	-54992.00	0.60	0.42	-0.004970
aflow30a	77	1032.00	0.28	0.44	76	1034.00	0.29	0.34	-0.011440
mod011	55	-61787185.56	0.04	1.03	69	-61688425.49	0.06	0.69	-0.013058
clorox	148	12183.95	0.56	0.31	142	12498.71	0.58	0.16	-0.015312
set1ch	153	42196.72	0.45	0.16	138	42598.00	0.47	0.05	-0.017811
fixnet6	52	3564.30	0.85	0.11	69	3617.36	0.87	0.09	-0.019072
aflow40b	148	1054.00	0.30	3.66	117	1060.00	0.33	3.02	-0.036961
bienst1	3	11.72	0.00	0.02	76	14.05	0.07	0.14	-0.066384
bienst2	4	11.72	0.00	0.02	100	14.92	0.07	0.20	-0.074451
ches1	2	69.12	0.18	0.01	14	69.64	0.26	0.02	-0.081340
timtab2	54	100112.69	0.02	0.06	136	212322.94	0.12	0.05	-0.107002
timtab1	18	34269.53	0.01	0.01	70	159114.11	0.18	0.02	-0.169608

Table D.8.: Comparison between the new and the old flow cover cut generator.

name	with cuts out of cuts				without cuts out of cuts				
	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
p2756	268	3063.00	0.86	1.31	253	2988.00	0.68	1.14	0.176449
m20-75-2	404	-52198.38	0.81	26.84	86	-53851.43	0.63	9.62	0.171031
m20-75-4	355	-54734.14	0.78	24.20	84	-56198.70	0.61	10.14	0.164571
m20-75-3	418	-53688.55	0.73	17.34	93	-55131.00	0.58	9.75	0.151640
m20-75-1	371	-51261.82	0.89	20.38	84	-52573.33	0.75	9.91	0.148448
m20-75-5	436	-53165.14	0.80	34.99	84	-54509.03	0.66	9.92	0.146418
tr12-30	823	129106.22	0.99	1.31	768	114684.92	0.86	0.76	0.128221
b4-10b	92	13936.59	0.68	2.03	71	13892.29	0.56	1.88	0.124029
tr6-30	345	60742.59	0.98	0.53	318	54863.91	0.86	0.31	0.117941
set1ch	413	54240.68	0.99	0.39	402	51762.61	0.88	0.31	0.109989
tr12-15	377	73297.66	0.98	0.52	366	67220.49	0.87	0.33	0.104990
tr24-15	716	135396.11	0.99	1.19	667	123906.69	0.90	0.78	0.095124
gesa3_o	170	27954866.51	0.77	0.55	127	27941030.41	0.68	0.44	0.087898
gesa3	116	27963866.98	0.83	0.50	93	27952173.63	0.75	0.42	0.074286
ches1	39	71.03	0.48	0.09	31	70.57	0.41	0.08	0.072794
gesa2_o	230	25774034.47	0.98	0.53	189	25752343.70	0.91	0.38	0.071500
momentum1	348	84003.68	0.31	48.66	207	81471.52	0.24	27.69	0.069660
gesa2	145	25775745.36	0.99	0.42	157	25755490.51	0.92	0.34	0.066767
tr6-15	197	36817.15	0.97	0.25	200	34865.27	0.91	0.14	0.064730
a1c1s1	686	5398.31	0.42	2.75	557	4783.40	0.36	2.09	0.058530
b4-20b	263	22466.70	0.32	18.56	247	22398.81	0.27	17.67	0.051997
b4-12b	159	15839.55	0.54	4.12	143	15816.09	0.50	3.78	0.040879
tr24-30	984	238004.34	0.81	0.83	984	227902.75	0.77	0.66	0.038299
con-24	266	17959.84	0.48	0.81	233	17428.04	0.44	0.53	0.035401
khb05250	25	96428245.44	0.05	0.16	16	96070104.35	0.01	0.12	0.032497
aflow30a	236	1075.00	0.53	1.61	174	1070.00	0.50	0.67	0.028599
arki001	232	7579847.42	0.20	6.52	155	7579814.00	0.18	6.30	0.027544
p0548	142	8232.00	0.94	0.20	132	8013.00	0.92	0.17	0.026509
ran13x13	120	3024.25	0.59	0.16	112	3009.58	0.57	0.11	0.026167
qnet1_o	83	15607.95	0.89	0.24	77	15505.78	0.87	0.22	0.025970
b4-10	157	13294.83	0.35	0.78	161	13266.11	0.32	0.74	0.024855
aflow40b	406	1082.00	0.47	16.73	269	1078.00	0.45	5.59	0.024640
multiF	129	2050.53	0.44	0.17	125	2036.01	0.42	0.12	0.021605
multiD	76	3884.64	0.09	0.38	31	3836.45	0.07	0.25	0.019947
ran10x26	106	4078.28	0.54	0.14	104	4070.56	0.52	0.12	0.018683
gt2	30	20726.00	0.94	0.01	25	20593.00	0.93	0.02	0.017260
modglob	112	20684799.25	0.82	0.12	106	20679686.42	0.80	0.09	0.016516
vpm5	107	3002.68	0.58	0.59	99	3002.66	0.57	0.50	0.016480
bell3a	17	870990.98	0.39	0.06	15	870793.00	0.38	0.05	0.016150
fiber	70	385611.78	0.92	0.24	59	381596.53	0.90	0.20	0.016070
roll3000	161	12120.94	0.57	3.50	163	12092.62	0.56	3.41	0.015797
dcmulti	94	186330.79	0.56	0.26	96	186264.76	0.54	0.22	0.015698
binkar10_1	62	6702.62	0.62	0.41	62	6701.06	0.61	0.38	0.014857
mod011	289	-59384558.73	0.36	1.77	317	-59493500.53	0.35	1.77	0.014404
bc1	63	2.61	0.72	117.22	45	2.58	0.70	109.08	0.013111
multiC	39	1497.49	0.09	0.22	48	1489.21	0.08	0.22	0.012813
neos7	269	668555.02	0.86	1.11	248	664469.73	0.84	0.99	0.011054
momentum2	1200	10804.25	0.70	106.23	1266	10750.92	0.69	114.74	0.010479
multiB	54	3627.88	0.08	0.27	33	3624.47	0.07	0.20	0.009315
BASF6-10	153	20920.55	0.20	1.89	150	20916.67	0.19	1.75	0.008945
msc98-ip	411	19564697.17	0.14	23.41	349	19561947.01	0.13	18.67	0.008634
lrn	367	44307469.65	0.88	9.83	264	44295040.60	0.87	9.17	0.008548
vpm2a	114	13.04	0.76	0.17	102	13.02	0.75	0.09	0.007301
mitre	1125	115026.00	0.69	5.84	1103	115023.00	0.68	5.59	0.007238
p0282	56	254264.00	0.95	0.11	55	253746.00	0.94	0.11	0.006352
bell5	31	8928247.67	0.89	0.09	20	8926029.48	0.89	0.05	0.006196
ran12x21	126	3451.51	0.58	0.19	123	3448.51	0.57	0.14	0.005930
egout	16	568.10	1.00	0.02	14	566.94	0.99	0.01	0.005754
fixnet6	82	3662.77	0.88	0.30	66	3646.79	0.88	0.17	0.005742
Con-12	168	4577.84	0.51	0.38	157	4549.96	0.51	0.25	0.004504
mas76	13	38998.28	0.09	0.51	12	38993.49	0.09	0.33	0.004310
timtab2	348	313976.30	0.22	0.62	325	309603.97	0.22	0.39	0.004169
dano3_3	23	576.24	0.04	5.30	16	576.24	0.04	4.00	0.003555
vpm2	133	12.94	0.79	0.17	129	12.93	0.79	0.09	0.003542
lseu	29	1030.00	0.68	0.05	25	1029.00	0.68	0.02	0.003505
timtab1	215	245864.72	0.30	0.28	200	245218.44	0.29	0.19	0.000878
bienst1	89	14.07	0.07	0.53	75	14.05	0.07	0.47	0.000725
bienst2	119	14.94	0.07	0.56	98	14.91	0.07	0.45	0.000648

continued on the next page

	with cuts out of cuts				without cuts out of cuts				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
danoint	108	62.69	0.02	0.47	105	62.69	0.02	0.39	0.000204
prod1	129	-81.38	0.42	0.41	135	-81.38	0.42	0.41	0.000150
dano3mip	509	576.56	0.00	25.53	452	576.55	0.00	22.70	0.000092
mas74	14	10568.56	0.07	0.50	14	10568.52	0.07	0.36	0.000031
prod2	128	-85.22	0.39	1.81	126	-85.22	0.39	1.36	0.000009
...									
atlanta-ip	133	81.25	0.00	12.23	116	81.25	0.00	12.27	-0.000006
dano3_4	60	576.25	0.11	9.41	64	576.25	0.11	9.34	-0.000315
BASF6-5	166	11800.92	0.19	1.12	147	11801.17	0.19	0.98	-0.000751
ches5	43	-7370.94	0.74	0.14	49	-7370.80	0.74	0.22	-0.001215
multiA	31	3562.04	0.22	0.31	34	3562.43	0.22	0.26	-0.001457
pp08a	154	7143.27	0.96	0.17	164	7161.51	0.96	0.11	-0.003962
multiE	79	2279.49	0.25	0.17	78	2284.15	0.26	0.14	-0.008073
momentum3	2694	93079.94	0.01	3270.88	3219	94616.58	0.02	3832.94	-0.008882
neos3	106	-5998.04	0.08	9.53	140	-5929.39	0.09	7.55	-0.009892
dano3_5	82	576.29	0.08	11.28	89	576.30	0.09	12.05	-0.012985
pp08aCUTS	123	7157.97	0.90	0.24	117	7189.71	0.91	0.17	-0.016978
neos2	87	-4311.53	0.08	6.42	121	-4223.46	0.10	5.72	-0.017026
b4-12	213	14587.80	0.21	1.03	212	14639.68	0.24	0.86	-0.026929
qnet1	71	15333.09	0.60	0.27	73	15432.66	0.66	0.30	-0.056714
p0033	8	2890.00	0.65	0.00	9	2926.00	0.71	0.01	-0.063333
mod008	17	296.00	0.32	0.28	16	299.00	0.50	0.22	-0.186696
clorox	152	13783.99	0.64	0.72	165	20708.61	0.98	0.61	-0.336850

Table D.9.: Comparison between the default version of the cMIR cut generator that generates cuts out of cuts and one that does not.

	with algorithmic improvements				without algorithmic improvements				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
b4-10b	92	13936.59	0.68	2.03	28	13756.33	0.18	0.78	0.504701
gesa2_o	230	25774034.47	0.98	0.53	172	25647400.29	0.56	0.44	0.417429
b4-12b	159	15839.55	0.54	4.12	40	15610.30	0.14	2.64	0.399406
swath	45	373.88	0.30	45.83	39	334.50	0.00	15.22	0.296334
b4-20b	263	22466.70	0.32	18.56	56	22135.24	0.06	7.34	0.253851
neos671048	3	2999.00	0.50	2.25	0	2001.00	0.25	2.06	0.249750
neos7	269	668555.02	0.86	1.11	75	590969.65	0.65	0.34	0.209932
rgn	76	81.80	0.99	0.16	78	75.30	0.79	0.14	0.194524
multiA	31	3562.04	0.22	0.31	5	3512.78	0.04	0.11	0.180017
aflow30a	236	1075.00	0.53	1.61	226	1051.00	0.39	3.05	0.137274
momentum1	348	84003.68	0.31	48.66	207	79204.21	0.18	32.91	0.132035
gesa2	145	25775745.36	0.99	0.42	152	25735855.19	0.85	0.42	0.131492
modglob	112	20684799.25	0.82	0.12	139	20644380.92	0.69	0.12	0.130567
aflow40b	406	1082.00	0.47	16.73	300	1063.00	0.35	20.06	0.117042
b4-10	157	13294.83	0.35	0.78	153	13169.68	0.24	0.73	0.108309
gesa3_o	170	27954866.51	0.77	0.55	157	27939930.76	0.68	0.41	0.094884
dcmulti	94	186330.79	0.56	0.26	68	185984.80	0.48	0.25	0.082253
ran10x26	106	4078.28	0.54	0.14	75	4045.22	0.46	0.12	0.080037
vpm5	107	3002.68	0.58	0.59	68	3002.58	0.50	0.50	0.078861
ches1	39	71.03	0.48	0.09	31	70.54	0.40	0.09	0.077788
bell3a	17	870990.98	0.39	0.06	8	870118.76	0.32	0.05	0.071152
b4-12	213	14587.80	0.21	1.03	173	14454.97	0.14	0.92	0.068952
fixnet6	82	3662.77	0.88	0.30	94	3478.62	0.82	0.47	0.066188
gesa3	116	27963866.98	0.83	0.50	131	27954645.62	0.77	0.49	0.058582
a1c1s1	686	5398.31	0.42	2.75	624	4841.65	0.37	2.97	0.052985
bell5	31	8928247.67	0.89	0.09	12	8911014.71	0.85	0.02	0.048138
khb05250	25	96428245.44	0.05	0.16	3	95919464.00	0.00	0.02	0.046166
ran13x13	120	3024.25	0.59	0.16	88	2999.68	0.55	0.14	0.043819
dano3_4	60	576.25	0.11	9.41	33	576.25	0.07	5.31	0.038523
multiD	76	3884.64	0.09	0.38	48	3810.93	0.06	0.38	0.030511
mod011	289	-59384558.73	0.36	1.77	263	-59611381.49	0.33	1.80	0.029989
vpm2	133	12.94	0.79	0.17	139	12.83	0.76	0.17	0.029653
roll3000	161	12120.94	0.57	3.50	163	12068.70	0.54	3.55	0.029136
seymour1	7	404.13	0.04	1.78	2	403.93	0.01	1.06	0.028544
arki001	232	7579847.42	0.20	6.52	66	7579813.21	0.18	2.58	0.028195
rentacar	21	29274325.20	0.24	0.41	34	29235639.28	0.22	1.27	0.027084
multiC	39	1497.49	0.09	0.22	26	1483.89	0.07	0.19	0.021052

continued on the next page

	with algorithmic improvements				without algorithmic improvements				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
momentum2	1200	10804.25	0.70	106.23	509	10697.37	0.68	90.73	0.021002
BASF6-10	153	20920.55	0.20	1.89	198	20911.68	0.18	2.50	0.020435
clorox	152	13783.99	0.64	0.72	139	13424.92	0.62	0.73	0.017467
neos3	106	-5998.04	0.08	9.53	9	-6111.38	0.07	1.41	0.016331
ran12x21	126	3451.51	0.58	0.19	107	3443.64	0.57	0.19	0.015535
bc1	63	2.61	0.72	117.22	25	2.58	0.70	102.69	0.013614
multiF	129	2050.53	0.44	0.17	87	2043.50	0.43	0.20	0.010463
vpm2a	114	13.04	0.76	0.17	101	13.01	0.75	0.19	0.009654
neos2	87	-4311.53	0.08	6.42	9	-4360.27	0.07	1.05	0.009423
msc98-ip	411	19564697.17	0.14	23.41	170	19561947.01	0.13	18.62	0.008634
BASF6-5	166	11800.92	0.19	1.12	176	11798.07	0.18	1.58	0.008513
dano3_5	82	576.29	0.08	11.28	74	576.28	0.07	10.94	0.008121
egout	16	568.10	1.00	0.02	19	566.59	0.99	0.02	0.007511
momentum3	2694	93079.94	0.01	3270.88	620	91952.39	0.00	1403.89	0.006518
ches2	44	-2891.65	0.11	0.11	33	-2891.67	0.11	0.06	0.006419
multiB	54	3627.88	0.08	0.27	33	3625.80	0.07	0.34	0.005683
m20-75-3	418	-53688.55	0.73	17.34	436	-53741.39	0.73	16.24	0.005554
con-24	266	17959.84	0.48	0.81	238	17882.60	0.47	0.78	0.005142
m20-75-2	404	-52198.38	0.81	26.84	401	-52242.91	0.80	19.16	0.004607
timtab2	348	313976.30	0.22	0.62	350	309224.26	0.22	0.61	0.004531
pp08a	154	7143.27	0.96	0.17	147	7128.87	0.95	0.17	0.003130
dano3_3	23	576.24	0.04	5.30	18	576.24	0.04	4.30	0.002682
m20-75-4	355	-54734.14	0.78	24.20	358	-54754.30	0.78	18.28	0.002265
dano3mip	509	576.56	0.00	25.53	253	576.33	0.00	27.41	0.002050
multiE	79	2279.49	0.25	0.17	51	2278.77	0.25	0.17	0.001240
neos22	51	777686.43	0.99	0.47	18	777291.43	0.99	0.53	0.001079
timtab1	215	245864.72	0.30	0.28	240	245606.81	0.29	0.30	0.000350
bienst2	119	14.94	0.07	0.56	107	14.93	0.07	0.53	0.000181
bienst1	89	14.07	0.07	0.53	85	14.07	0.07	0.53	0.000034
...									
atlanta-ip	133	81.25	0.00	12.23	54	81.26	0.00	10.66	-0.000460
tr6-30	345	60742.59	0.98	0.53	355	60767.34	0.98	0.58	-0.000496
danoint	108	62.69	0.02	0.47	93	62.70	0.02	0.45	-0.001513
m20-75-5	436	-53165.14	0.80	34.99	537	-53147.81	0.80	21.05	-0.001888
m20-75-1	371	-51261.82	0.89	20.38	387	-51240.89	0.90	17.45	-0.002370
tr24-15	716	135396.11	0.99	1.19	642	136027.95	1.00	1.33	-0.005231
tr12-15	377	73297.66	0.98	0.52	355	73634.10	0.98	0.53	-0.005812
tr12-30	823	129106.22	0.99	1.31	768	129814.98	0.99	1.28	-0.006302
tr24-30	984	238004.34	0.81	0.83	984	240217.12	0.82	0.86	-0.008389
tr6-15	197	36817.15	0.97	0.25	183	37117.55	0.98	0.25	-0.009962
Con-12	168	4577.84	0.51	0.38	144	4651.23	0.52	0.36	-0.011857
pp08aCUTS	123	7157.97	0.90	0.24	101	7181.07	0.91	0.23	-0.012361
set1ch	413	54240.68	0.99	0.39	335	54520.18	1.00	0.39	-0.012406
lrn	367	44307469.65	0.88	9.83	352	44330450.86	0.90	9.55	-0.015804
ches5	43	-7370.94	0.74	0.14	27	-7369.12	0.76	0.09	-0.016696
qnet1	71	15333.09	0.60	0.27	68	15368.94	0.62	0.28	-0.020421
opt1217	10	-20.02	0.00	0.02	9	-19.94	0.02	0.03	-0.021277

Table D.10.: Comparison between a version of the cMIR cut generator that uses improved bound substitution and aggregation strategies and one that uses traditional strategies.

	new				old				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
p2756	268	3063.00	0.86	1.31	72	2703.00	0.01	0.38	0.846954
gesa3_o	170	27954866.51	0.77	0.55	98	27851070.81	0.11	0.16	0.659396
gesa2_o	230	25774034.47	0.98	0.53	203	25584088.13	0.35	0.22	0.626128
binkar10_1	62	6702.62	0.62	0.41	0	6637.19	0.00	0.02	0.623129
b4-10b	92	13936.59	0.68	2.03	9	13734.24	0.11	0.23	0.566564
gesa2	145	25775745.36	0.99	0.42	142	25605284.27	0.42	0.20	0.561898
gt2	30	20726.00	0.94	0.01	57	17191.00	0.48	0.02	0.458747
b4-12b	159	15839.55	0.54	4.12	10	15582.14	0.09	0.42	0.448451
dcmulti	94	186330.79	0.56	0.26	10	184624.15	0.15	0.02	0.405719
fiber	70	385611.78	0.92	0.24	25	297623.78	0.57	0.12	0.352160
swath	45	373.88	0.30	45.83	39	334.50	0.00	15.17	0.296334

continued on the next page

	new				old				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
b4-20b	263	22466.70	0.32	18.56	26	22102.29	0.04	4.06	0.279082
b4-10	157	13294.83	0.35	0.78	62	12991.41	0.08	0.25	0.262579
modglob	112	20684799.25	0.82	0.12	141	20607394.77	0.57	0.08	0.250046
neos671048	3	2999.00	0.50	2.25	0	2001.00	0.25	1.47	0.249750
p0548	142	8232.00	0.94	0.20	149	6443.00	0.73	0.14	0.216551
neos7	269	668555.02	0.86	1.11	100	592877.12	0.65	0.30	0.204770
arki001	232	7579847.42	0.20	6.52	36	7579599.81	0.00	0.11	0.204091
ches1	39	71.03	0.48	0.09	13	69.81	0.29	0.02	0.192152
roll3000	161	12120.94	0.57	3.50	166	11779.41	0.38	2.75	0.190494
con-24	266	17959.84	0.48	0.81	189	15166.23	0.29	0.22	0.185966
gesa3	116	27963866.98	0.83	0.50	150	27935763.55	0.65	0.23	0.178536
multiA	31	3562.04	0.22	0.31	20	3514.36	0.05	0.17	0.174228
rgn	76	81.80	0.99	0.16	58	75.98	0.81	0.06	0.174201
b4-12	213	14587.80	0.21	1.03	63	14272.55	0.05	0.30	0.163638
vpm2a	114	13.04	0.76	0.17	97	12.58	0.60	0.09	0.158689
aflow30a	236	1075.00	0.53	1.61	205	1048.00	0.37	1.38	0.154433
lrn	367	44307469.65	0.88	9.83	662	44099096.67	0.74	6.16	0.143299
aflow40b	406	1082.00	0.47	16.73	264	1059.00	0.33	16.72	0.141682
msc98-ip	411	19564697.17	0.14	23.41	31	19520966.15	0.00	0.84	0.137290
momentum1	348	84003.68	0.31	48.66	270	79203.50	0.18	16.92	0.132054
mod010	3	6535.00	0.18	0.44	0	6533.00	0.06	0.17	0.125654
Con-12	168	4577.84	0.51	0.38	140	3825.00	0.39	0.11	0.121641
ran10x26	106	4078.28	0.54	0.14	84	4033.52	0.43	0.16	0.108363
dano3_4	60	576.25	0.11	9.41	1	576.23	0.00	0.72	0.107394
vpm5	107	3002.68	0.58	0.59	43	3002.54	0.48	0.17	0.106176
ran12x21	126	3451.51	0.58	0.19	102	3403.24	0.49	0.16	0.095280
ran13x13	120	3024.25	0.59	0.16	84	2977.83	0.51	0.11	0.082805
dano3_5	82	576.29	0.08	11.28	1	576.23	0.00	0.77	0.079197
timtab2	348	313976.30	0.22	0.62	346	232193.65	0.14	0.19	0.077986
timtab1	215	245864.72	0.30	0.28	187	196413.90	0.23	0.05	0.067182
vpm2	133	12.94	0.79	0.17	108	12.69	0.73	0.09	0.064383
fixnet6	82	3662.77	0.88	0.30	101	3484.29	0.82	0.20	0.064150
rentacar	21	29274325.20	0.24	0.41	22	29194392.10	0.19	0.28	0.055961
lseu	29	1030.00	0.68	0.05	16	1015.00	0.63	0.02	0.052573
dano3_3	23	576.24	0.04	5.30	1	576.23	0.00	0.66	0.042647
seymour1	7	404.13	0.04	1.78	0	403.85	0.00	0.08	0.040336
qnet1_o	83	15607.95	0.89	0.24	84	15475.39	0.86	0.17	0.033696
m20-75-1	371	-51261.82	0.89	20.38	342	-51526.09	0.86	3.02	0.029912
a1c1s1	686	5398.31	0.42	2.75	706	5100.00	0.39	1.28	0.028395
bell5	31	8928247.67	0.89	0.09	23	8918901.75	0.87	0.01	0.026107
multiC	39	1497.49	0.09	0.22	36	1483.47	0.07	0.22	0.021706
pp08aCUTS	123	7157.97	0.90	0.24	110	7117.45	0.88	0.09	0.021671
momentum2	1200	10804.25	0.70	106.23	334	10696.26	0.68	53.48	0.021221
multiF	129	2050.53	0.44	0.17	113	2036.40	0.42	0.11	0.021024
neos3	106	-5998.04	0.08	9.53	3	-6111.38	0.07	1.30	0.016331
tr12-30	823	129106.22	0.99	1.31	863	127289.79	0.97	0.72	0.016150
mitre	1125	115026.00	0.69	5.84	642	115020.00	0.67	3.55	0.014476
tr6-30	345	60742.59	0.98	0.53	369	60080.33	0.97	0.25	0.013287
ches5	43	-7370.94	0.74	0.14	32	-7372.29	0.73	0.05	0.012477
mas76	13	38998.28	0.09	0.51	14	38984.44	0.08	0.11	0.012456
ches2	44	-2891.65	0.11	0.11	20	-2891.67	0.10	0.03	0.010410
net12	99	72.00	0.28	9.49	99	70.00	0.27	7.75	0.010165
m20-75-5	436	-53165.14	0.80	34.99	366	-53255.39	0.79	3.17	0.009833
m20-75-4	355	-54734.14	0.78	24.20	369	-54819.33	0.77	3.36	0.009573
neos2	87	-4311.53	0.08	6.42	1	-4360.27	0.07	0.88	0.009423
multiE	79	2279.49	0.25	0.17	64	2274.41	0.24	0.09	0.008799
bc1	63	2.61	0.72	117.22	34	2.59	0.71	43.62	0.008175
mas74	14	10568.56	0.07	0.50	13	10558.81	0.06	0.12	0.007396
momentum3	2694	93079.94	0.01	3270.88	1865	91960.54	0.00	236.81	0.006470
multiB	54	3627.88	0.08	0.27	30	3625.68	0.07	0.23	0.005999
tr12-15	377	73297.66	0.98	0.52	389	72964.10	0.97	0.28	0.005763
p0282	56	254264.00	0.95	0.11	50	253955.00	0.95	0.08	0.003789
m20-75-2	404	-52198.38	0.81	26.84	397	-52232.84	0.80	3.30	0.003565
BASF6-10	153	20920.55	0.20	1.89	185	20919.14	0.20	1.58	0.003260
prod1	129	-81.38	0.42	0.41	26	-81.50	0.42	0.11	0.002833
multiD	76	3884.64	0.09	0.38	81	3878.76	0.09	0.30	0.002437
dano3mip	509	576.56	0.00	25.53	20	576.29	0.00	13.86	0.002397
prod2	128	-85.22	0.39	1.81	45	-85.28	0.39	0.41	0.001422

continued on the next page

	new				old				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
neos22	51	777686.43	0.99	0.47	12	777191.43	0.99	0.19	0.001352
BASF6-5	166	11800.92	0.19	1.12	159	11800.48	0.19	0.99	0.001310
bienst1	89	14.07	0.07	0.53	89	14.07	0.07	0.20	0.000073
. . .									
bienst2	119	14.94	0.07	0.56	106	14.94	0.08	0.28	-0.000158
neos4	0	-49463016984.65	0.55	2.05	2	-49461919046.45	0.55	1.67	-0.000577
atlanta-ip	133	81.25	0.00	12.23	116	81.26	0.00	16.01	-0.000660
tr6-15	197	36817.15	0.97	0.25	204	36859.69	0.97	0.11	-0.001411
danoint	108	62.69	0.02	0.47	88	62.70	0.02	0.30	-0.002292
swath2	24	334.50	0.00	1.47	22	334.64	0.00	2.91	-0.002763
nsrand-ipx	85	49980.00	0.47	2.01	153	49987.00	0.48	2.84	-0.003017
sp97ar	16	652734973.16	0.02	4.41	36	652769660.25	0.02	4.97	-0.003681
pp08a	154	7143.27	0.96	0.17	155	7163.52	0.96	0.08	-0.004400
tr24-15	716	135396.11	0.99	1.19	690	136007.29	1.00	0.66	-0.005060
tr24-30	984	238004.34	0.81	0.83	984	240036.19	0.82	0.45	-0.007703
m20-75-3	418	-53688.55	0.73	17.34	424	-53605.48	0.74	3.09	-0.008734
set1ch	413	54240.68	0.99	0.39	327	54517.88	1.00	0.16	-0.012304
opt1217	10	-20.02	0.00	0.02	13	-19.94	0.02	0.03	-0.021277
clorox	152	13783.99	0.64	0.72	121	14479.89	0.67	0.23	-0.033852
blend2	7	7.04	0.19	0.08	16	7.10	0.26	0.12	-0.074834
qnet1	71	15333.09	0.60	0.27	72	15500.76	0.70	0.20	-0.095509
p0033	8	2890.00	0.65	0.00	26	2958.00	0.77	0.01	-0.119628
mod008	17	296.00	0.32	0.28	20	298.00	0.44	0.09	-0.124464
mod011	289	-59384558.73	0.36	1.77	236	-58347923.42	0.50	1.11	-0.137059
bell3a	17	870990.98	0.39	0.06	12	873351.52	0.59	0.02	-0.192562
harp2	0	-74325169.00	0.00	0.12	121	-74084955.00	0.56	1.28	-0.564716
khb05250	25	96428245.44	0.05	0.16	59	103966809.04	0.73	0.39	-0.684033

Table D.14.: Comparison between the new and the old cMIR cut generator of MOPS .

	cMIR				flow cover				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
rgn	76	81.80	0.99	0.16	20	48.80	0.00	0.02	0.988022
gt2	30	20726.00	0.94	0.01	0	13461.00	0.00	0.00	0.942800
qnet1_o	83	15607.95	0.89	0.24	0	12095.57	0.00	0.03	0.892799
gesa3	116	27963866.98	0.83	0.50	15	27851274.98	0.11	0.08	0.715278
gesa3_o	170	27954866.51	0.77	0.55	35	27851274.98	0.11	0.11	0.658099
gesa2_o	230	25774034.47	0.98	0.53	74	25588610.23	0.37	0.12	0.611221
gesa2	145	25775745.36	0.99	0.42	58	25591166.32	0.38	0.16	0.608435
qnet1	71	15333.09	0.60	0.27	0	14274.10	0.00	0.03	0.603209
roll3000	161	12120.94	0.57	3.50	85	11099.34	0.00	1.62	0.569810
vpm5	107	3002.68	0.58	0.59	21	3002.02	0.06	0.17	0.522720
vpm2a	114	13.04	0.76	0.17	57	11.68	0.30	0.06	0.461025
vpm1	40	20.00	1.00	0.00	22	18.00	0.56	0.03	0.436364
binkar10_1	62	6702.62	0.62	0.41	55	6656.92	0.19	0.31	0.435259
bell3a	17	870990.98	0.39	0.06	0	866171.73	0.00	0.02	0.393132
vpm2	133	12.94	0.79	0.17	106	11.68	0.46	0.09	0.327762
b4-10b	92	13936.59	0.68	2.03	45	13826.25	0.37	1.48	0.308932
ches1	39	71.03	0.48	0.09	2	69.12	0.18	0.02	0.300790
swath	45	373.88	0.30	45.83	20	334.50	0.00	15.41	0.296334
neos7	269	668555.02	0.86	1.11	0	562977.43	0.57	0.12	0.285673
timtab1	215	245864.72	0.30	0.28	30	49751.13	0.03	0.06	0.266430
multiE	79	2279.49	0.25	0.17	0	2133.08	0.00	0.02	0.253515
m20-75-2	404	-52198.38	0.81	26.84	75	-54436.00	0.57	0.41	0.231513
m20-75-1	371	-51261.82	0.89	20.38	74	-53226.01	0.67	0.39	0.222324
m20-75-3	418	-53688.55	0.73	17.34	73	-55795.88	0.51	0.41	0.221535
m20-75-4	355	-54734.14	0.78	24.20	75	-56669.00	0.56	0.42	0.217419
aflow30a	236	1075.00	0.53	1.61	148	1038.00	0.31	0.64	0.211631
arki001	232	7579847.42	0.20	6.52	0	7579599.81	0.00	0.20	0.204091
m20-75-5	436	-53165.14	0.80	34.99	72	-55037.62	0.60	0.39	0.204008
timtab2	348	313976.30	0.22	0.62	93	106236.65	0.02	0.19	0.198097
a1c1s1	686	5398.31	0.42	2.75	497	3475.07	0.24	1.64	0.183062
b4-10	157	13294.83	0.35	0.78	127	13088.64	0.17	0.61	0.178435
ran12x21	126	3451.51	0.58	0.19	120	3361.72	0.40	0.11	0.177233
dcmulti	94	186330.79	0.56	0.26	89	185634.93	0.39	0.26	0.165428
bc1	63	2.61	0.72	117.22	1	2.19	0.55	7.95	0.165217
mitre	1125	115026.00	0.69	5.84	850	114963.00	0.54	4.09	0.151997

continued on the next page

name	cMIR cuts	cMIR xLP	cMIR gap closed	cMIR time (s)	flow cover cuts	flow cover xLP	flow cover gap closed	flow cover time (s)	Δ
ran13x13	120	3024.25	0.59	0.16	104	2945.17	0.45	0.09	0.141071
momentum1	348	84003.68	0.31	48.66	29	79203.04	0.18	12.36	0.132067
ran10x26	106	4078.28	0.54	0.14	108	4023.88	0.40	0.11	0.131714
b4-20b	263	22466.70	0.32	18.56	146	22313.58	0.20	13.26	0.117269
ches2	44	-2891.65	0.11	0.11	0	-2891.88	0.00	0.01	0.113324
b4-12b	159	15839.55	0.54	4.12	77	15781.85	0.44	2.97	0.100527
rentacar	21	29274325.20	0.24	0.41	15	29151329.73	0.16	0.47	0.086108
lseu	29	1030.00	0.68	0.05	29	1006.00	0.60	0.03	0.084117
dano3_4	60	576.25	0.11	9.41	17	576.24	0.03	40.62	0.077898
bienst2	119	14.94	0.07	0.56	4	11.72	0.00	0.05	0.074897
set1ch	413	54240.68	0.99	0.39	469	52560.13	0.91	0.31	0.074592
con-24	266	17959.84	0.48	0.81	170	16845.27	0.40	0.53	0.074195
dano3_5	82	576.29	0.08	11.28	12	576.24	0.01	32.72	0.071394
bienst1	89	14.07	0.07	0.53	3	11.72	0.00	0.12	0.067022
multiD	76	3884.64	0.09	0.38	20	3727.38	0.02	0.09	0.065094
nsrand-ipx	85	49980.00	0.47	2.01	83	49832.00	0.41	1.94	0.063793
mod008	17	296.00	0.32	0.28	24	295.00	0.25	0.11	0.062232
mod011	289	-59384558.73	0.36	1.77	345	-59836429.49	0.30	1.86	0.059744
mas76	13	38998.28	0.09	0.51	12	38936.52	0.04	0.08	0.055580
bell5	31	8928247.67	0.89	0.09	0	8908552.45	0.84	0.02	0.055016
b4-12	213	14587.80	0.21	1.03	199	14503.76	0.17	0.86	0.043624
aflow40b	406	1082.00	0.47	16.73	310	1075.00	0.43	7.53	0.043121
seymour1	7	404.13	0.04	1.78	1	403.85	0.00	0.45	0.040336
msc98-ip	411	19564697.17	0.14	23.41	384	19553706.29	0.10	12.67	0.034505
Con-12	168	4577.84	0.51	0.38	104	4378.27	0.48	0.24	0.032246
p0548	142	8232.00	0.94	0.20	162	7967.00	0.91	0.16	0.032077
mas74	14	10568.56	0.07	0.50	13	10526.33	0.03	0.09	0.032030
fixnet6	82	3662.77	0.88	0.30	74	3601.85	0.86	0.16	0.021896
momentum2	1200	10804.25	0.70	106.23	179	10696.59	0.68	101.03	0.021155
sp97ar	16	652734973.16	0.02	4.41	4	652568542.94	0.00	3.20	0.017661
multiC	39	1497.49	0.09	0.22	47	1487.38	0.08	1.30	0.015654
danoint	108	62.69	0.02	0.47	11	62.65	0.01	0.22	0.013192
neos3	106	-5998.04	0.08	9.53	14	-6082.65	0.07	4.91	0.012191
fiber	70	385611.78	0.92	0.24	93	382638.99	0.91	0.23	0.011898
BASF6-10	153	20920.55	0.20	1.89	172	20915.93	0.19	1.88	0.010636
multiF	129	2050.53	0.44	0.17	109	2043.54	0.43	0.12	0.010409
ches5	43	-7370.94	0.74	0.14	37	-7372.00	0.73	0.08	0.009763
multiA	31	3562.04	0.22	0.31	51	3559.78	0.21	1.34	0.008253
neos2	87	-4311.53	0.08	6.42	13	-4336.13	0.07	3.89	0.004757
momentum3	2694	93079.94	0.01	3270.88	978	92326.39	0.00	570.41	0.004356
BASF6-5	166	11800.92	0.19	1.12	157	11799.69	0.19	1.09	0.003681
dano3mip	509	576.56	0.00	25.53	32	576.24	0.00	59.53	0.002803
prod2	128	-85.22	0.39	1.81	86	-85.31	0.39	0.59	0.002216
prod1	129	-81.38	0.42	0.41	63	-81.47	0.42	0.24	0.002134
neos22	51	777686.43	0.99	0.47	6	777191.43	0.99	4.50	0.001352
tr24-30	984	238004.34	0.81	0.83	984	237735.59	0.81	0.77	0.001019
lrn	367	44307469.65	0.88	9.83	342	44306047.77	0.88	8.62	0.000978
multiB	54	3627.88	0.08	0.27	19	3627.72	0.08	0.49	0.000426
...									
atlanta-ip	133	81.25	0.00	12.23	394	81.28	0.00	20.64	-0.002497
p0282	56	254264.00	0.95	0.11	49	254545.00	0.95	0.09	-0.003446
dano3_3	23	576.24	0.04	5.30	18	576.24	0.05	45.92	-0.004067
tr6-30	345	60742.59	0.98	0.53	369	60973.69	0.98	0.47	-0.004636
p2756	268	3063.00	0.86	1.31	260	3065.00	0.86	1.06	-0.004705
tr24-15	716	135396.11	0.99	1.19	787	136111.83	1.00	0.98	-0.005926
tr12-30	823	129106.22	0.99	1.31	897	129852.81	0.99	1.12	-0.006638
tr12-15	377	73297.66	0.98	0.52	423	73789.76	0.99	0.42	-0.008502
pp08a	154	7143.27	0.96	0.17	179	7189.99	0.97	0.14	-0.010152
tr6-15	197	36817.15	0.97	0.25	222	37237.48	0.98	0.22	-0.013939
pp08aCUTS	123	7157.97	0.90	0.24	129	7188.53	0.91	0.19	-0.016348
modglob	112	20684799.25	0.82	0.12	112	20706399.26	0.89	0.11	-0.069776
clorox	152	13783.99	0.64	0.72	191	17360.22	0.81	0.52	-0.173966
khb05250	25	96428245.44	0.05	0.16	88	106724316.42	0.98	0.48	-0.934243

Table D.15.: Comparison between the cMIR and the aggregated flow cover cut generator.

	whith path-based tightest row aggregation				with traditional aggregation				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
b4-10b	92	13936.59	0.68	2.03	32	13757.01	0.18	0.80	0.502796
b4-12b	159	15839.55	0.54	4.12	42	15609.45	0.14	2.74	0.400872
gesa2_o	230	25774034.47	0.98	0.53	156	25681053.15	0.67	0.41	0.306498
swath	45	373.88	0.30	45.83	36	334.50	0.00	15.22	0.296334
b4-20b	263	22466.70	0.32	18.56	72	22137.60	0.07	8.09	0.252042
rgn	76	81.80	0.99	0.16	78	75.30	0.79	0.14	0.194524
multiA	31	3562.04	0.22	0.31	5	3512.78	0.04	0.12	0.180017
b4-10	157	13294.83	0.35	0.78	118	13113.80	0.19	0.72	0.156664
momentum1	348	84003.68	0.31	48.66	344	79682.72	0.19	39.55	0.118871
b4-12	213	14587.80	0.21	1.03	114	14384.72	0.11	0.78	0.105416
gesa3_o	170	27954866.51	0.77	0.55	123	27940138.60	0.68	0.36	0.093564
arki001	232	7579847.42	0.20	6.52	168	7579764.21	0.14	5.36	0.068586
vpm5	107	3002.68	0.58	0.59	80	3002.60	0.52	0.56	0.061708
dcmulti	94	186330.79	0.56	0.26	70	186077.56	0.50	0.25	0.060200
a1c1s1	686	5398.31	0.42	2.75	613	4867.42	0.37	2.86	0.050533
rentacar	21	29274325.20	0.24	0.41	20	29213967.24	0.20	0.66	0.042256
mod011	289	-59384558.73	0.36	1.77	268	-59694755.74	0.32	1.78	0.041013
bell3a	17	870990.98	0.39	0.06	11	870587.05	0.36	0.06	0.032950
seymour1	7	404.13	0.04	1.78	2	403.93	0.01	1.08	0.028544
bell5	31	8928247.67	0.89	0.09	24	8918498.30	0.87	0.08	0.027234
multiD	76	3884.64	0.09	0.38	60	3825.32	0.06	0.55	0.024555
neos7	269	668555.02	0.86	1.11	227	660149.10	0.83	0.95	0.022745
momentum2	1200	10804.25	0.70	106.23	515	10702.37	0.68	75.33	0.020021
vpm2a	114	13.04	0.76	0.17	91	12.99	0.74	0.16	0.018230
gesa2	145	25775745.36	0.99	0.42	155	25770446.16	0.97	0.45	0.017468
gesa3	116	27963866.98	0.83	0.50	119	27961143.18	0.81	0.48	0.017304
clorox	152	13783.99	0.64	0.72	148	13430.00	0.62	0.74	0.017220
aflow30a	236	1075.00	0.53	1.61	201	1072.00	0.51	1.58	0.017159
multiC	39	1497.49	0.09	0.22	38	1487.53	0.08	0.23	0.015422
bc1	63	2.61	0.72	117.22	60	2.58	0.70	136.70	0.010943
multiF	129	2050.53	0.44	0.17	119	2045.25	0.43	0.17	0.007861
vpm2	133	12.94	0.79	0.17	124	12.91	0.78	0.14	0.007857
pp08a	154	7143.27	0.96	0.17	161	7109.31	0.95	0.19	0.007382
aflow40b	406	1082.00	0.47	16.73	313	1081.00	0.46	14.12	0.006160
msc98-ip	411	19564697.17	0.14	23.41	315	19562735.61	0.13	12.39	0.006158
ches1	39	71.03	0.48	0.09	31	70.99	0.47	0.09	0.006148
neos3	106	-5998.04	0.08	9.53	88	-6039.37	0.08	7.39	0.005956
multiB	54	3627.88	0.08	0.27	31	3625.81	0.07	0.30	0.005655
m20-75-3	418	-53688.55	0.73	17.34	436	-53741.39	0.73	16.39	0.005554
pp08aCUTS	123	7157.97	0.90	0.24	109	7148.40	0.89	0.24	0.005117
ches5	43	-7370.94	0.74	0.14	41	-7371.47	0.74	0.17	0.004912
tr24-15	716	135396.11	0.99	1.19	686	134837.02	0.99	1.42	0.004629
m20-75-2	404	-52198.38	0.81	26.84	401	-52242.91	0.80	19.17	0.004607
modglob	112	20684799.25	0.82	0.12	107	20683613.38	0.82	0.11	0.003831
timtab2	348	313976.30	0.22	0.62	345	310289.76	0.22	0.62	0.003515
set1ch	413	54240.68	0.99	0.39	409	54173.11	0.98	0.41	0.002999
m20-75-4	355	-54734.14	0.78	24.20	358	-54754.30	0.78	18.34	0.002265
egout	16	568.10	1.00	0.02	23	567.75	1.00	0.03	0.001743
tr6-15	197	36817.15	0.97	0.25	186	36764.86	0.97	0.25	0.001734
con-24	266	17959.84	0.48	0.81	257	17935.87	0.48	0.80	0.001595
ran12x21	126	3451.51	0.58	0.19	121	3450.71	0.58	0.23	0.001572
ran13x13	120	3024.25	0.59	0.16	110	3023.70	0.59	0.14	0.000975
timtab1	215	245864.72	0.30	0.28	240	245606.81	0.29	0.30	0.000350
BASF6-10	153	20920.55	0.20	1.89	166	20920.45	0.20	1.94	0.000229
bienst2	119	14.94	0.07	0.56	107	14.93	0.07	0.51	0.000181
roll3000	161	12120.94	0.57	3.50	161	12120.82	0.57	3.64	0.000070
tr6-30	345	60742.59	0.98	0.53	343	60740.52	0.98	0.58	0.000042
bienst1	89	14.07	0.07	0.53	85	14.07	0.07	0.53	0.000034
atlanta-ip	133	81.25	0.00	12.23	80	81.25	0.00	10.23	0.000025
...									
dano3mip	509	576.56	0.00	25.53	457	576.56	0.00	22.34	-0.000027
tr24-30	984	238004.34	0.81	0.83	984	238058.93	0.81	0.92	-0.000207
ran10x26	106	4078.28	0.54	0.14	99	4078.37	0.54	0.16	-0.000234
tr12-30	823	129106.22	0.99	1.31	829	129152.91	0.99	1.44	-0.000415
BASF6-5	166	11800.92	0.19	1.12	155	11801.20	0.19	1.17	-0.000848
khb05250	25	96428245.44	0.05	0.16	22	96439441.74	0.05	0.16	-0.001016
danoint	108	62.69	0.02	0.47	91	62.70	0.02	0.45	-0.001836
m20-75-5	436	-53165.14	0.80	34.99	537	-53147.81	0.80	20.89	-0.001888
tr12-15	377	73297.66	0.98	0.52	368	73424.20	0.98	0.55	-0.002186
m20-75-1	371	-51261.82	0.89	20.38	387	-51240.89	0.90	17.30	-0.002370
dano3_4	60	576.25	0.11	9.41	56	576.25	0.11	8.28	-0.002947
neos2	87	-4311.53	0.08	6.42	61	-4287.63	0.08	5.78	-0.004619
dano3_3	23	576.24	0.04	5.30	22	576.24	0.05	6.36	-0.004756
dano3_5	82	576.29	0.08	11.28	88	576.29	0.09	10.70	-0.007377
multiE	79	2279.49	0.25	0.17	68	2283.79	0.26	0.17	-0.007453
momentum3	2694	93079.94	0.01	3270.88	2749	94431.73	0.01	2133.30	-0.007814
fixnet6	82	3662.77	0.88	0.30	91	3690.11	0.89	0.33	-0.009830
Con-12	168	4577.84	0.51	0.38	165	4651.69	0.52	0.38	-0.011933
lrn	367	44307469.65	0.88	9.83	345	44353913.61	0.91	10.11	-0.031940

Table D.11.: Comparison between a version of the cMIR cut generator that uses path based-tightest row aggregation and one that uses the aggregation strategy from literature.

name	with improved bound substitution cuts	xLP	gap closed	time (s)	with traditional bound substitution cuts	xLP	gap closed	time (s)	Δ
swath	45	373.88	0.30	45.83	39	334.50	0.00	15.22	0.296334
neos671048	3	2999.00	0.50	2.25	0	2001.00	0.25	2.06	0.249750
aflow30a	236	1075.00	0.53	1.61	232	1050.00	0.38	3.27	0.142994
momentum1	348	84003.68	0.31	48.66	198	79204.42	0.18	48.59	0.132029
modglob	112	20684799.25	0.82	0.12	138	20648174.35	0.70	0.12	0.118313
aflow40b	406	1082.00	0.47	16.73	299	1063.00	0.35	21.67	0.117042
gesa2_o	230	25774034.47	0.98	0.53	223	25738989.25	0.87	0.47	0.115521
neos7	269	668555.02	0.86	1.11	258	627807.81	0.75	1.00	0.110254
gesa2	145	25775745.36	0.99	0.42	161	25743804.84	0.88	0.41	0.105287
b4-10b	92	13936.59	0.68	2.03	81	13910.49	0.61	1.50	0.073088
ran10x26	106	4078.28	0.54	0.14	75	4050.74	0.47	0.14	0.066684
fixnet6	82	3662.77	0.88	0.30	83	3503.66	0.83	0.36	0.057188
roll3000	161	12120.94	0.57	3.50	163	12035.13	0.52	3.38	0.047864
khb05250	25	96428245.44	0.05	0.16	3	95919464.00	0.00	0.02	0.046166
b4-12b	159	15839.55	0.54	4.12	140	15814.64	0.50	3.44	0.043400
bell3a	17	870990.98	0.39	0.06	14	870493.10	0.35	0.05	0.040614
ran13x13	120	3024.25	0.59	0.16	95	3002.95	0.56	0.14	0.037987
gesa3	116	27963866.98	0.83	0.50	136	27958196.38	0.79	0.50	0.036024
gesa3_o	170	27954866.51	0.77	0.55	159	27950431.83	0.74	0.50	0.028173
vpm2	133	12.94	0.79	0.17	159	12.84	0.76	0.17	0.026973
b4-12	213	14587.80	0.21	1.03	188	14545.39	0.19	0.95	0.022017
momentum2	1200	10804.25	0.70	106.23	638	10697.51	0.68	130.64	0.020975
BASF6-10	153	20920.55	0.20	1.89	209	20912.31	0.18	2.52	0.018979
multiC	39	1497.49	0.09	0.22	41	1488.05	0.08	0.24	0.014610
ran12x21	126	3451.51	0.58	0.19	113	3444.20	0.57	0.20	0.014435
dano3_5	82	576.29	0.08	11.28	79	576.28	0.07	11.23	0.014133
bell5	31	8928247.67	0.89	0.09	22	8923210.45	0.88	0.06	0.014071
neos3	106	-5998.04	0.08	9.53	32	-6092.02	0.07	3.30	0.013541
multiF	129	2050.53	0.44	0.17	103	2042.17	0.42	0.17	0.012450
b4-10	157	13294.83	0.35	0.78	160	13280.56	0.33	0.76	0.012349
multiD	76	3884.64	0.09	0.38	59	3858.12	0.08	0.36	0.010978
arki001	232	7579847.42	0.20	6.52	102	7579834.50	0.19	7.42	0.010647
bc1	63	2.61	0.72	117.22	31	2.59	0.71	115.01	0.010169
dano3_4	60	576.25	0.11	9.41	60	576.25	0.10	9.80	0.009727
msc98-ip	411	19564697.17	0.14	23.41	256	19561947.01	0.13	9.03	0.008634
neos2	87	-4311.53	0.08	6.42	12	-4355.28	0.07	1.00	0.008460
momentum3	2694	93079.94	0.01	3270.88	1721	91952.39	0.00	1813.33	0.006518
ches2	44	-2891.65	0.11	0.11	33	-2891.67	0.11	0.05	0.006419
b4-20b	263	22466.70	0.32	18.56	203	22458.33	0.31	14.70	0.006408
vpm2a	114	13.04	0.76	0.17	117	13.03	0.76	0.20	0.004629
BASF6-5	166	11800.92	0.19	1.12	206	11799.65	0.19	1.61	0.003799
timtab2	348	313976.30	0.22	0.62	354	310797.26	0.22	0.69	0.003031
dano3mip	509	576.56	0.00	25.53	418	576.34	0.00	51.61	0.001930
neos22	51	777686.43	0.99	0.47	18	777291.43	0.99	0.50	0.001079
dcmulti	94	186330.79	0.56	0.26	94	186330.02	0.56	0.26	0.000183
...									
multiB	54	3627.88	0.08	0.27	49	3627.89	0.08	0.26	-0.000033
danoint	108	62.69	0.02	0.47	113	62.70	0.02	0.45	-0.000377
tr6-30	345	60742.59	0.98	0.53	348	60763.12	0.98	0.55	-0.000412
atlanta-ip	133	81.25	0.00	12.23	128	81.26	0.00	14.30	-0.000534
Con-12	168	4577.84	0.51	0.38	144	4590.36	0.51	0.36	-0.002024
con-24	266	17959.84	0.48	0.81	253	17999.78	0.48	0.80	-0.002659
tr12-30	823	129106.22	0.99	1.31	803	129594.20	0.99	1.27	-0.004339
tr24-15	716	135396.11	0.99	1.19	656	136078.83	1.00	1.14	-0.005652
tr12-15	377	73297.66	0.98	0.52	361	73651.68	0.98	0.48	-0.006116
lrn	367	44307469.65	0.88	9.83	353	44318584.76	0.89	9.16	-0.007644
tr24-30	984	238004.34	0.81	0.83	984	240217.12	0.82	0.84	-0.008389
dano3_3	23	576.24	0.04	5.30	32	576.24	0.05	5.72	-0.009131
multiE	79	2279.49	0.25	0.17	61	2285.05	0.26	0.16	-0.009623
pp08a	154	7143.27	0.96	0.17	160	7188.90	0.96	0.17	-0.009915
tr6-15	197	36817.15	0.97	0.25	197	37125.29	0.98	0.25	-0.010219
vpm5	107	3002.68	0.58	0.59	84	3002.69	0.59	0.59	-0.011532
set1ch	413	54240.68	0.99	0.39	336	54521.10	1.00	0.36	-0.012447
pp08aCUTS	123	7157.97	0.90	0.24	101	7192.50	0.92	0.25	-0.018471
qnet1	71	15333.09	0.60	0.27	68	15368.94	0.62	0.25	-0.020421
ches5	43	-7370.94	0.74	0.14	34	-7368.68	0.76	0.14	-0.020734
opt1217	10	-20.02	0.00	0.02	10	-19.94	0.02	0.03	-0.021277
multiA	31	3562.04	0.22	0.31	32	3569.06	0.25	0.25	-0.025675
ches1	39	71.03	0.48	0.09	30	71.39	0.54	0.09	-0.056528
clorox	152	13783.99	0.64	0.72	147	15257.49	0.71	0.72	-0.071678

Table D.12.: Comparison between a version of the cMIR cut generator that uses the improved bound substitution and one that uses the traditional one.

name	with variable bound on integer				without variable bounds on integers				Δ
	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	
neos671048	3	2999.00	0.50	2.25	0	2001.00	0.25	2.06	0.249750
roll3000	161	12120.94	0.57	3.50	163	12035.13	0.52	3.45	0.047864
neos7	269	668555.02	0.86	1.11	288	664102.96	0.84	1.17	0.012046
...									
ches1	39	71.03	0.48	0.09	36	71.04	0.48	0.11	-0.000845
qnet1	71	15333.09	0.60	0.27	68	15368.94	0.62	0.27	-0.020421
ches5	43	-7370.94	0.74	0.14	36	-7367.67	0.77	0.24	-0.029997

Table D.13.: Comparison between a version of the cMIR cut generator that uses variable bounds on integer variables and one that does not.

	path-based tightest row aggregation				traditional cMIR aggregation				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
b4-10b	91	13936.59	0.68	2.39	33	13757.01	0.18	0.88	0.502796
b4-12b	147	15850.98	0.56	4.80	41	15609.45	0.14	3.02	0.420779
gesa2_o	230	25774034.47	0.98	0.55	156	25681053.15	0.67	0.39	0.306498
swath	66	373.88	0.30	45.89	36	334.50	0.00	15.22	0.296334
b4-20b	255	22482.45	0.33	22.08	74	22137.60	0.07	9.30	0.264100
rgn	76	81.80	0.99	0.17	78	75.30	0.79	0.17	0.194524
multiA	38	3564.64	0.23	0.30	7	3512.78	0.04	0.09	0.189548
clorox	153	17312.83	0.81	0.88	163	13571.66	0.63	0.91	0.181990
b4-10	153	13256.06	0.31	0.86	119	13114.41	0.19	0.80	0.122587
momentum1	346	84003.68	0.31	52.14	344	79682.72	0.19	42.61	0.118871
dano3_3	76	576.25	0.15	10.67	34	576.24	0.05	5.11	0.105711
b4-12	200	14567.51	0.20	1.11	119	14384.72	0.11	0.86	0.094882
gesa3_o	170	27954866.51	0.77	0.56	123	27940138.60	0.68	0.34	0.093564
arki001	233	7579847.42	0.20	7.12	168	7579764.21	0.14	5.83	0.068586
dano3_4	119	576.27	0.17	11.97	74	576.25	0.11	9.36	0.065622
a1c1s1	698	5378.64	0.42	3.09	612	4717.40	0.35	3.44	0.062940
vpm5	107	3002.68	0.58	0.64	80	3002.60	0.52	0.55	0.061708
dcmulti	104	186280.87	0.55	0.33	85	186068.45	0.50	0.33	0.050497
rentacar	21	29274325.20	0.24	0.53	21	29213967.24	0.20	0.84	0.042256
mod011	289	-59384558.73	0.36	1.98	268	-59694755.74	0.32	2.05	0.041013
bell3a	17	870990.98	0.39	0.08	11	870587.05	0.36	0.06	0.032950
dano3_5	175	576.31	0.11	15.17	104	576.29	0.08	11.78	0.030867
seymour1	7	404.13	0.04	2.39	2	403.93	0.01	1.44	0.028544
bell5	31	8928247.67	0.89	0.09	24	8918498.30	0.87	0.08	0.027234
momentum2	1138	10804.76	0.70	125.58	471	10701.10	0.68	105.50	0.020369
vpm2a	114	13.04	0.76	0.20	91	12.99	0.74	0.19	0.018230
pp08a	165	7153.35	0.96	0.20	158	7072.93	0.94	0.20	0.017476
gesa2	145	25775745.36	0.99	0.45	155	25770446.16	0.97	0.47	0.017468
gesa3	116	27963866.98	0.83	0.53	119	27961143.18	0.81	0.52	0.017304
neos7	257	672282.07	0.87	1.23	233	666986.63	0.85	1.06	0.014328
multiF	127	2061.29	0.45	0.20	121	2052.09	0.44	0.20	0.013692
bc1	63	2.61	0.72	118.50	60	2.58	0.70	138.52	0.010943
multiE	87	2289.01	0.27	0.22	77	2283.79	0.26	0.20	0.009039
vpm2	133	12.94	0.79	0.20	124	12.91	0.78	0.16	0.007857
msc98-ip	411	19564697.17	0.14	23.80	338	19562230.43	0.13	11.78	0.007744
ches1	39	71.03	0.48	0.09	31	70.99	0.47	0.08	0.006148
neos3	106	-5998.04	0.08	9.67	88	-6039.37	0.08	7.44	0.005956
m20-75-3	418	-53688.55	0.73	18.14	436	-53741.39	0.73	16.95	0.005554
ches5	43	-7370.94	0.74	0.16	42	-7371.47	0.74	0.17	0.004912
m20-75-2	404	-52198.38	0.81	27.59	402	-52242.91	0.80	20.03	0.004607
tr24-15	765	136359.52	1.00	1.38	729	135878.48	0.99	1.38	0.003983
modglob	112	20684799.25	0.82	0.12	107	20683613.38	0.82	0.12	0.003831
con-24	277	17998.19	0.48	0.92	271	17940.73	0.48	0.97	0.003825
timtab2	348	313976.30	0.22	0.75	345	310289.76	0.22	0.73	0.003515
m20-75-4	353	-54733.76	0.78	21.64	358	-54754.30	0.78	18.77	0.002309
egout	17	568.10	1.00	0.00	23	567.75	1.00	0.03	0.001743
ran12x21	126	3451.51	0.58	0.19	121	3450.71	0.58	0.20	0.001572
tr24-30	984	238400.65	0.81	0.92	984	238058.93	0.81	0.98	0.001296
tr12-15	391	73847.99	0.99	0.62	369	73775.51	0.99	0.64	0.001252
tr12-30	844	130176.42	1.00	1.53	829	130040.16	1.00	1.72	0.001211
tr6-15	206	37255.36	0.98	0.30	201	37222.62	0.98	0.33	0.001086
tr6-30	337	60964.39	0.98	0.53	338	60913.42	0.98	0.67	0.001023
ran13x13	120	3024.25	0.59	0.19	110	3023.70	0.59	0.14	0.000975
multiB	54	3627.87	0.08	0.28	34	3627.54	0.08	0.30	0.000895
timtab1	215	245864.72	0.30	0.34	240	245606.81	0.29	0.41	0.000350
dano3mip	595	576.65	0.00	32.55	537	576.64	0.00	28.26	0.000090
roll3000	161	12120.94	0.57	3.86	161	12120.82	0.57	3.91	0.000070
atlanta-ip	133	81.25	0.00	12.45	80	81.25	0.00	10.41	0.000025
...									
ran10x26	106	4078.28	0.54	0.16	99	4078.37	0.54	0.14	-0.000234
bienst2	130	14.94	0.07	0.62	138	14.96	0.08	0.67	-0.000451
bienst1	103	14.09	0.07	0.66	106	14.11	0.07	0.67	-0.000612
BASF6-10	154	20920.75	0.20	2.05	163	20921.04	0.20	2.08	-0.000661
danoint	148	62.71	0.03	0.64	129	62.72	0.03	0.64	-0.000860
khb05250	25	96428245.44	0.05	0.17	22	96439441.74	0.05	0.16	-0.001016
BASF6-5	162	11801.25	0.19	1.20	163	11802.02	0.19	1.17	-0.002288
m20-75-1	373	-51261.82	0.89	21.09	390	-51240.89	0.90	17.94	-0.002370
m20-75-5	442	-53167.89	0.80	31.03	460	-53145.20	0.80	29.27	-0.002472
set1ch	410	54348.26	0.99	0.45	408	54435.31	1.00	0.45	-0.003864
neos2	87	-4311.53	0.08	6.45	61	-4287.63	0.08	5.92	-0.004619
aflow40b	392	1082.00	0.47	18.56	344	1083.00	0.48	14.44	-0.006160
multiC	46	1490.53	0.08	0.24	47	1495.81	0.09	0.25	-0.008163
multiD	60	3821.19	0.06	0.39	47	3842.88	0.07	0.62	-0.008978
Con-12	181	4580.79	0.51	0.45	149	4636.91	0.52	0.39	-0.009066
pp08aCUTS	123	7159.63	0.90	0.25	113	7177.39	0.91	0.25	-0.009503
fixnet6	82	3662.77	0.88	0.33	91	3690.11	0.89	0.33	-0.009830
momentum3	1203	91952.39	0.00	1844.53	2744	94431.74	0.01	2452.23	-0.014331
aflow30a	226	1074.00	0.52	2.45	197	1077.00	0.54	2.06	-0.017159
lrn	357	44307174.57	0.88	10.86	347	44353913.61	0.91	10.95	-0.032143

Table D.16.: Comparison between the two aggregation strategies for the flow path cut generator.

	extended network inequalities				simple network inequalities				
name	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	Δ
clorox	153	17312.83	0.81	0.88	161	13759.84	0.64	0.89	0.172836
pp08aCUTS	123	7159.63	0.90	0.25	119	7093.05	0.86	0.27	0.035616
b4-20b	255	22482.45	0.33	22.08	254	22465.63	0.32	22.41	0.012878
multiF	127	2061.29	0.45	0.20	125	2056.00	0.45	0.20	0.007872
b4-12	200	14567.51	0.20	1.11	196	14558.36	0.20	1.08	0.004748
dano3_5	175	576.31	0.11	15.17	163	576.31	0.11	15.05	0.002765
set1ch	410	54348.26	0.99	0.45	414	54300.28	0.99	0.45	0.002130
tr24-30	984	238400.65	0.81	0.92	984	238004.34	0.81	0.92	0.001503
BASF6-10	154	20920.75	0.20	2.05	150	20920.28	0.20	2.02	0.001078
tr6-15	206	37255.36	0.98	0.30	208	37223.75	0.98	0.30	0.001048
multiE	87	2289.01	0.27	0.22	77	2288.79	0.27	0.17	0.000395
tr24-15	765	136359.52	1.00	1.38	725	136325.95	1.00	1.22	0.000278
danoint	148	62.71	0.03	0.64	140	62.71	0.03	0.67	0.000262
tr12-30	844	130176.42	1.00	1.53	840	130156.08	1.00	1.47	0.000181
dano3mip	595	576.65	0.00	32.55	579	576.63	0.00	30.86	0.000162
tr12-15	391	73847.99	0.99	0.62	375	73844.23	0.99	0.56	0.000065
neos7	257	672282.07	0.87	1.23	255	672276.71	0.87	1.20	0.000015
multiB	54	3627.87	0.08	0.28	53	3627.87	0.08	0.30	0.000003
...									
bienst2	130	14.94	0.07	0.62	123	14.94	0.07	0.67	-0.000012
tr6-30	337	60964.39	0.98	0.53	344	60971.80	0.98	0.61	-0.000149
lrn	357	44307174.57	0.88	10.86	369	44307469.73	0.88	10.73	-0.000203
bienst1	103	14.09	0.07	0.66	103	14.11	0.07	0.66	-0.000484
Con-12	181	4580.79	0.51	0.45	175	4588.44	0.51	0.44	-0.001236
con-24	277	17998.19	0.48	0.92	277	18017.06	0.48	0.92	-0.001256
pp08a	165	7153.35	0.96	0.20	165	7169.55	0.96	0.20	-0.003521
multiD	60	3821.19	0.06	0.39	61	3831.49	0.07	0.42	-0.004264
momentum3	1203	91952.39	0.00	1844.53	3699	93005.09	0.01	2231.22	-0.006085
a1c1s1	698	5378.64	0.42	3.09	720	5467.90	0.43	3.11	-0.008496
dano3_3	76	576.25	0.15	10.67	59	576.25	0.17	10.64	-0.012031
multiC	46	1490.53	0.08	0.24	55	1499.43	0.10	0.25	-0.013767
aflow30a	226	1074.00	0.52	2.45	234	1077.00	0.54	2.31	-0.017159
dano3_4	119	576.27	0.17	11.97	120	576.27	0.19	12.39	-0.017415
b4-10	153	13256.06	0.31	0.86	164	13296.58	0.35	0.86	-0.035060

Table D.17.: Comparison between the simple and the extended network inequalities.

name	flow path cuts				cMIR cuts				Δ
	cuts	xLP	gap closed	time (s)	cuts	xLP	gap closed	time (s)	
tr12-30	811	130174.22	1.00	0.98	768	114684.92	0.86	0.76	0.137717
tr6-30	369	60895.17	0.98	0.39	318	54863.91	0.86	0.31	0.121002
tr12-15	384	73846.20	0.99	0.42	366	67220.49	0.87	0.33	0.114467
tr24-15	752	136326.74	1.00	1.03	667	123906.69	0.90	0.78	0.102829
set1ch	412	54036.97	0.98	0.26	402	51762.61	0.88	0.31	0.100948
dano3_3	72	576.25	0.13	10.05	16	576.24	0.04	4.00	0.095630
tr6-15	223	37218.74	0.98	0.17	200	34865.27	0.91	0.14	0.078047
dano3_4	118	576.27	0.16	11.77	64	576.25	0.11	9.34	0.057193
multiF	129	2061.33	0.45	0.12	125	2036.01	0.42	0.12	0.037675
tr24-30	984	237715.88	0.81	0.72	984	227902.75	0.77	0.66	0.037205
con-24	247	17843.81	0.47	0.59	233	17428.04	0.44	0.53	0.027677
aflow30a	210	1074.00	0.52	0.78	174	1070.00	0.50	0.67	0.022879
b4-20b	274	22415.89	0.28	22.08	247	22398.81	0.27	17.67	0.013081
aflow40b	295	1080.00	0.46	6.41	269	1078.00	0.45	5.59	0.012320
momentum3	3426	96744.88	0.03	4070.80	3219	94616.58	0.02	3832.94	0.012302
dano3_5	158	576.30	0.10	14.70	89	576.30	0.09	12.05	0.012023
multiE	83	2290.30	0.27	0.14	78	2284.15	0.26	0.14	0.010656
clorox	167	20912.45	0.99	0.77	165	20708.61	0.98	0.61	0.009916
multiB	36	3627.81	0.08	0.24	33	3624.47	0.07	0.20	0.009125
BASF6-10	160	20919.91	0.20	1.89	150	20916.67	0.19	1.75	0.007476
danoint	152	62.72	0.03	0.55	105	62.69	0.02	0.39	0.007366
egout	21	568.10	1.00	0.00	14	566.94	0.99	0.01	0.005754
dcmulti	97	186287.49	0.55	0.25	96	186264.76	0.54	0.22	0.005404
Con-12	160	4581.22	0.51	0.28	157	4549.96	0.51	0.25	0.005051
dano3mip	595	576.66	0.00	30.02	452	576.55	0.00	22.70	0.001001
multiA	35	3562.62	0.22	0.27	34	3562.43	0.22	0.26	0.000681
bienst2	122	14.93	0.07	0.55	98	14.91	0.07	0.45	0.000487
momentum2	955	10752.84	0.69	123.83	1266	10750.92	0.69	114.74	0.000377
b4-10	154	13266.53	0.32	0.74	161	13266.11	0.32	0.74	0.000362
bienst1	103	14.05	0.07	0.58	75	14.05	0.07	0.47	0.000085
...									
multiC	50	1488.77	0.08	0.22	48	1489.21	0.08	0.22	-0.000681
BASF6-5	151	11800.56	0.19	1.08	147	11801.17	0.19	0.98	-0.001819
pp08a	169	7147.45	0.96	0.12	164	7161.51	0.96	0.11	-0.003054
b4-12	222	14633.60	0.24	0.97	212	14639.68	0.24	0.86	-0.003156
multiD	75	3827.76	0.06	0.36	31	3836.45	0.07	0.25	-0.003599
b4-12b	148	15812.24	0.49	4.53	143	15816.09	0.50	3.78	-0.006712
modglob	105	20677297.96	0.80	0.09	106	20679686.42	0.80	0.09	-0.007716
pp08aCUTS	125	7151.91	0.89	0.19	117	7189.71	0.91	0.17	-0.020217

Table D.18.: Comparison between the flow path cut generator and the cMIR cut generator, both without cuts out of cuts.

name	uPMC cuts	uPMC xLP	uPMC gap closed	uPMC time (s)	cMIR cuts cuts	cMIR cuts xLP	cMIR cuts gap closed	cMIR cuts time (s)	Δ
liu	567	560.00	0.28	0.61	421	346.00	0.00	0.17	0.282322
tr12-30	799	129173.26	0.99	0.97	768	114684.92	0.86	0.76	0.128817
tr12-15	385	73493.88	0.98	0.42	366	67220.49	0.87	0.33	0.108380
tr6-30	351	60139.65	0.97	0.38	318	54863.91	0.86	0.31	0.105844
set1ch	419	54081.80	0.98	0.28	402	51762.61	0.88	0.31	0.102937
tr24-15	715	134189.83	0.98	0.97	667	123906.69	0.90	0.78	0.085137
ches1	28	71.08	0.49	0.09	31	70.57	0.41	0.08	0.080390
tr6-15	195	36642.48	0.96	0.17	200	34865.27	0.91	0.14	0.058937
tr24-30	984	237723.44	0.81	0.73	984	227902.75	0.77	0.66	0.037234
multiF	126	2060.16	0.45	0.14	125	2036.01	0.42	0.12	0.035930
con-24	244	17841.49	0.47	0.61	233	17428.04	0.44	0.53	0.027522
momentum3	3227	99323.51	0.04	4093.14	3219	94616.58	0.02	3832.94	0.027207
dano3_4	70	576.26	0.13	10.48	64	576.25	0.11	9.34	0.020890
mod011	318	-59411387.70	0.36	2.02	317	-59493500.53	0.35	1.77	0.010857
neos2	119	-4177.08	0.10	5.73	121	-4223.46	0.10	5.72	0.008966
msc98-ip	360	19564147.14	0.14	19.17	349	19561947.01	0.13	18.67	0.006907
Con-12	158	4577.42	0.51	0.30	157	4549.96	0.51	0.25	0.004437
multiB	39	3626.04	0.07	0.23	33	3624.47	0.07	0.20	0.004298
ran13x13	117	3011.97	0.57	0.12	112	3009.58	0.57	0.11	0.004256
multiD	74	3846.51	0.07	0.38	31	3836.45	0.07	0.25	0.004164
multiE	80	2286.47	0.27	0.16	78	2284.15	0.26	0.14	0.004010
dano3_5	92	576.30	0.09	11.89	89	576.30	0.09	12.05	0.002572
pp08a	161	7167.00	0.96	0.11	164	7161.51	0.96	0.11	0.001193
dano3_3	15	576.24	0.04	4.27	16	576.24	0.04	4.00	0.000802
b4-20b	246	22399.73	0.27	19.64	247	22398.81	0.27	17.67	0.000710
BASF6-10	156	20916.91	0.19	1.95	150	20916.67	0.19	1.75	0.000557
danoint	108	62.70	0.02	0.44	105	62.69	0.02	0.39	0.000432
dano3mip	497	576.57	0.00	25.30	452	576.55	0.00	22.70	0.000165
multiC	49	1489.29	0.08	0.24	48	1489.21	0.08	0.22	0.000113
bienst2	101	14.91	0.07	0.52	98	14.91	0.07	0.45	0.000096
...									
atlanta-ip	121	81.25	0.00	12.88	116	81.25	0.00	12.27	-0.000027
momentum2	1176	10750.40	0.69	116.03	1266	10750.92	0.69	114.74	-0.000103
bienst1	76	14.04	0.07	0.53	75	14.05	0.07	0.47	-0.000155
vpm2	131	12.93	0.79	0.11	129	12.93	0.79	0.09	-0.000636
b4-12	228	14637.91	0.24	0.97	212	14639.68	0.24	0.86	-0.000921
modglob	117	20679225.61	0.80	0.09	106	20679686.42	0.80	0.09	-0.001489
a1c1s1	561	4766.83	0.36	2.42	557	4783.40	0.36	2.09	-0.001577
ches5	44	-7371.12	0.74	0.16	49	-7370.80	0.74	0.22	-0.002927
neos3	113	-5952.88	0.09	7.02	140	-5929.39	0.09	7.55	-0.003386
b4-10	147	13261.56	0.32	0.75	161	13266.11	0.32	0.74	-0.003943
aflow30a	164	1069.00	0.49	0.69	174	1070.00	0.50	0.67	-0.005720
arki001	155	7579802.85	0.17	6.84	155	7579814.00	0.18	6.30	-0.009195
b4-12b	147	15810.19	0.49	4.23	143	15816.09	0.50	3.78	-0.010277
vpm5	90	3002.64	0.55	0.53	99	3002.66	0.57	0.50	-0.013113
pp08aCUTS	110	7144.69	0.89	0.20	117	7189.71	0.91	0.17	-0.024083
clorox	158	16700.85	0.78	0.78	165	20708.61	0.98	0.61	-0.194959

Table D.19.: Comparison between the uPMC generator and the cMIR cut generator, both without cuts out of cuts.

name	cPMC cuts	cPMC xLP	cPMC gap closed	cPMC time (s)	cMIR cuts	cMIR xLP	cMIR gap closed	cMIR time (s)	Δ
tr12-30	799	129395.30	0.99	1.00	768	114684.92	0.86	0.76	0.130792
tr6-30	356	60748.41	0.98	0.39	318	54863.91	0.86	0.31	0.118057
set1ch	400	54341.00	0.99	0.28	402	51762.61	0.88	0.31	0.114442
tr12-15	368	73405.00	0.98	0.44	366	67220.49	0.87	0.33	0.106844
tr24-15	697	135301.35	0.99	1.05	667	123906.69	0.90	0.78	0.094340
tr6-15	201	36819.35	0.97	0.17	200	34865.27	0.91	0.14	0.064803
tr24-30	984	238023.94	0.81	0.75	984	227902.75	0.77	0.66	0.038373
multiF	129	2059.52	0.45	0.14	125	2036.01	0.42	0.12	0.034983
con-24	245	17879.98	0.47	0.64	233	17428.04	0.44	0.53	0.030085
b4-20b	248	22421.05	0.28	22.00	247	22398.81	0.27	17.67	0.017034
multiD	75	3874.12	0.08	0.36	31	3836.45	0.07	0.25	0.015591
mod011	310	-59415895.75	0.36	2.05	317	-59493500.53	0.35	1.77	0.010261
BASF6-10	135	20920.99	0.20	1.95	150	20916.67	0.19	1.75	0.009962
clorox	158	20910.49	0.99	0.78	165	20708.61	0.98	0.61	0.009820
multiB	42	3627.84	0.08	0.27	33	3624.47	0.07	0.20	0.009211
neos7	237	667722.35	0.85	1.19	248	664469.73	0.84	0.99	0.008801
b4-12b	152	15820.64	0.51	4.73	143	15816.09	0.50	3.78	0.007934
dano3_5	99	576.30	0.10	13.30	89	576.30	0.09	12.05	0.006702
Con-12	149	4589.83	0.51	0.31	157	4549.96	0.51	0.25	0.006442
aflow40b	276	1079.00	0.45	5.67	269	1078.00	0.45	5.59	0.006160
pp08a	168	7187.99	0.96	0.11	164	7161.51	0.96	0.11	0.005756
aflow30a	180	1071.00	0.50	0.66	174	1070.00	0.50	0.67	0.005720
vpm5	99	3002.66	0.57	0.58	99	3002.66	0.57	0.50	0.005672
khb05250	17	96112735.68	0.02	0.12	16	96070104.35	0.01	0.12	0.003868
b4-10	173	13269.89	0.32	0.81	161	13266.11	0.32	0.74	0.003271
multiC	50	1491.01	0.08	0.25	48	1489.21	0.08	0.22	0.002787
timtab2	340	312451.59	0.22	0.53	325	309603.97	0.22	0.39	0.002715
vpm2a	102	13.03	0.76	0.11	102	13.02	0.75	0.09	0.002610
momentum2	1178	10763.72	0.70	120.94	1266	10750.92	0.69	114.74	0.002515
neos3	140	-5916.41	0.09	7.64	140	-5929.39	0.09	7.55	0.001869
a1c1s1	563	4798.96	0.36	2.42	557	4783.40	0.36	2.09	0.001481
dano3_3	20	576.24	0.04	5.72	16	576.24	0.04	4.00	0.001459
ran13x13	115	3010.36	0.57	0.11	112	3009.58	0.57	0.11	0.001385
lrn	280	44296388.06	0.87	10.03	264	44295040.60	0.87	9.17	0.000927
vpm2	121	12.93	0.79	0.11	129	12.93	0.79	0.09	0.000382
ran12x21	123	3448.69	0.58	0.16	123	3448.51	0.57	0.14	0.000365
atlanta-ip	125	81.26	0.00	15.52	116	81.25	0.00	12.27	0.000347
danoint	109	62.69	0.02	0.47	105	62.69	0.02	0.39	0.000322
bienst2	107	14.92	0.07	0.53	98	14.91	0.07	0.45	0.000299
dano3mip	470	576.58	0.00	26.39	452	576.55	0.00	22.70	0.000290
bell5	21	8926098.24	0.89	0.05	20	8926029.48	0.89	0.05	0.000192
fixnet6	67	3646.91	0.88	0.19	66	3646.79	0.88	0.17	0.000040
...									
bienst1	77	14.05	0.07	0.55	75	14.05	0.07	0.47	-0.000025
dano3_4	47	576.25	0.11	8.52	64	576.25	0.11	9.34	-0.000179
neos2	116	-4227.31	0.09	5.62	121	-4223.46	0.10	5.72	-0.000744
BASF6-5	144	11800.86	0.19	1.09	147	11801.17	0.19	0.98	-0.000920
multiE	74	2283.39	0.26	0.14	78	2284.15	0.26	0.14	-0.001324
ran10x26	108	4069.62	0.51	0.12	104	4070.56	0.52	0.12	-0.002267
ches2	45	-2891.66	0.11	0.11	41	-2891.65	0.11	0.08	-0.002455
ches5	46	-7371.11	0.74	0.17	49	-7370.80	0.74	0.22	-0.002841
modglob	115	20678787.37	0.80	0.11	106	20679686.42	0.80	0.09	-0.002904
b4-10b	66	13891.12	0.55	2.30	71	13892.29	0.56	1.88	-0.003279
timtab1	199	242168.97	0.29	0.17	200	245218.44	0.29	0.19	-0.004143
ches1	29	70.47	0.39	0.09	31	70.57	0.41	0.08	-0.015329
momentum3	1432	91952.39	0.00	2229.78	3219	94616.58	0.02	3832.94	-0.015400
bell3a	14	870493.10	0.35	0.06	15	870793.00	0.38	0.05	-0.024464
pp08aCUTS	115	7125.63	0.88	0.20	117	7189.71	0.91	0.17	-0.034275
b4-12	217	14561.98	0.20	1.01	212	14639.68	0.24	0.86	-0.040333

Table D.20.: Comparison between the cPMC generator and the cMIR cut generator, both without cuts out of cuts.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.25	924.0000	235	0.04	Y
30_05_100	0	9.0000	13.95	14.0000	180729	35.71%	N
30_95_100	1	3.0000	16.94	3.0000	3897	7.36	Y
30_95_98	0	12.0000	12.20	13.0000	85474	7.69%	N
a1c1s1	776	6135.4291	3.61	11837.6401	609070	41.29%	N
acc0	7	0.0000	0.41	0.0000	236	0.06	Y
acc1	14	0.0000	0.97	0.0000	1076	0.33	Y
acc2	9	0.0000	0.70	0.0000	4454	1.97	Y
acc3	0	0.0000	0.31	0.0000	4630	7.36	Y
acc4	0	0.0000	0.33	-	22261	-	N
acc5	0	0.0000	1.25	0.0000	3664	24.96	Y
aflow30a	228	1079.0000	1.63	1158.0000	11194	0.60	Y
aflow40b	424	1087.0000	26.30	1181.0000	128074	4.74%	N
air03	2	338864.2500	0.12	340160.0000	0	0.01	Y
air04	0	55536.0000	2.39	56138.0000	5473	2.45	Y
air05	0	25878.0000	0.86	26374.0000	16466	3.58	Y
arki001	140	7579880.1818	15.25	-	1189	-	N
atlanta-ip	969	81.3034	61.40	-	372	-	N
b4-10	189	13377.1466	2.20	14050.8397	34951	1.24	Y
b4-10b	132	13979.7652	4.33	14050.8397	107	0.08	Y
b4-12	290	14724.9099	1.89	-	1214778	-	N
b4-12b	205	15834.7578	9.17	16103.8837	2749	0.70	Y
b4-20b	334	22516.3558	34.30	23388.6517	57674	1.45%	N
BASF6-10	219	20962.9041	2.69	21267.5689	77325	11.33	Y
BASF6-5	210	11898.6277	1.56	12071.5772	25854	3.12	Y
bc1	67	2.5955	115.04	3.3663	10412	3.31%	N
bell3a	16	873196.5787	0.06	878430.3160	45716	0.13	Y
bell5	29	8922311.1807	0.06	8966406.4915	2145218	6.36	Y
bienst1	119	14.0998	0.75	46.7500	32544	2.15	Y
bienst2	130	14.9447	0.75	54.6000	272008	14.69	Y
binkar10_1	95	6702.1432	0.67	6742.2000	318139	9.20	Y
blend2	37	7.0952	0.59	7.5990	2997	0.04	Y
cap6000	7	-2451535.0000	0.16	-2451377.0000	37770	4.39	Y
ches1	102	73.8056	0.58	74.3405	20	0.02	Y
ches2	66	-2891.6536	0.19	-2889.5569	2758190	32.47	Y
ches3	30	-1303896.9248	0.14	-1303896.9248	20	0.00	Y
ches4	32	-647403.5167	0.03	-647403.5167	13	0.00	Y
ches5	78	-7370.5310	0.09	-7342.8188	7948	0.14	Y
clorox	190	17326.7046	1.31	21217.8144	114	0.04	Y
Con-12	248	4618.4475	0.78	7593.3400	177576	4.03	Y
con-24	289	18181.3246	0.59	25804.9600	1871007	1.55%	N
dano3mip	628	576.6893	39.96	738.5385	4060	21.9%	N
dano3_3	103	576.2571	12.86	576.3964	12	1.68	Y
dano3_4	173	576.2714	10.36	576.4352	16	2.45	Y
dano3_5	306	576.3352	15.89	576.9249	216	5.74	Y
danoint	153	62.7198	0.81	65.6667	419940	4.23%	N
dcmulti	172	187529.5533	0.51	188182.0000	142	0.01	Y
disktom	0	-5000.0000	0.50	-	136336	-	N
dlsp	31	375.3100	0.45	613.0000	103209	2.18	Y
ds	0	57.2346	5.41	-	2222	-	N
dsbmip	102	-305.1982	0.12	-305.1982	59	0.02	Y
egout	22	567.9932	0.03	568.1007	0	0.00	Y
enigma	2	0.0000	0.00	0.0000	21176	0.04	Y
fast0507	2	173.0000	5.52	176.0000	12494	1.7%	N
fiber	84	388306.7843	0.70	405935.1800	164	0.02	Y
fixnet6	206	3807.4818	1.94	3983.0000	128	0.05	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0.00	Y
gen	44	112312.9529	0.09	112313.3627	0	0.00	Y
gesa2	166	25771293.5544	0.64	25779856.3717	340	0.02	Y
gesa2_o	209	25775432.4183	0.73	25779856.3717	230	0.02	Y
gesa3	192	27973351.6108	1.02	27991042.6484	48	0.03	Y
gesa3_o	234	27963406.7044	1.02	27991042.6484	93	0.03	Y
glass4	63	800002400.0000	0.11	1675016325.0000	4988390	52.24%	N
gt2	31	20726.0000	0.03	21166.0000	193	0.00	Y
harp2	103	-74231352.0000	2.33	-73899798.0000	1967668	31.49	Y
khb05250	124	106915722.2610	0.62	106940226.0000	22	0.02	Y
l152lav	0	4657.0000	0.16	4722.0000	10637	0.22	Y
liu	691	560.0000	0.69	1412.0000	1537631	60.34%	N
lrn	900	44546780.6458	12.61	44705245.0050	287094	0.1%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
lseu	34	1034.0000	0.14	1120.0000	785	0.01	Y
m20-75-1	389	-51161.8537	6.38	-49113.0000	300886	3.35%	N
m20-75-2	605	-52005.4722	64.41	-50322.0000	68392	16.78	Y
m20-75-3	653	-53238.0531	121.61	-51158.0000	262374	2.79%	N
m20-75-4	419	-54693.4443	99.17	-52752.0000	230556	49.40	Y
m20-75-5	523	-53017.6993	16.32	-51349.0000	163488	33.92	Y
manna81	0	-13297.0000	0.41	-13163.0000	873785	1.02%	N
markshare1	3	0.0000	0.03	8.0000	9999999	100%	N
markshare1_1	7	0.0000	0.02	0.0000	823388	1.70	Y
markshare2	6	0.0000	0.00	17.0000	9999999	100%	N
markshare2_1	11	0.0000	0.02	0.0000	7287843	19.98	Y
mas74	25	10583.2346	0.91	11801.1857	6718997	3.51%	N
mas76	23	39010.8237	0.83	40005.0541	1159854	8.10	Y
misc03	0	1910.0000	0.01	3360.0000	603	0.00	Y
misc06	29	12846.2683	0.12	12851.0763	19	0.02	Y
misc07	0	1415.0000	0.03	2810.0000	37474	0.56	Y
mitre	970	115107.0000	5.81	115155.0000	2488	0.35	Y
mkc	346	-605.9688	8.36	-511.7520	599537	18.2%	N
mod008	48	304.0000	0.69	307.0000	3398	0.07	Y
mod010	5	6535.0000	0.59	6548.0000	18	0.01	Y
mod011	823	-56654368.7838	4.67	-54558535.0142	4280	1.44	Y
modglob	163	20727876.2786	0.31	20740508.0863	62	0.01	Y
momentum1	690	96249.1959	50.50	-	95	-	N
momentum2	1479	11697.6124	143.92	-	59	-	N
momentum3	2667	91975.3222	1186.44	-	0	-	N
msc98-ip	624	19702877.0058	25.50	21287346.0059	4727	7.44%	N
multiA	97	3569.8677	0.50	3774.7600	48614	1.23	Y
multiB	108	3628.6516	1.22	3995.5200	1942745	8.79%	N
multiC	101	1501.6364	0.62	2088.4200	1986760	24.42%	N
multiD	83	3808.4903	0.08	6021.6167	1694764	36.15%	N
multiE	247	2299.8939	0.59	2710.5925	2408830	13.92%	N
multiF	219	2070.2834	0.56	2428.9300	2506812	13.39%	N
mzzv11	103	-22689.0000	55.34	-19040.0000	45645	17.66%	N
mzzv42z	75	-21450.0000	56.30	-19308.0000	8833	9.03%	N
neos1	125	7.0000	0.45	19.0000	1890822	57.89%	N
neos10	81	-1182.0000	214.25	-1135.0000	40	3.68	Y
neos11	10	6.0000	0.75	9.0000	30044	16.48	Y
neos12	6	9.4116	0.75	13.0000	11805	16.22%	N
neos13	6	-126.1784	284.44	-87.0062	121172	45.02%	N
neos2	110	-3986.4225	11.64	454.8647	132551	11.32	Y
neos20	357	-474.8940	0.95	-434.0000	24823	1.10	Y
neos21	0	3.0000	0.06	7.0000	30130	2.40	Y
neos22	242	779500.7143	0.36	779715.0000	32	0.04	Y
neos23	136	59.3098	1.05	137.0000	3302625	38.69%	N
neos3	174	-5674.9374	15.78	368.8428	788772	304.2%	N
neos4	0	-49463016984.6474	2.53	-48603440750.5898	1305	0.63	Y
neos5	0	13.0000	0.02	15.0000	7939707	3.33%	N
neos6	4	83.0000	3.30	83.0000	4721	7.37	Y
neos648910	365	16.0000	0.52	32.0000	322488	7.78	Y
neos671048	4	2999.0000	2.74	5001.0000	15868	13.57	Y
neos7	290	688512.2482	1.55	721934.0000	752317	0.48%	N
neos8	23	-3725.0000	178.30	-3719.0000	0	2.99	Y
neos9	35	794.0000	13.69	798.0000	13003	0.5%	N
net12	467	78.0000	14.86	-	40776	-	N
noswot	14	-43.0000	0.03	-40.0000	9618950	7.5%	N
nsrand-ipx	305	50187.0000	0.92	51680.0000	165481	2.85%	N
nug08	0	204.0000	0.22	214.0000	151	0.10	Y
nw04	0	16311.0000	1.02	16862.0000	1638	1.13	Y
opt1217	31	-19.3221	0.06	-16.0000	5050765	20.76%	N
p0033	22	2942.0000	0.03	3089.0000	85	0.00	Y
p0201	8	7125.0000	0.39	7615.0000	1099	0.02	Y
p0282	109	255708.0000	0.20	258411.0000	713	0.01	Y
p0548	170	8691.0000	0.08	8691.0000	0	0.01	Y
p2756	250	3121.0000	0.95	3124.0000	422	0.06	Y
pk1	0	0.0000	0.00	11.0000	529908	1.79	Y
pp08a	230	7242.6338	0.41	7350.0000	629	0.02	Y
pp08aCUTS	143	7204.5948	0.30	7350.0000	1040	0.02	Y
prod1	137	-81.3751	0.89	-56.0000	3986086	11.27%	N
prod2	130	-85.2228	2.92	-62.0000	2609602	9.84%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
protfold	0	-41.0000	0.23	-	5	-	N
qap10	0	333.0000	0.61	358.0000	43	6.98%	N
qiu	0	-931.6389	0.06	-132.8731	15640	1.34	Y
qnet1	76	15438.7245	0.48	16029.6927	298	0.03	Y
qnet1_o	90	15624.5847	0.51	16030.9927	230	0.02	Y
ran10x26	218	4094.8122	1.06	4270.0000	40140	1.15	Y
ran12x21	264	3462.6667	1.83	3664.0000	50435	1.64	Y
ran13x13	211	3065.7297	0.81	3252.0000	34533	0.75	Y
rd-rplusc-21	208	100.0000	606.83	-	15229	-	N
rentacar	24	29274325.2003	0.69	30356760.9841	25	0.03	Y
rgn	110	81.8363	0.28	82.2000	354	0.01	Y
rgna	0	48.8000	0.02	82.2000	2504	0.00	Y
roll3000	377	11512.1280	6.12	13240.0000	478878	11.44%	N
rout	39	982.1729	0.58	1077.5600	824442	16.00	Y
set1ch	475	54528.4375	0.64	54537.7500	26	0.02	Y
seymour	8	406.0000	1.53	434.0000	74767	6.22%	N
seymour1	16	405.4473	5.45	410.7637	25409	17.99	Y
sp97ar	253	653445845.1576	5.52	672355872.3000	68001	2.72%	N
stein27	7	13.0000	0.02	18.0000	4240	0.01	Y
stein45	0	22.0000	0.01	30.0000	62605	0.34	Y
stp3d	6	481.9510	212.64	-	12	-	N
swath	138	379.9005	448.08	536.5078	469496	29.19%	N
swath2	19	334.4969	1.55	385.1997	421028	38.06	Y
swath3	19	334.4969	1.74	399.8501	659017	12.35%	N
t1717	0	134532.0000	7.66	-	2513	-	N
timtab1	270	255646.2133	0.61	792722.0000	4029935	65.78%	N
timtab2	418	381352.8194	1.41	1452023.0000	2554585	73.62%	N
tr12-15	395	73877.2001	0.78	74634.0000	16170	0.37	Y
tr12-30	859	130177.2010	0.66	130596.0000	999190	40.27	Y
tr24-15	807	136365.6881	1.19	136509.0000	28890	1.36	Y
tr24-30	984	237994.8573	1.12	294759.0000	1232296	18.52%	N
tr6-15	227	37252.9562	0.42	37721.0000	5090	0.07	Y
tr6-30	331	60965.3867	0.09	61746.0000	2515927	36.67	Y
vpm1	41	20.0000	0.05	20.0000	0	0.00	Y
vpm2	170	12.9692	0.31	13.7500	20559	0.23	Y
vpm2a	129	13.0633	0.25	13.7500	9948	0.10	Y
vpm5	134	3002.7436	0.86	3003.3000	1165	0.09	Y

Table D.21.: Results for a 1-hour test with the improved SOTA configuration.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.28	924.0000	235	0.04	Y
30_05_100	0	9.0000	13.88	14.0000	180726	35.71%	N
30_95_100	1	3.0000	16.98	3.0000	3897	7.27	Y
30_95_98	0	12.0000	12.19	13.0000	85508	7.69%	N
a1c1s1	866	6200.9933	4.83	11671.7123	606088	38.02%	N
acc0	7	0.0000	0.39	0.0000	236	0.06	Y
acc1	14	0.0000	0.95	0.0000	1076	0.33	Y
acc2	9	0.0000	0.70	0.0000	4454	1.96	Y
acc3	0	0.0000	0.33	0.0000	4630	7.33	Y
acc4	0	0.0000	0.34	-	22234	-	N
acc5	0	0.0000	1.25	0.0000	3664	25.01	Y
aflow30a	284	1083.0000	2.45	1158.0000	19673	1.19	Y
aflow40b	459	1088.0000	20.88	1179.0000	158615	4.58%	N
air03	2	338864.2500	0.12	340160.0000	0	0.01	Y
air04	0	55536.0000	2.39	56138.0000	5473	2.45	Y
air05	0	25878.0000	0.86	26374.0000	16466	3.59	Y
arki001	140	7579880.1818	34.05	-	1189	-	N
atlanta-ip	1043	0.0000	63.95	-	0	-	N
b4-10	222	13378.6745	21.44	-	1681212	-	N
b4-10b	136	13981.9765	19.28	14050.8397	216	0.35	Y
b4-12	306	14634.8138	2.83	16103.8837	1379857	1.73%	N
b4-12b	213	15854.7515	24.28	16103.8837	4053	1.05	Y
b4-20b	363	22502.1540	40.42	23358.2110	60952	1.18%	N
BASF6-10	236	20966.2195	3.20	21267.5689	168394	21.41	Y
BASF6-5	223	11899.5370	1.88	12072.3655	57571	5.68	Y
bc1	72	2.5852	121.95	3.3384	7462	44.90	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
bell3a	20	873203.1839	0.16	878430.3160	49937	0.15	Y
bell5	29	8922311.1807	0.08	8966406.4915	2145218	6.37	Y
bienst1	119	14.0998	1.03	46.7500	32544	2.17	Y
bienst2	130	14.9447	0.97	54.6000	272008	14.73	Y
binkar10_1	95	6702.1432	0.72	6742.2000	318139	9.21	Y
blend2	37	7.0952	0.59	7.5990	2997	0.04	Y
cap6000	7	-2451535.0000	0.16	-2451377.0000	37770	4.39	Y
ches1	79	73.3945	0.26	74.3405	0	0.01	Y
ches2	66	-2891.6536	0.19	-2889.5569	2758190	32.54	Y
ches3	31	-1303896.9248	0.17	-1303896.9248	6	0.01	Y
ches4	32	-647403.5167	0.03	-647403.5167	13	0.01	Y
ches5	80	-7370.4925	0.50	-7342.8188	3150	0.06	Y
clorox	209	17338.3607	1.74	21218.8920	226	0.06	Y
Con-12	211	4674.5920	0.70	7593.3400	179808	3.77	Y
con-24	295	18024.3010	0.61	25804.9600	1783643	2.63%	N
dano3mip	634	576.6712	219.66	748.9510	3322	22.99%	N
dano3_3	77	576.2520	22.22	576.3446	7	2.92	Y
dano3_4	206	576.2808	128.23	576.4352	13	5.96	Y
dano3_5	318	576.3284	140.45	576.9249	188	10.02	Y
danoint	251	62.7229	1.77	65.6667	355347	4.15%	N
dcmulti	158	187511.0373	0.70	188182.0000	132	0.02	Y
disktom	0	-5000.0000	0.47	-	136173	-	N
dlsp	34	375.3360	0.94	613.0000	60862	1.31	Y
ds	0	57.2346	5.49	-	2218	-	N
dsbmip	69	-305.1982	0.16	-305.1982	37	0.02	Y
egout	20	567.8702	0.01	568.1007	0	0.01	Y
enigma	2	0.0000	0.00	0.0000	21176	0.04	Y
fast0507	2	173.0000	5.41	176.0000	12496	1.7%	N
fiber	86	388328.4284	0.72	405935.1800	138	0.01	Y
fixnet6	253	3810.5117	2.38	3983.0000	44	0.05	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0.01	Y
gen	44	112312.9529	0.09	112313.3627	0	0.01	Y
gesa2	174	25770857.4176	0.70	25779856.3717	408	0.02	Y
gesa2_o	222	25776509.7718	0.80	25779856.3717	198	0.02	Y
gesa3	173	27967044.2789	1.16	27991042.6484	108	0.04	Y
gesa3_o	224	27959820.8972	1.01	27991042.6484	112	0.04	Y
glass4	63	800002400.0000	0.11	1675016325.0000	4985307	52.24%	N
gt2	31	20726.0000	0.02	21166.0000	193	0.01	Y
harp2	103	-74231352.0000	2.31	-73899798.0000	1967668	31.47	Y
khb05250	124	106915722.2610	3.81	106940226.0000	22	0.07	Y
l152lav	0	4657.0000	0.16	4722.0000	10637	0.22	Y
liu	708	560.0000	0.77	1514.0000	1462533	63.01%	N
lrn	940	44576193.7519	16.64	44679584.9230	285108	0.02%	N
lseu	34	1034.0000	0.12	1120.0000	785	0.01	Y
m20-75-1	389	-51161.8537	6.42	-49113.0000	300646	3.35%	N
m20-75-2	605	-52005.4722	64.94	-50322.0000	68392	16.80	Y
m20-75-3	653	-53238.0531	123.30	-51158.0000	262033	2.79%	N
m20-75-4	419	-54693.4443	100.44	-52752.0000	230556	49.52	Y
m20-75-5	523	-53017.6993	16.44	-51349.0000	163488	33.96	Y
manna81	0	-13297.0000	0.41	-13163.0000	878192	1.02%	N
markshare1	3	0.0000	0.05	8.0000	9999999	100%	N
markshare1_1	9	0.0000	0.01	0.0000	407157	0.90	Y
markshare2	6	0.0000	0.01	17.0000	9999999	100%	N
markshare2_1	13	0.0000	0.05	0.0000	6353340	18.59	Y
mas74	24	10575.3466	0.92	11801.1857	6728975	2.41%	N
mas76	23	39015.3883	0.84	40005.0541	1123402	7.45	Y
misc03	0	1910.0000	0.01	3360.0000	603	0.01	Y
misc06	24	12845.9373	0.23	12850.8607	83	0.02	Y
misc07	0	1415.0000	0.03	2810.0000	37474	0.56	Y
mitre	970	115107.0000	5.81	115155.0000	2488	0.35	Y
mkc	346	-605.9688	8.28	-511.7520	596132	18.2%	N
mod008	48	304.0000	0.70	307.0000	3398	0.07	Y
mod010	5	6535.0000	0.52	6548.0000	18	0.01	Y
mod011	869	-56571569.2127	5.83	-54558535.0142	2558	1.09	Y
modglob	147	20728719.4371	0.16	20740508.0863	30	0.01	Y
momentum1	696	96251.4843	99.75	-	438	-	N
momentum2	1430	12138.6543	254.22	-	164	-	N
momentum3	1614	92981.4344	4961.42	n.a.	0	n.a.	N
msc98-ip	582	19699455.1058	13.97	-	2399	-	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
multiA	105	3568.5128	2.98	3774.7600	44680	1.16	Y
multiB	133	3630.4493	11.03	3991.6900	1608127	8.6%	N
multiC	114	1498.5951	4.05	2083.2867	2060666	24.45%	N
multiD	75	3795.4978	0.11	5863.9045	1522775	34.44%	N
multiE	281	2303.9884	0.95	2710.5925	2118422	11.6%	N
multiF	206	2068.5267	0.66	2428.9300	3008134	11.44%	N
mzzv11	103	-22689.0000	55.12	-19040.0000	45645	17.66%	N
mzzv42z	75	-21450.0000	56.42	-19308.0000	8847	9.03%	N
neos1	125	7.0000	0.45	19.0000	1893721	57.89%	N
neos10	81	-1182.0000	215.53	-1135.0000	40	3.70	Y
neos11	10	6.0000	0.81	9.0000	30044	16.53	Y
neos12	6	9.4116	0.81	13.0000	11779	16.22%	N
neos13	6	-126.1784	331.53	-95.1452	124209	25.77%	N
neos2	111	-3965.4236	14.16	454.8647	59126	5.18	Y
neos20	363	-474.8940	0.97	-434.0000	43014	1.76	Y
neos21	0	3.0000	0.06	7.0000	30130	2.41	Y
neos22	242	779500.7143	1.56	779715.0000	36	0.17	Y
neos23	140	61.6402	1.64	137.0000	3431607	43.8%	N
neos3	179	-5665.8976	17.59	369.4102	552862	326.37%	N
neos4	0	-49463016984.6474	2.64	-48603440750.5898	1305	0.63	Y
neos5	0	13.0000	0.02	15.0000	7932201	3.33%	N
neos6	4	83.0000	3.38	83.0000	4721	7.37	Y
neos648910	336	16.0000	0.64	32.0000	345539	10.49	Y
neos671048	4	2999.0000	2.73	5001.0000	15868	13.64	Y
neos7	287	686267.8857	1.70	721934.0000	744518	1.39%	N
neos8	23	-3725.0000	180.89	-3719.0000	0	3.03	Y
neos9	35	794.0000	14.14	798.0000	12919	0.5%	N
net12	452	78.0000	16.16	-	15501	-	N
noswot	16	-43.0000	0.03	-41.0000	9999999	4.88%	N
nsrand-ipx	305	50187.0000	0.92	51680.0000	165295	2.85%	N
nug08	0	204.0000	0.22	214.0000	151	0.10	Y
nw04	0	16311.0000	1.01	16862.0000	1638	1.13	Y
opt1217	35	-19.3943	0.11	-16.0000	4958509	21.21%	N
p0033	22	2942.0000	0.73	3089.0000	85	0.04	Y
p0201	8	7125.0000	0.47	7615.0000	1099	0.02	Y
p0282	109	255708.0000	0.31	258411.0000	713	0.02	Y
p0548	170	8691.0000	0.11	8691.0000	0	0.01	Y
p2756	250	3121.0000	0.97	3124.0000	422	0.07	Y
pk1	0	0.0000	0.02	11.0000	529908	1.80	Y
pp08a	232	7241.4280	0.61	7350.0000	642	0.03	Y
pp08aCUTS	186	7216.3844	0.84	7350.0000	796	0.04	Y
prod1	137	-81.3751	0.84	-56.0000	3984028	11.27%	N
prod2	130	-85.2228	2.94	-62.0000	2609698	9.84%	N
protfold	0	-41.0000	0.22	-	5	-	N
qap10	0	0.0000	0.61	-	0	-	N
qiu	0	-931.6389	0.08	-132.8731	15640	1.35	Y
qnet1	76	15438.7245	0.49	16029.6927	298	0.03	Y
qnet1_o	90	15624.5847	0.52	16030.9927	230	0.02	Y
ran10x26	229	4092.3948	1.23	4270.0000	25203	0.72	Y
ran12x21	276	3470.7403	1.62	3664.0000	77613	2.59	Y
ran13x13	231	3057.8648	1.41	3252.0000	52406	1.14	Y
rd-rplusc-21	346	100.0000	560.92	-	20378	-	N
rentacar	27	29274325.2003	1.31	30356760.9841	31	0.04	Y
rgn	102	81.8000	0.06	82.2000	331	0.01	Y
rgna	0	48.8000	0.01	82.2000	2504	0.01	Y
roll3000	377	11512.1280	6.58	13107.0000	488551	9.51%	N
rout	39	982.1729	0.62	1077.5600	824442	15.98	Y
set1ch	495	54530.4424	0.86	54537.7500	96	0.03	Y
seymour	8	406.0000	1.53	434.0000	74627	6.22%	N
seymour1	16	405.4473	6.69	410.7637	25409	18.03	Y
sp97ar	253	653445845.1576	5.34	672355872.3000	67999	2.72%	N
stein27	7	13.0000	0.02	18.0000	4240	0.01	Y
stein45	0	22.0000	0.02	30.0000	62605	0.34	Y
stp3d	6	481.9510	212.72	-	12	-	N
swath	229	378.8363	477.08	474.9842	375831	17.74%	N
swath2	19	334.4969	1.75	385.1997	461141	42.99	Y
swath3	19	334.4969	1.72	397.8494	670490	11.71%	N
t1717	0	134532.0000	7.62	-	2499	-	N
timtab1	286	273688.4870	1.09	794975.0000	3434444	58.08%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
timtab2	423	380748.3096	2.09	1321630.0000	2403081	70.45%	N
tr12-15	423	73872.0960	1.09	74634.0000	11226	0.29	Y
tr12-30	898	129785.3311	0.06	130596.0000	1350667	0.18%	N
tr24-15	848	136365.6881	2.38	136509.0000	51922	2.28	Y
tr24-30	984	237512.8900	1.12	295150.0000	1171449	19.12%	N
tr6-15	236	37246.0120	0.53	37721.0000	7180	0.10	Y
tr6-30	381	61018.7618	0.16	61746.0000	1407454	24.39	Y
vpm1	41	20.0000	0.03	20.0000	0	0.01	Y
vpm2	180	13.0786	0.59	13.7500	10822	0.15	Y
vpm2a	140	13.0711	0.62	13.7500	9453	0.12	Y
vpm5	145	3002.7260	1.01	3003.2000	437	0.05	Y

Table D.22.: Results for a 1-hour test with the improved SOTA configuration and aggregated flow cover cuts. For the instance `momentum3`, MOPS returned an invalid result.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.25	924.0000	235	0.04	Y
30_05_100	0	9.0000	13.70	14.0000	181694	35.71%	N
30_95_100	1	3.0000	16.84	3.0000	3897	7.23	Y
30_95_98	0	12.0000	12.06	13.0000	85945	7.69%	N
a1c1s1	571	5312.4968	2.44	11889.8499	774124	49.39%	N
acc0	7	0.0000	0.38	0.0000	236	0.05	Y
acc1	14	0.0000	0.89	0.0000	1076	0.32	Y
acc2	9	0.0000	0.69	0.0000	4454	1.96	Y
acc3	0	0.0000	0.30	0.0000	4630	7.32	Y
acc4	0	0.0000	0.31	-	22306	-	N
acc5	0	0.0000	1.22	0.0000	3664	24.91	Y
aflow30a	173	1071.0000	0.67	1158.0000	26583	1.01	Y
aflow40b	319	1080.0000	10.19	1217.0000	157475	7.64%	N
air03	2	338864.2500	0.14	340160.0000	0	0.01	Y
air04	0	55536.0000	2.38	56138.0000	5473	2.44	Y
air05	0	25878.0000	0.86	26374.0000	16466	3.58	Y
arki001	120	7579832.1481	4.39	7580928.3811	963540	0.01%	N
atlanta-ip	459	81.2791	21.59	-	3392	-	N
b4-10	220	13360.8863	0.88	-	1651588	-	N
b4-10b	127	13984.5246	2.42	14050.8397	489	0.08	Y
b4-12	274	14715.7401	1.11	16103.8837	1166852	3.73%	N
b4-12b	191	15819.6140	4.42	16103.8837	3778	0.64	Y
b4-20b	326	22436.6211	28.42	23588.4117	58249	4.13%	N
BASF6-10	170	20958.1596	2.11	21267.8894	80161	10.50	Y
BASF6-5	169	11894.5315	1.17	12071.5772	53872	5.57	Y
bc1	45	2.5780	110.59	3.3384	8462	52.02	Y
bell3a	15	873196.5787	0.06	878430.3160	46285	0.13	Y
bell5	21	8918959.2124	0.03	8966406.4915	16722	0.04	Y
bienst1	101	14.0612	0.44	46.7500	36770	2.03	Y
bienst2	102	14.9164	0.52	54.6000	251744	12.03	Y
binkar10_1	88	6689.6924	0.55	6742.2000	690802	16.71	Y
blend2	31	7.8121	0.45	8.4056	3520	0.04	Y
cap6000	7	-2451535.0000	0.14	-2451377.0000	37770	4.38	Y
ches1	43	73.4626	0.08	74.3405	8	0.01	Y
ches2	58	-2891.6536	0.12	-2889.6909	3927534	43.42	Y
ches3	30	-1303896.9248	0.05	-1303896.6448	18	0.01	Y
ches4	37	-647403.5167	0.03	-647403.5167	0	0.01	Y
ches5	82	-7367.9934	0.06	-7342.8188	2564	0.04	Y
clorox	169	20745.6133	0.42	21217.8144	246	0.02	Y
Con-12	153	4553.1586	0.11	7593.3100	335184	5.10	Y
con-24	223	17560.0037	0.06	25804.9600	2062104	4.67%	N
dano3mip	474	576.5456	50.22	778.8571	4893	25.96%	N
dano3_3	18	576.2353	0.84	576.3964	9	1.69	Y
dano3_4	55	576.2526	0.83	576.4352	24	2.11	Y
dano3_5	117	576.3038	8.45	576.9249	238	5.06	Y
danoint	105	62.6937	0.50	65.6667	525515	4.29%	N
dcmulti	130	187327.6650	0.30	188182.0000	234	0.01	Y
disktom	0	-5000.0000	0.45	-	136556	-	N
dlsp	15	371.0682	0.11	613.0000	104701	2.03	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
ds	0	57.2346	5.41	-	2234	-	N
dsbmip	65	-305.1982	0.11	-305.1982	67	0.01	Y
egout	18	567.0998	0.00	568.1007	0	0.01	Y
enigma	2	0.0000	0.02	0.0000	21176	0.04	Y
fast0507	2	173.0000	5.39	176.0000	12510	1.7%	N
fiber	65	386653.3942	0.30	405935.1800	216	0.01	Y
fixnet6	140	3765.5964	0.36	3983.0000	154	0.02	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0.01	Y
gen	40	112312.9529	0.09	112313.3627	0	0.01	Y
gesa2	148	25758040.3176	0.08	25779856.3717	82	0.01	Y
gesa2_o	181	25753330.2302	0.52	25779856.3717	176	0.01	Y
gesa3	99	27952231.5903	0.09	27991042.6484	58	0.01	Y
gesa3_o	140	27940975.1567	0.12	27991042.6484	246	0.02	Y
glass4	43	800002400.0000	0.05	1650014050.0000	5650700	51.52%	N
gt2	42	20647.0000	0.03	21166.0000	691	0.01	Y
harp2	94	-74202514.0000	2.30	-73872399.4600	1853655	29.46	Y
khb05250	124	106916419.0931	0.17	106940226.0000	22	0.01	Y
l152lav	0	4657.0000	0.16	4722.0000	10637	0.22	Y
liu	527	560.0000	0.36	1362.0000	666597	58.88%	N
lrn	649	44374013.5751	9.84	44491801.5478	322135	0.09%	N
lseu	35	1036.0000	0.09	1120.0000	887	0.01	Y
m20-75-1	94	-52236.0477	9.92	-49113.0000	411546	5.21%	N
m20-75-2	97	-53447.3278	10.55	-50314.0000	383054	5.34%	N
m20-75-3	105	-54711.4710	10.36	-51102.0000	393847	6.37%	N
m20-75-4	94	-55832.7524	10.63	-52612.0000	424264	5.24%	N
m20-75-5	97	-54168.3999	10.74	-51349.0000	407443	3.37%	N
manna81	0	-13297.0000	0.38	-13163.0000	877277	1.02%	N
markshare1	4	0.0000	0.00	7.0000	9999999	100%	N
markshare1_1	6	0.0000	0.00	0.0000	68417	0.13	Y
markshare2	6	0.0000	0.00	17.0000	9999999	100%	N
markshare2_1	11	0.0000	0.00	0.0000	94199	0.27	Y
mas74	24	10580.6249	0.41	11801.1857	6664412	3.84%	N
mas76	23	39007.8454	0.38	40005.0541	881696	6.02	Y
misc03	0	1910.0000	0.02	3360.0000	603	0.01	Y
misc06	34	12846.8491	0.06	12851.0763	44	0.01	Y
misc07	0	1415.0000	0.05	2810.0000	37474	0.56	Y
mitre	962	115105.0000	5.55	115155.0000	11328	1.23	Y
mkc	360	-604.9997	11.81	-555.1300	488660	7.68%	N
mod008	56	304.0000	0.28	307.0000	117	0.02	Y
mod010	4	6535.0000	0.66	6548.0000	18	0.01	Y
mod011	774	-57246472.5262	3.30	-54558535.0142	3360	0.94	Y
modglob	146	20716487.1910	0.12	20740508.0863	782	0.01	Y
momentum1	591	96248.4612	43.22	-	434	-	N
momentum2	1037	10715.7256	93.65	-	234	-	N
momentum3	3665	95404.1737	3406.58	-	1	-	N
msc98-ip	410	19695288.0058	9.09	-	2654	-	N
multiA	67	3568.5075	0.05	3774.7600	167960	3.37	Y
multiB	49	3624.4707	0.03	3964.8800	2255565	8.16%	N
multiC	47	1487.6198	0.03	2083.2867	2523158	25.33%	N
multiD	115	3887.8291	0.51	5872.1231	1213884	32.81%	N
multiE	232	2298.8769	0.23	2718.2050	2778919	10.63%	N
multiF	148	2036.3646	0.36	2428.9300	3844994	15.74%	N
mzzv11	109	-22721.0000	74.82	-21168.0000	91393	5.48%	N
mzzv42z	71	-21450.0000	56.57	-17400.0000	5160	21.11%	N
neos1	87	7.0000	0.28	19.0000	1888010	57.89%	N
neos10	81	-1182.0000	213.50	-1135.0000	32	3.67	Y
neos11	10	6.0000	0.69	9.0000	30044	16.55	Y
neos12	6	9.4116	0.70	13.0000	11766	16.22%	N
neos13	4	-126.1784	39.16	-92.5828	150276	29.07%	N
neos2	137	-3979.8472	9.03	454.8697	146255	7.12	Y
neos20	321	-474.8940	0.73	-434.0000	25569	1.11	Y
neos21	0	3.0000	0.06	7.0000	30130	2.41	Y
neos22	195	778990.4286	1.09	779715.0000	0	0.02	Y
neos23	119	64.3292	0.81	137.0000	3478907	32.85%	N
neos3	179	-5743.4783	12.38	368.9010	1061657	261.17%	N
neos4	0	-49463016984.6474	2.41	-48603440750.5898	1305	0.62	Y
neos5	0	13.0000	0.01	15.0000	7921889	3.33%	N
neos6	4	83.0000	3.30	83.0000	4721	7.38	Y
neos648910	365	16.0000	0.48	32.0000	577114	16.93	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
neos671048	3	2999.0000	2.27	5001.0000	873	0.43	Y
neos7	278	687060.1468	1.20	721934.0000	685048	53.10	Y
neos8	23	-3725.0000	178.31	-3719.0000	0	2.99	Y
neos9	35	794.0000	12.67	798.0000	12876	0.5%	N
net12	438	78.0000	13.00	-	56779	-	N
noswot	10	-43.0000	0.03	-41.0000	9999999	4.88%	N
nsrand-ipx	301	50193.0000	1.27	52000.0000	146006	3.43%	N
nug08	0	204.0000	0.22	214.0000	151	0.10	Y
nw04	0	16311.0000	1.01	16862.0000	1638	1.13	Y
opt1217	51	-19.0809	0.05	-16.0000	4090798	19.26%	N
p0033	21	2942.0000	0.47	3089.0000	84	0.03	Y
p0201	8	7125.0000	0.45	7615.0000	1099	0.02	Y
p0282	102	255563.0000	0.17	258411.0000	460	0.01	Y
p0548	145	8675.0000	0.30	8691.0000	18	0.01	Y
p2756	256	3121.0000	1.19	3124.0000	968	0.08	Y
pk1	0	0.0000	0.00	11.0000	529908	1.78	Y
pp08a	191	7205.4549	0.16	7350.0000	808	0.02	Y
pp08aCUTS	144	7193.3648	0.25	7350.0000	993	0.02	Y
prod1	135	-81.3838	0.67	-56.0000	3174234	48.33	Y
prod2	126	-85.2231	2.39	-61.0000	2730856	20.27%	N
protfold	0	-41.0000	0.22	-	5	-	N
qap10	0	0.0000	0.58	-	0	-	N
qiu	0	-931.6389	0.05	-132.8731	15640	1.35	Y
qnet1	69	15322.9755	0.42	16029.6927	177	0.03	Y
qnet1_o	82	15348.7293	0.38	16029.6927	139	0.02	Y
ran10x26	176	4086.9607	0.30	4270.0000	25925	0.63	Y
ran12x21	176	3453.2323	0.24	3664.0000	78878	1.81	Y
ran13x13	137	3016.1864	0.14	3252.0000	61448	0.97	Y
rd-rplusc-21	364	100.0000	556.89	-	19346	-	N
rentacar	17	29274325.2003	0.52	30356760.9841	55	0.04	Y
rgn	72	81.8363	0.02	82.2000	0	0.01	Y
rgna	0	48.8000	0.00	82.2000	2504	0.01	Y
roll3000	375	12092.6533	4.30	13118.0000	417981	5.79%	N
rout	45	982.1729	0.20	1077.5600	1008167	19.34	Y
set1ch	439	52093.5939	0.28	54537.7500	2819811	1.43%	N
seymour	8	406.0000	1.36	434.0000	74264	6.22%	N
seymour1	15	405.4461	3.39	410.7637	35874	24.74	Y
sp97ar	282	653445845.1576	5.50	676558691.7800	66191	3.35%	N
stein27	7	13.0000	0.01	18.0000	4240	0.01	Y
stein45	0	22.0000	0.02	30.0000	62605	0.34	Y
stp3d	6	481.9510	212.39	-	12	-	N
swath	65	378.0411	690.03	519.5717	339470	27.17%	N
swath2	24	334.4969	1.53	385.1997	444677	41.53	Y
swath3	24	334.4969	1.69	399.6350	637116	11.39%	N
t1717	0	134532.0000	7.62	-	2501	-	N
timtab1	254	261727.6992	0.45	789911.0000	3860929	56.68%	N
timtab2	371	366676.8464	0.80	1527027.0000	2840540	75.98%	N
tr12-15	364	67712.0680	0.45	74833.0000	3181167	6.29%	N
tr12-30	842	115449.2434	1.20	132021.0000	1483411	11.13%	N
tr24-15	709	124783.1587	1.26	137126.0000	1608935	7.12%	N
tr24-30	984	228426.7873	0.77	296731.0000	1222256	22.51%	N
tr6-15	205	34977.6634	0.34	37721.0000	211284	2.07	Y
tr6-30	333	55795.2893	0.05	61806.0000	3700784	5.4%	N
vpm1	45	20.0000	0.02	20.0000	3	0.01	Y
vpm2	126	13.0544	0.20	13.7500	13580	0.13	Y
vpm2a	111	13.0499	0.05	13.7500	7432	0.07	Y
vpm5	151	3002.7449	0.62	3003.2000	1073	0.07	Y

Table D.23.: Results for a 1-hour test with the improved SOTA configuration but without generating cuts out of cuts.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.27	924.0000	235	0.04	Y
30_05_100	0	9.0000	13.83	14.0000	181676	35.71%	N
30_95_100	1	3.0000	16.83	3.0000	3897	7.24	Y
30_95_98	0	12.0000	12.08	13.0000	85812	7.69%	N
a1c1s1	761	6119.3020	3.41	11749.4579	587986	40.6%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
acc0	7	0.0000	0.36	0.0000	236	0.06	Y
acc1	14	0.0000	0.89	0.0000	1076	0.32	Y
acc2	9	0.0000	0.73	0.0000	4454	1.96	Y
acc3	0	0.0000	0.30	0.0000	4630	7.33	Y
acc4	0	0.0000	0.31	-	22265	-	N
acc5	0	0.0000	1.24	0.0000	3664	25.03	Y
aflow30a	252	1074.0000	2.56	1158.0000	24538	1.42	Y
aflow40b	353	1082.0000	16.85	1282.0000	100730	12.17%	N
air03	2	338864.2500	0.14	340160.0000	0	0.01	Y
air04	0	55536.0000	2.38	56138.0000	5473	2.45	Y
air05	0	25878.0000	0.88	26374.0000	16466	3.6	Y
arki001	140	7579880.1818	15.14	-	1211	-	N
atlanta-ip	473	81.2791	22.25	-	2727	-	N
b4-10	163	13323.9283	1.52	14050.8397	31726	1.18	Y
b4-10b	126	13978.4298	4.41	14050.8397	192	0.09	Y
b4-12	295	14812.9738	1.52	16103.8837	1361531	0.4%	N
b4-12b	211	15844.0956	9.72	16103.8837	1903	0.49	Y
b4-20b	321	22539.2464	32.76	23360.8870	55007	1.2%	N
BASF6-10	160	20960.5787	2.39	21267.8894	255018	31.96	Y
BASF6-5	183	11897.9721	1.38	12072.3655	38524	3.91	Y
bc1	63	2.6115	120.72	3.3384	8354	51.82	Y
bell3a	16	873196.5787	0.08	878430.3160	45716	0.13	Y
bell5	29	8922311.1807	0.06	8966406.4915	2145218	6.38	Y
bienst1	107	14.1023	0.70	46.7500	38926	2.39	Y
bienst2	131	14.9268	0.73	54.6000	228156	12.93	Y
binkar10_1	92	6702.8150	0.64	6742.2000	632947	15.98	Y
blend2	37	7.0952	0.58	7.5990	2997	0.04	Y
cap6000	7	-2451535.0000	0.14	-2451377.0000	37770	4.4	Y
ches1	73	73.7856	0.47	74.3405	30	0.01	Y
ches2	66	-2891.6536	0.17	-2889.5569	2758190	32.58	Y
ches3	29	-1303896.9248	0.02	-1303896.9248	0	0	Y
ches4	32	-647403.5167	0.02	-647403.5167	13	0	Y
ches5	74	-7370.5261	0.09	-7342.8188	4184	0.08	Y
clorox	217	20944.2689	1.34	21217.8144	128	0.04	Y
Con-12	239	4590.7539	0.74	7593.3400	112808	2.59	Y
con-24	287	18099.6337	0.55	25804.9600	1799528	1.68%	N
dano3mip	610	576.6728	38.12	732.9667	3460	21.31%	N
dano3_3	100	576.2550	10.67	576.3964	9	1.62	Y
dano3_4	188	576.2782	12.02	576.4352	25	2.61	Y
dano3_5	312	576.3266	15.16	576.9249	221	6.24	Y
danoint	147	62.7132	0.77	65.6667	421153	4.31%	N
dcmulti	151	187366.6238	0.47	188182.0000	178	0.01	Y
disktom	0	-5000.0000	0.49	-	136080	-	N
dlsp	31	375.3100	0.44	613.0000	103209	2.19	Y
ds	0	57.2346	5.41	-	2224	-	N
dsbmip	64	-305.1982	0.17	-305.1982	50	0.02	Y
egout	18	567.0998	0.00	568.1007	0	0	Y
enigma	2	0.0000	0.02	0.0000	21176	0.04	Y
fast0507	2	173.0000	5.48	176.0000	12504	1.7%	N
fiber	73	388277.9881	0.44	405935.1800	107	0.01	Y
fixnet6	172	3813.8131	2.22	3983.0000	112	0.05	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0	Y
gen	40	112312.9529	0.08	112313.3627	0	0	Y
gesa2	196	25776436.7954	0.66	25779856.3717	138	0.02	Y
gesa2_o	260	25777105.7004	0.67	25779856.3717	22	0.02	Y
gesa3	183	27970743.0737	0.80	27991042.6484	64	0.03	Y
gesa3_o	235	27963539.9613	0.94	27991042.6484	117	0.03	Y
glass4	43	800002400.0000	0.06	1650014050.0000	5629172	51.52%	N
gt2	31	20726.0000	0.02	21166.0000	193	0	Y
harp2	101	-74229925.0000	2.30	-73899798.0000	2350783	37.34	Y
khb05250	124	106915722.2610	0.34	106940226.0000	22	0.01	Y
l152lav	0	4657.0000	0.16	4722.0000	10637	0.21	Y
liu	619	560.0000	0.53	1284.0000	439413	56.39%	N
lrn	797	44535469.9213	12.41	44710462.5337	262090	0.19%	N
lseu	36	1030.0000	0.11	1120.0000	763	0	Y
m20-75-1	620	-51174.1673	143.89	-49213.0000	229295	2.95%	N
m20-75-2	700	-51950.5450	159.36	-50322.0000	127188	36.64	Y
m20-75-3	810	-53170.6273	109.08	-51102.0000	218739	3.6%	N
m20-75-4	401	-54696.7132	102.75	-52752.0000	283534	1.2%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
m20-75-5	417	-53045.5947	94.81	-51349.0000	161020	40.04	Y
manna81	0	-13297.0000	0.36	-13163.0000	887449	1.02%	N
markshare1	4	0.0000	0.03	7.0000	9999999	100%	N
markshare1_1	7	0.0000	0.01	0.0000	207192	0.43	Y
markshare2	6	0.0000	0.01	17.0000	9999999	100%	N
markshare2_1	12	0.0000	0.02	0.0000	7844144	22.06	Y
mas74	25	10583.2346	0.91	11801.1857	6719973	3.51%	N
mas76	23	39010.8237	0.81	40005.0541	1159854	8.1	Y
misc03	0	1910.0000	0.02	3360.0000	603	0	Y
misc06	29	12846.2683	0.12	12851.0763	19	0.02	Y
misc07	0	1415.0000	0.05	2810.0000	37474	0.56	Y
mitre	979	115119.0000	5.59	115155.0000	1171	0.21	Y
mkc	410	-603.9839	16.03	-556.8700	432140	8.2%	N
mod008	49	304.0000	0.80	307.0000	2142	0.06	Y
mod010	4	6535.0000	0.66	6548.0000	18	0.01	Y
mod011	813	-56510687.3737	4.05	-54558535.0142	2244	0.88	Y
modglob	166	20715756.4942	0.28	20740508.0863	492	0.01	Y
momentum1	687	96249.1952	48.86	-	80	-	N
momentum2	1063	10699.2626	126.26	-	952	-	N
momentum3	3133	94407.6540	2784.00	-	4	-	N
msc98-ip	445	19695288.0058	9.42	22088602.0058	3390	10.8%	N
multiA	85	3568.9318	0.36	3774.7600	488338	10.25	Y
multiB	106	3628.6488	0.41	3999.3500	1861192	8.95%	N
multiC	101	1501.6364	0.45	2088.4200	1984896	24.42%	N
multiD	196	3955.3376	0.81	6254.6450	1139657	35.79%	N
multiE	245	2299.8939	0.53	2721.7425	2428139	13.49%	N
multiF	218	2070.0352	0.52	2429.5300	2648760	13.45%	N
mzzv11	179	-22643.0000	74.19	-21648.0000	47997	2.34%	N
mzzv42z	75	-21450.0000	55.41	-19308.0000	8869	9.03%	N
neos1	87	7.0000	0.28	19.0000	1892683	57.89%	N
neos10	81	-1182.0000	214.28	-1135.0000	32	3.67	Y
neos11	10	6.0000	0.73	9.0000	30044	16.5	Y
neos12	6	9.4116	0.73	13.0000	11803	16.22%	N
neos13	4	-126.1784	345.98	-84.2047	113788	49.85%	N
neos2	110	-3986.4225	11.41	454.8647	132551	11.29	Y
neos20	323	-474.8940	0.80	-434.0000	42015	1.78	Y
neos21	0	3.0000	0.05	7.0000	30130	2.4	Y
neos22	199	778990.4286	1.23	779715.0000	0	0.02	Y
neos23	136	59.3098	1.12	137.0000	3341506	40.15%	N
neos3	174	-5674.9374	15.66	368.8428	790009	304.2%	N
neos4	0	-49463016984.6474	2.51	-48603440750.5898	1305	0.62	Y
neos5	0	13.0000	0.02	15.0000	7940043	3.33%	N
neos6	4	83.0000	3.28	83.0000	4721	7.36	Y
neos648910	425	16.0000	0.61	32.0000	52670	1.42	Y
neos671048	3	2999.0000	2.28	5001.0000	873	0.43	Y
neos7	300	686493.9935	1.44	721934.0000	76992	2.98%	N
neos8	23	-3725.0000	177.22	-3719.0000	0	2.97	Y
neos9	35	794.0000	12.83	798.0000	13006	0.5%	N
net12	450	78.0000	13.33	-	56187	-	N
noswot	10	-43.0000	0.03	-41.0000	9999999	4.88%	N
nsrand-ipx	255	50185.0000	1.81	53600.0000	126441	6.32%	N
nug08	0	204.0000	0.22	214.0000	151	0.1	Y
nw04	0	16311.0000	1.00	16862.0000	1638	1.12	Y
opt1217	31	-19.3221	0.08	-16.0000	5154783	20.76%	N
p0033	22	2942.0000	0.53	3089.0000	75	0.03	Y
p0201	8	7125.0000	0.41	7615.0000	1099	0.02	Y
p0282	102	255636.0000	0.17	258411.0000	48	0.01	Y
p0548	147	8688.0000	0.03	8691.0000	0	0.01	Y
p2756	260	3120.0000	2.11	3124.0000	316	0.08	Y
pk1	0	0.0000	0.00	11.0000	529908	1.79	Y
pp08a	223	7236.1931	0.44	7350.0000	692	0.02	Y
pp08aCUTS	161	7204.3638	0.52	7350.0000	992	0.03	Y
prod1	129	-81.3771	0.62	-56.0000	1442337	20.27	Y
prod2	128	-85.2228	2.81	-61.0000	2722626	18.39%	N
protfold	0	-41.0000	0.22	-	5	-	N
qap10	0	333.0000	0.59	358.0000	43	6.98%	N
qiu	0	-931.6389	0.05	-132.8731	15640	1.35	Y
qnet1	76	15438.7245	0.44	16029.6927	298	0.03	Y
qnet1_o	90	15624.5847	0.47	16030.9927	230	0.02	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
ran10x26	169	4095.7656	1.12	4270.0000	27445	0.66	Y
ran12x21	199	3460.4177	6.19	3664.0000	74456	2.18	Y
ran13x13	185	3065.7457	0.73	3252.0000	44361	0.87	Y
rd-rplusc-21	205	100.0000	535.36	-	17999	-	N
rentacar	22	29274325.2003	0.66	30356760.9841	33	0.03	Y
rgn	83	82.1999	0.00	82.2000	0	0.01	Y
rgna	0	48.8000	0.00	82.2000	2504	0	Y
roll3000	341	11486.5552	4.97	13428.0000	529767	11.84%	N
rout	29	982.1729	0.14	1077.5600	599083	9.56	Y
set1ch	429	54530.8609	0.31	54537.7500	84	0.02	Y
seymour	8	406.0000	1.36	434.0000	74865	6.22%	N
seymour1	16	405.4473	5.22	410.7637	25409	18.01	Y
sp97ar	282	653445845.1576	5.56	673207925.0400	62596	2.87%	N
stein27	7	13.0000	0.02	18.0000	4240	0.01	Y
stein45	0	22.0000	0.02	30.0000	62605	0.34	Y
stp3d	6	481.9510	210.50	-	12	-	N
swath	90	380.2513	329.62	498.2925	409090	23.67%	N
swath2	24	334.4969	1.53	385.1997	444677	41.47	Y
swath3	24	334.4969	1.70	399.6350	638698	11.39%	N
t1717	0	134532.0000	7.72	-	2507	-	N
timtab1	267	271520.7170	0.59	796863.0000	3942342	60.08%	N
timtab2	404	378673.9016	1.03	-	2665902	-	N
tr12-15	409	73877.4789	0.28	74634.0000	13816	0.36	Y
tr12-30	846	130177.2010	0.45	130596.0000	996224	38.92	Y
tr24-15	789	136365.6881	0.92	136509.0000	32545	1.47	Y
tr24-30	984	238660.0922	1.00	294724.0000	1204150	19.02%	N
tr6-15	230	37253.8417	0.30	37721.0000	7910	0.11	Y
tr6-30	360	60975.7141	0.17	61746.0000	1383186	22.21	Y
vpm1	62	20.0000	0.05	20.0000	3	0	Y
vpm2	172	13.0586	0.36	13.7500	14666	0.19	Y
vpm2a	145	13.1231	1.94	13.7500	6273	0.1	Y
vpm5	136	3002.7327	0.80	3003.2000	254	0.03	Y

Table D.24.: Results for a 1-hour test with the improved SOTA configuration but without flow cover cuts.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.23	924.0000	235	0.04	Y
30_05_100	0	9.0000	13.69	14.0000	181471	35.71%	N
30_95_100	1	3.0000	16.94	3.0000	3897	7.26	Y
30_95_98	0	12.0000	12.20	13.0000	85721	7.69%	N
a1c1s1	743	6062.2554	3.05	11651.4325	565353	40.05%	N
acc0	7	0.0000	0.38	0.0000	236	0.05	Y
acc1	14	0.0000	0.94	0.0000	1076	0.33	Y
acc2	9	0.0000	0.66	0.0000	4454	1.96	Y
acc3	0	0.0000	0.30	0.0000	4630	7.35	Y
acc4	0	0.0000	0.31	-	22242	-	N
acc5	0	0.0000	1.22	0.0000	3664	24.99	Y
aflow30a	235	1077.0000	1.56	1158.0000	12293	0.66	Y
aflow40b	381	1081.0000	16.30	1179.0000	183483	5%	N
air03	2	338864.2500	0.12	340160.0000	0	0.01	Y
air04	0	55536.0000	2.39	56138.0000	5473	2.44	Y
air05	0	25878.0000	0.84	26374.0000	16466	3.58	Y
arki001	140	7579880.1818	14.59	-	1211	-	N
atlanta-ip	473	81.2791	21.72	-	2747	-	N
b4-10	200	13334.5382	1.72	14050.8397	24759	0.82	Y
b4-10b	171	13977.7184	7.47	14050.8397	327	0.22	Y
b4-12	270	14757.9649	1.66	16103.8837	1163169	49.3	Y
b4-12b	210	15852.4941	9.88	16103.8837	6682	1.18	Y
b4-20b	324	22449.5557	27.23	23358.2110	59773	2.35%	N
BASF6-10	160	20957.2823	2.17	21267.5689	129870	17.78	Y
BASF6-5	181	11895.7453	1.41	12071.5772	37122	3.67	Y
bc1	63	2.6115	118.67	3.3384	8354	51.58	Y
bell3a	16	873196.5787	0.06	878430.3160	45716	0.13	Y
bell5	29	8922311.1807	0.06	8966406.4915	2145218	6.34	Y
bienst1	94	14.0766	0.58	46.7500	40494	2.21	Y
bienst2	112	14.9297	0.56	54.6000	277749	14.4	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
binkar10_1	92	6702.8150	0.61	6742.2000	632947	15.89	Y
blend2	37	7.0952	0.58	7.5990	2997	0.04	Y
cap6000	7	-2451535.0000	0.12	-2451377.0000	37770	4.38	Y
ches1	73	73.7856	0.45	74.3405	30	0.01	Y
ches2	66	-2891.6536	0.16	-2889.5569	2758190	32.44	Y
ches3	30	-1303896.9248	0.05	-1303896.9248	6	0	Y
ches4	32	-647403.5167	0.02	-647403.5167	13	0	Y
ches5	70	-7370.9964	0.08	-7342.8188	9546	0.16	Y
clorox	162	17155.6482	0.42	21217.8144	274	0.03	Y
Con-12	184	4598.1991	0.27	7593.0700	213324	3.96	Y
con-24	290	18035.5571	0.47	25804.9600	1917221	2.47%	N
dano3mip	504	576.5699	48.39	775.6500	4961	25.65%	N
dano3_3	25	576.2361	0.83	576.3964	9	1.72	Y
dano3_4	72	576.2557	3.25	576.4352	22	2.16	Y
dano3_5	110	576.3003	7.28	576.9249	230	5	Y
danoint	103	62.6944	0.56	65.6667	512452	4.02%	N
dcmulti	134	187333.5090	0.38	188182.0000	214	0.01	Y
disktom	0	-5000.0000	0.53	-	136206	-	N
dlsp	31	375.3100	0.38	613.0000	103209	2.19	Y
ds	0	57.2346	5.53	-	2222	-	N
dsbmip	64	-305.1982	0.09	-305.1982	50	0.01	Y
egout	18	567.0998	0.02	568.1007	0	0	Y
enigma	2	0.0000	0.00	0.0000	21176	0.04	Y
fast0507	2	173.0000	5.39	176.0000	12504	1.7%	N
fiber	73	388277.9881	0.44	405935.1800	107	0.01	Y
fixnet6	172	3813.8131	2.08	3983.0000	112	0.05	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0	Y
gen	40	112312.9529	0.08	112313.3627	0	0	Y
gesa2	178	25776342.8956	0.55	25779856.3717	229	0.02	Y
gesa2_o	260	25777105.7004	0.62	25779856.3717	22	0.02	Y
gesa3	183	27970743.0737	0.73	27991042.6484	64	0.03	Y
gesa3_o	235	27963539.9613	0.88	27991042.6484	117	0.03	Y
glass4	43	800002400.0000	0.05	1650014050.0000	5639756	51.52%	N
gt2	31	20726.0000	0.02	21166.0000	193	0	Y
harp2	101	-74229925.0000	2.30	-73899798.0000	2350783	37.3	Y
khb05250	124	106915722.2610	0.31	106940226.0000	22	0.01	Y
l152lav	0	4657.0000	0.17	4722.0000	10637	0.22	Y
liu	542	560.0000	0.39	1332.0000	560539	57.96%	N
lrn	801	44553374.5369	10.80	44656794.6587	360198	0.01%	N
lseu	36	1030.0000	0.11	1120.0000	763	0	Y
m20-75-1	618	-51174.1673	138.62	-49213.0000	238112	2.8%	N
m20-75-2	700	-51950.5450	152.53	-50322.0000	127188	36.45	Y
m20-75-3	799	-53184.5835	107.44	-51158.0000	218416	3.39%	N
m20-75-4	435	-54662.9976	108.44	-52752.0000	296193	2.07%	N
m20-75-5	363	-53108.7897	131.12	-51349.0000	116062	27.74	Y
manna81	0	-13297.0000	0.36	-13163.0000	858302	1.02%	N
markshare1	4	0.0000	0.03	7.0000	9999999	100%	N
markshare1_1	7	0.0000	0.02	0.0000	207192	0.44	Y
markshare2	6	0.0000	0.02	17.0000	9999999	100%	N
markshare2_1	12	0.0000	0.02	0.0000	7844144	22.08	Y
mas74	25	10583.2346	0.88	11801.1857	6710044	3.51%	N
mas76	23	39010.8237	0.80	40005.0541	1159854	8.12	Y
misc03	0	1910.0000	0.00	3360.0000	603	0	Y
misc06	29	12846.2683	0.11	12851.0763	19	0.01	Y
misc07	0	1415.0000	0.03	2810.0000	37474	0.56	Y
mitre	979	115119.0000	5.78	115155.0000	1171	0.21	Y
mkc	410	-603.9839	15.74	-556.8700	431285	8.2%	N
mod008	49	304.0000	0.80	307.0000	2142	0.06	Y
mod010	4	6535.0000	0.66	6548.0000	18	0.01	Y
mod011	813	-56510687.3737	3.70	-54558535.0142	2244	0.87	Y
modglob	166	20715756.4942	0.24	20740508.0863	492	0.01	Y
momentum1	688	96249.1952	45.33	-	210	-	N
momentum2	895	10698.4783	103.95	-	63	-	N
momentum3	1388	91964.1297	1522.62	n.a.	0	n.a.	N
msc98-ip	414	19695288.0058	9.09	-	2947	-	N
multiA	65	3557.1627	0.05	3774.7600	76963	1.6	Y
multiB	112	3629.2792	0.41	3984.0300	1746405	8.56%	N
multiC	72	1492.4464	0.38	2095.0200	2274655	25.61%	N
multiD	146	3920.2823	0.73	6100.3564	1186589	35.07%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
multiE	256	2297.8358	0.45	2721.7425	2376292	12.69%	N
multiF	205	2059.3586	0.48	2433.4400	2846645	14.33%	N
mzzv11	179	-22643.0000	74.47	-21648.0000	47729	2.34%	N
mzzv42z	75	-21450.0000	55.80	-19308.0000	8831	9.03%	N
neos1	87	7.0000	0.28	19.0000	1892683	57.89%	N
neos10	81	-1182.0000	213.36	-1135.0000	32	3.65	Y
neos11	10	6.0000	0.69	9.0000	30044	16.49	Y
neos12	6	9.4116	0.72	13.0000	11789	16.22%	N
neos13	4	-126.1784	33.70	-84.2047	132431	49.85%	N
neos2	110	-3986.4225	11.27	454.8647	132551	11.28	Y
neos20	323	-474.8940	0.80	-434.0000	42015	1.78	Y
neos21	0	3.0000	0.06	7.0000	30130	2.41	Y
neos22	198	778990.4286	1.12	779715.0000	0	0.02	Y
neos23	122	59.3098	0.89	137.0000	3421614	48.43%	N
neos3	174	-5674.9374	15.66	368.8428	789519	304.2%	N
neos4	0	-49463016984.6474	2.39	-48603440750.5898	1305	0.62	Y
neos5	0	13.0000	0.00	15.0000	7939707	3.33%	N
neos6	4	83.0000	3.34	83.0000	4721	7.4	Y
neos648910	380	16.0000	0.41	32.0000	252891	6.54	Y
neos671048	3	2999.0000	2.27	5001.0000	873	0.43	Y
neos7	380	695844.4985	1.91	721934.0000	462557	35.61	Y
neos8	23	-3725.0000	178.06	-3719.0000	0	2.99	Y
neos9	35	794.0000	12.91	798.0000	12922	0.5%	N
net12	450	78.0000	13.31	-	55846	-	N
noswot	10	-43.0000	0.02	-41.0000	9999999	4.88%	N
nsrand-ipx	255	50185.0000	1.80	53600.0000	126251	6.32%	N
nug08	0	204.0000	0.22	214.0000	151	0.1	Y
nw04	0	16311.0000	1.02	16862.0000	1638	1.12	Y
opt1217	31	-19.3221	0.08	-16.0000	5240025	20.76%	N
p0033	22	2942.0000	0.23	3089.0000	75	0.01	Y
p0201	8	7125.0000	0.42	7615.0000	1099	0.02	Y
p0282	102	255636.0000	0.17	258411.0000	48	0.01	Y
p0548	147	8688.0000	0.05	8691.0000	0	0.01	Y
p2756	260	3120.0000	2.08	3124.0000	316	0.08	Y
pk1	0	0.0000	0.02	11.0000	529908	1.79	Y
pp08a	213	7212.1598	0.38	7350.0000	666	0.02	Y
pp08aCUTS	153	7210.4012	0.41	7350.0000	1000	0.03	Y
prod1	129	-81.3771	0.67	-56.0000	1442337	20.26	Y
prod2	128	-85.2228	2.81	-61.0000	2723064	18.39%	N
protfold	0	-41.0000	0.20	-	5	-	N
qap10	0	333.0000	0.59	358.0000	43	6.98%	N
qiu	0	-931.6389	0.06	-132.8731	15640	1.34	Y
qnet1	76	15438.7245	0.45	16029.6927	298	0.03	Y
qnet1_o	90	15624.5847	0.48	16030.9927	230	0.02	Y
ran10x26	169	4095.7656	1.05	4270.0000	27445	0.66	Y
ran12x21	199	3460.4177	6.06	3664.0000	74456	2.17	Y
ran13x13	185	3065.7457	0.69	3252.0000	44361	0.86	Y
rd-rplusc-21	521	100.0000	1242.52	-	9633	-	N
rentacar	22	29274325.2003	0.53	30356760.9841	30	0.03	Y
rgn	83	82.1999	0.02	82.2000	0	0	Y
rgna	0	48.8000	0.00	82.2000	2504	0	Y
roll3000	341	11486.5552	4.81	13428.0000	528272	11.84%	N
rout	29	982.1729	0.14	1077.5600	599083	9.6	Y
set1ch	440	54500.9343	0.34	54537.7500	61	0.02	Y
seymour	8	406.0000	1.36	434.0000	74609	6.22%	N
seymour1	16	405.4473	4.03	410.7637	25409	17.98	Y
sp97ar	282	653445845.1576	5.59	673207925.0400	62624	2.87%	N
stein27	7	13.0000	0.02	18.0000	4240	0.01	Y
stein45	0	22.0000	0.02	30.0000	62605	0.34	Y
stp3d	6	481.9510	211.52	-	12	-	N
swath	65	379.4261	686.05	517.0587	357218	26.55%	N
swath2	24	334.4969	1.53	385.1997	444677	41.45	Y
swath3	24	334.4969	1.72	399.6350	638341	11.39%	N
t1717	0	134532.0000	7.75	-	2507	-	N
timtab1	267	271520.7170	0.50	796863.0000	3953486	60.08%	N
timtab2	405	373167.5107	1.33	-	2542849	-	N
tr12-15	373	73517.6543	0.59	74634.0000	66292	1.35	Y
tr12-30	878	129187.0391	1.64	130596.0000	1352633	0.4%	N
tr24-15	754	135382.1529	1.97	136509.0000	688155	27.57	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
tr24-30	984	238241.4596	0.92	294751.0000	1214482	18.82%	N
tr6-15	236	36996.8928	0.41	37721.0000	10298	0.13	Y
tr6-30	372	60705.5345	0.73	61746.0000	3751542	0.46%	N
vpm1	62	20.0000	0.03	20.0000	3	0	Y
vpm2	172	13.0586	0.33	13.7500	14666	0.18	Y
vpm2a	145	13.1231	1.91	13.7500	6273	0.1	Y
vpm5	136	3002.7327	0.75	3003.2000	254	0.03	Y

Table D.25.: Results for a 1-hour test with the improved SOTA configuration but without flow cover and flow path cuts. For the instance `momentum3`, MOPS returned an invalid result.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.25	924.0000	235	0.04	Y
30_05_100	0	9.0000	13.80	14.0000	181560	35.71%	N
30_95_100	1	3.0000	16.88	3.0000	3897	7.23	Y
30_95_98	0	12.0000	12.17	13.0000	85936	7.69%	N
a1c1s1	767	6213.4499	3.39	12123.4474	659079	45.08%	N
acc0	7	0.0000	0.39	0.0000	236	0.06	Y
acc1	14	0.0000	0.97	0.0000	1076	0.33	Y
acc2	9	0.0000	0.67	0.0000	4454	1.96	Y
acc3	0	0.0000	0.30	0.0000	4630	7.32	Y
acc4	0	0.0000	0.31	-	22292	-	N
acc5	0	0.0000	1.23	0.0000	3664	24.95	Y
aflow30a	230	1071.0000	1.86	1158.0000	19784	0.96	Y
aflow40b	376	1081.0000	16.83	1202.0000	124239	7.07%	N
air03	2	338864.2500	0.14	340160.0000	0	0.01	Y
air04	0	55536.0000	2.39	56138.0000	5473	2.44	Y
air05	0	25878.0000	0.86	26374.0000	16466	3.59	Y
arki001	147	7579880.6002	17.73	7580814.5116	836338	0.01%	N
atlanta-ip	478	81.2791	21.66	-	2049	-	N
b4-10	198	13318.4808	2.89	14050.8397	71987	2.55	Y
b4-10b	122	13984.4225	4.36	14050.8397	201	0.09	Y
b4-12	329	14820.7416	1.25	16103.8837	606278	27.63	Y
b4-12b	221	15852.1335	8.08	16103.8837	11333	1.72	Y
b4-20b	293	22439.1537	30.48	23376.6473	62389	2.3%	N
BASF6-10	159	20959.8571	2.38	21267.5689	28650	4.25	Y
BASF6-5	183	11898.6064	1.39	12072.4747	35164	3.12	Y
bc1	62	2.5908	118.09	3.3611	9886	1.55%	N
bell3a	16	873196.5787	0.08	878430.3160	45716	0.13	Y
bell5	29	8922311.1807	0.08	8966406.4915	2145218	6.38	Y
bienst1	133	14.1072	0.70	46.7500	35872	2.29	Y
bienst2	114	14.9358	0.66	54.6000	264746	13.87	Y
binkar10_1	92	6702.8150	0.66	6742.2000	632947	16.21	Y
blend2	37	7.0952	0.58	7.5990	2997	0.04	Y
cap6000	7	-2451535.0000	0.14	-2451377.0000	37770	4.39	Y
ches1	40	73.0187	0.08	74.3405	33	0.01	Y
ches2	66	-2891.6536	0.17	-2889.8455	2976409	36.38	Y
ches3	30	-1303896.9248	0.05	-1303896.9248	6	0.01	Y
ches4	36	-647403.5167	0.02	-647403.5167	17	0.01	Y
ches5	67	-7371.0644	0.08	-7342.8188	8972	0.16	Y
clorox	206	13819.9124	1.38	21217.8144	374	0.05	Y
Con-12	207	4585.4797	0.66	7593.0700	196778	4.21	Y
con-24	323	18209.5900	1.00	25804.9600	1698648	1.79%	N
dano3mip	517	576.5667	43.81	748.3889	5508	22.94%	N
dano3_3	31	576.2371	3.42	576.3964	9	1.49	Y
dano3_4	66	576.2554	3.75	576.4352	22	1.69	Y
dano3_5	134	576.3015	11.55	576.9249	257	5.29	Y
danoint	108	62.7006	0.66	65.6667	503020	4.07%	N
dcmulti	134	187333.5090	0.42	188182.0000	214	0.01	Y
disktom	0	-5000.0000	0.45	-	136016	-	N
dlsp	31	375.3100	0.64	613.0000	103209	2.19	Y
ds	0	57.2346	5.41	-	2216	-	N
dsbmip	84	-305.1982	0.19	-305.1982	50	0.02	Y
egout	18	567.0998	0.02	568.1007	0	0.01	Y
enigma	2	0.0000	0.00	0.0000	21176	0.04	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
fast0507	2	173.0000	5.50	176.0000	12494	1.7%	N
fiber	73	388277.9881	0.45	405935.1800	107	0.01	Y
fixnet6	173	3808.3357	2.19	3983.0000	78	0.05	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0.01	Y
gen	40	112312.9529	0.09	112313.3627	0	0.01	Y
gesa2	178	25776342.8956	0.59	25779856.3717	229	0.03	Y
gesa2_o	260	25777105.7004	0.67	25779856.3717	22	0.02	Y
gesa3	183	27970743.0737	0.78	27991042.6484	64	0.03	Y
gesa3_o	235	27963539.9613	0.98	27991042.6484	117	0.03	Y
glass4	43	800002400.0000	0.05	1650014050.0000	5623973	51.52%	N
gt2	31	20726.0000	0.03	21166.0000	193	0.01	Y
harp2	101	-74229925.0000	2.45	-73899798.0000	2350783	37.37	Y
khb05250	125	106915481.5201	0.27	106940226.0000	24	0.01	Y
l152lav	0	4657.0000	0.16	4722.0000	10637	0.22	Y
liu	698	560.0000	0.59	1356.0000	762646	58.7%	N
lrn	790	44552129.8923	11.58	44656797.1919	364368	0.02%	N
lseu	36	1030.0000	0.11	1120.0000	763	0.01	Y
m20-75-1	618	-51174.1673	139.17	-49213.0000	237584	2.8%	N
m20-75-2	700	-51950.5450	153.20	-50322.0000	127188	36.49	Y
m20-75-3	799	-53184.5835	108.11	-51158.0000	218155	3.39%	N
m20-75-4	435	-54662.9976	108.73	-52752.0000	295746	2.07%	N
m20-75-5	363	-53108.7897	131.59	-51349.0000	116062	27.74	Y
manna81	0	-13297.0000	0.39	-13163.0000	889304	1.02%	N
markshare1	4	0.0000	0.00	7.0000	9999999	100%	N
markshare1_1	7	0.0000	0.02	0.0000	328541	0.66	Y
markshare2	6	0.0000	0.02	17.0000	9999999	100%	N
markshare2_1	12	0.0000	0.02	0.0000	7844144	22.07	Y
mas74	25	10583.2346	0.89	11801.1857	6716952	3.51%	N
mas76	23	39010.8237	0.84	40005.0541	1159854	8.13	Y
misc03	0	1910.0000	0.00	3360.0000	603	0.01	Y
misc06	28	12847.3366	0.28	12851.0763	18	0.01	Y
misc07	0	1415.0000	0.05	2810.0000	37474	0.56	Y
mitre	979	115119.0000	5.69	115155.0000	1171	0.22	Y
mkc	410	-603.9839	15.89	-556.8700	431379	8.2%	N
mod008	49	304.0000	0.80	307.0000	2142	0.06	Y
mod010	4	6535.0000	0.66	6548.0000	18	0.01	Y
mod011	817	-56517956.2438	3.95	-54558535.0142	2152	0.84	Y
modglob	174	20722217.3080	0.33	20740508.0863	380	0.01	Y
momentum1	824	102663.0180	55.06	-	145	-	N
momentum2	982	10698.4399	89.61	-	111	-	N
momentum3	3904	94206.9284	3224.45	-	1	-	N
msc98-ip	521	19702877.0058	23.23	22191032.0059	2279	11.21%	N
multiA	67	3563.1030	0.05	3774.7600	424216	8.89	Y
multiB	82	3627.8627	0.06	3995.5200	2124210	8.74%	N
multiC	82	1513.0697	0.41	2083.2867	1999631	22.93%	N
multiD	105	3891.4576	0.36	6102.3545	1532767	35.37%	N
multiE	252	2301.1218	0.58	2710.5925	2421076	11.78%	N
multiF	200	2067.3049	0.44	2447.4000	3480146	14.92%	N
mzzv11	179	-22643.0000	74.58	-21648.0000	47747	2.34%	N
mzzv42z	75	-21450.0000	55.78	-19308.0000	8873	9.03%	N
neos1	87	7.0000	0.33	19.0000	1892683	57.89%	N
neos10	81	-1182.0000	212.92	-1135.0000	32	3.65	Y
neos11	10	6.0000	0.73	9.0000	30044	16.49	Y
neos12	9	9.4116	0.70	13.0000	12699	17.88%	N
neos13	4	-126.1784	36.53	-84.2047	132361	49.85%	N
neos2	131	-4066.6442	9.25	454.8647	175218	10.70	Y
neos20	323	-474.8940	0.83	-434.0000	42015	1.78	Y
neos21	0	3.0000	0.06	7.0000	30130	2.41	Y
neos22	198	778990.4286	1.30	779715.0000	0	0.02	Y
neos23	171	59.7439	1.47	137.0000	3119944	45.96%	N
neos3	168	-5720.9576	15.28	368.9010	694822	561.92%	N
neos4	0	-49463016984.6474	2.52	-48603440750.5898	1305	0.62	Y
neos5	0	13.0000	0.01	15.0000	7936160	3.33%	N
neos6	4	83.0000	3.30	83.0000	4721	7.35	Y
neos648910	393	16.0000	0.62	32.0000	917702	24.84	Y
neos671048	3	2999.0000	2.41	5001.0000	873	0.44	Y
neos7	312	687129.0593	1.48	721934.0000	682106	1.05%	N
neos8	23	-3725.0000	177.83	-3719.0000	0	2.98	Y
neos9	35	794.0000	13.12	798.0000	13012	0.5%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
net12	450	78.0000	13.81	-	56134	-	N
noswot	10	-43.0000	0.03	-41.0000	9999999	4.88%	N
nsrand-ipx	255	50185.0000	1.81	53600.0000	126365	6.32%	N
nug08	0	204.0000	0.22	214.0000	151	0.10	Y
nw04	0	16311.0000	1.02	16862.0000	1638	1.12	Y
opt1217	31	-19.3221	0.08	-16.0000	4994509	20.76%	N
p0033	22	2942.0000	0.05	3089.0000	75	0.01	Y
p0201	8	7125.0000	0.39	7615.0000	1099	0.02	Y
p0282	102	255636.0000	0.19	258411.0000	48	0.01	Y
p0548	147	8688.0000	0.03	8691.0000	0	0.01	Y
p2756	260	3120.0000	2.11	3124.0000	316	0.08	Y
pk1	0	0.0000	0.02	11.0000	529908	1.78	Y
pp08a	213	7209.9894	0.42	7350.0000	648	0.02	Y
pp08aCUTS	153	7210.4690	0.38	7350.0000	1014	0.03	Y
prod1	129	-81.3771	0.61	-56.0000	1442337	20.30	Y
prod2	128	-85.2228	2.84	-61.0000	2721324	18.39%	N
protfold	0	-41.0000	0.22	-	5	-	N
qap10	0	333.0000	0.59	358.0000	43	6.98%	N
qiu	0	-931.6389	0.05	-132.8731	15640	1.34	Y
qnet1	76	15438.7245	0.45	16029.6927	298	0.03	Y
qnet1_o	90	15624.5847	0.49	16030.9927	230	0.02	Y
ran10x26	174	4096.8543	0.86	4270.0000	28164	0.69	Y
ran12x21	208	3465.8956	4.78	3664.0000	90936	2.55	Y
ran13x13	179	3056.3723	0.95	3252.0000	52922	1.01	Y
rd-rplusc-21	204	100.0000	575.44	-	1480	-	N
rentacar	22	29274325.2003	0.59	30356760.9841	21	0.02	Y
rgn	83	82.1999	0.00	82.2000	0	0.01	Y
rgna	0	48.8000	0.02	82.2000	2504	0.01	Y
roll3000	341	11486.5552	4.78	13428.0000	529236	11.84%	N
rout	29	982.1729	0.14	1077.5600	599083	9.57	Y
set1ch	468	54528.1039	0.64	54537.7500	18	0.02	Y
seymour	8	406.0000	1.42	434.0000	74625	6.22%	N
seymour1	16	405.4473	4.77	410.7637	25409	17.99	Y
sp97ar	282	653445845.1576	5.59	673207925.0400	62592	2.87%	N
stein27	7	13.0000	0.02	18.0000	4240	0.01	Y
stein45	0	22.0000	0.00	30.0000	62605	0.34	Y
stp3d	6	481.9510	211.22	-	12	-	N
swath	87	379.1360	493.86	515.6334	450658	26.47%	N
swath2	24	334.4969	1.61	385.1997	412391	36.54	Y
swath3	24	334.4969	1.70	399.6350	638389	11.39%	N
t1717	0	134532.0000	7.72	-	2513	-	N
timtab1	274	273313.2972	0.67	795007.0000	3777068	54.69%	N
timtab2	403	375016.1681	1.61	-	2513444	-	N
tr12-15	374	73527.0120	0.75	74634.0000	127757	2.64	Y
tr12-30	881	129329.5985	2.25	130596.0000	1290423	0.54%	N
tr24-15	782	135417.4306	2.28	136509.0000	690082	28.80	Y
tr24-30	984	238241.4596	1.03	294751.0000	1208954	18.82%	N
tr6-15	223	36919.2776	0.36	37721.0000	8804	0.11	Y
tr6-30	360	60712.1760	0.36	61746.0000	3774611	0.52%	N
vpm1	55	20.0000	0.00	20.0000	0	0.01	Y
vpm2	182	13.0862	2.94	13.7500	14324	0.23	Y
vpm2a	148	13.1236	2.22	13.7500	6005	0.10	Y
vpm5	126	3002.7392	0.75	3003.2000	459	0.05	Y

Table D.26.: Results for a 1-hour test with the cMIR cut and uPMC generator.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.27	924.0000	235	0.04	Y
30_05_100	0	9.0000	13.92	14.0000	180794	35.71%	N
30_95_100	1	3.0000	17.02	3.0000	3897	7.28	Y
30_95_98	0	12.0000	12.16	13.0000	85459	7.69%	N
a1c1s1	789	6094.8687	3.59	12250.8968	672667	47.68%	N
acc0	7	0.0000	0.38	0.0000	236	0.06	Y
acc1	14	0.0000	0.92	0.0000	1076	0.33	Y
acc2	9	0.0000	0.67	0.0000	4454	1.96	Y
acc3	0	0.0000	0.31	0.0000	4630	7.34	Y
acc4	0	0.0000	0.31	-	22265	-	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
acc5	0	0.0000	1.23	0.0000	3664	25.00	Y
aflow30a	203	1077.0000	1.62	1158.0000	17191	0.74	Y
aflow40b	377	1083.0000	22.86	1481.0000	74750	26.06%	N
air03	2	338864.2500	0.14	340160.0000	0	0.01	Y
air04	0	55536.0000	2.42	56138.0000	5473	2.45	Y
air05	0	25878.0000	0.86	26374.0000	16466	3.59	Y
arki001	173	7579887.8809	21.05	-	948	-	N
atlanta-ip	477	81.2791	22.41	-	490	-	N
b4-10	227	13365.0305	1.84	14050.8397	24811	0.95	Y
b4-10b	116	13973.9289	3.69	14050.8397	338	0.09	Y
b4-12	268	14751.0590	1.92	16103.8837	1343170	2.35%	N
b4-12b	254	15840.5603	12.38	16103.8837	6977	1.44	Y
b4-20b	310	22468.8718	30.64	23387.1591	47155	2.33%	N
BASF6-10	158	20963.0789	2.45	21267.6938	236711	30.09	Y
BASF6-5	188	11896.3509	1.47	12072.3655	28570	3.10	Y
bc1	63	2.6008	140.30	3.3384	10828	1.68%	N
bell3a	16	873196.5787	0.08	878430.3160	56364	0.17	Y
bell5	29	8921811.4174	0.06	8966406.4915	2008250	6.45	Y
bienst1	127	14.1063	0.72	46.7500	35788	2.28	Y
bienst2	112	14.9288	0.66	54.6000	245202	12.45	Y
binkar10_1	92	6702.8150	0.66	6742.2000	632947	15.95	Y
blend2	37	7.0952	0.58	7.5990	2997	0.04	Y
cap6000	7	-2451535.0000	0.14	-2451377.0000	37770	4.39	Y
ches1	80	74.1860	0.25	74.3405	4	0.01	Y
ches2	66	-2891.6536	0.17	-2889.5569	37558	0.47	Y
ches3	30	-1303896.9248	0.05	-1303896.9248	6	0.01	Y
ches4	32	-647403.5167	0.03	-647403.5167	21	0.01	Y
ches5	76	-7370.8392	0.09	-7342.8188	12324	0.23	Y
clorox	142	17357.9455	0.11	21217.8144	1022	0.04	Y
Con-12	176	4593.4914	0.19	7593.3400	172058	3.37	Y
con-24	273	18050.6868	0.94	25804.9600	2084356	58.80	Y
dano3mip	501	576.5796	34.50	770.3077	4884	25.13%	N
dano3_3	21	576.2371	1.72	576.3964	9	1.59	Y
dano3_4	56	576.2548	6.27	576.4352	24	2.24	Y
dano3_5	120	576.3021	10.78	576.9249	286	5.53	Y
danoint	114	62.7018	0.72	65.6667	495818	4.28%	N
dcmulti	139	187305.2019	0.48	188182.0000	226	0.01	Y
disktom	0	-5000.0000	0.45	-	136243	-	N
dlsp	31	375.3100	0.59	613.0000	100424	2.22	Y
ds	0	57.2346	5.55	-	2218	-	N
dsbmip	64	-305.1982	0.12	-305.1982	50	0.01	Y
egout	18	567.0998	0.00	568.1007	0	0.01	Y
enigma	2	0.0000	0.00	0.0000	21176	0.04	Y
fast0507	2	173.0000	5.47	176.0000	12506	1.7%	N
fiber	73	388277.9881	0.45	405935.1800	107	0.01	Y
fixnet6	202	3829.7907	2.42	3983.0000	73	0.05	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0.01	Y
gen	40	112312.9529	0.09	112313.3627	0	0.01	Y
gesa2	178	25776342.8956	0.64	25779856.3717	229	0.03	Y
gesa2_o	260	25777105.7004	0.74	25779856.3717	22	0.03	Y
gesa3	183	27970743.0737	0.88	27991042.6484	64	0.03	Y
gesa3_o	235	27963539.9613	1.03	27991042.6484	117	0.04	Y
glass4	43	800002400.0000	0.05	1650014050.0000	5638752	51.52%	N
gt2	31	20726.0000	0.01	21166.0000	193	0.01	Y
harp2	101	-74229925.0000	2.28	-73899798.0000	2350783	37.27	Y
khb05250	121	106916129.9831	0.25	106940226.0000	22	0.01	Y
l152lav	0	4657.0000	0.14	4722.0000	10637	0.22	Y
liu	584	560.0000	0.51	1282.0000	533800	56.32%	N
lrn	836	44538202.0856	12.12	44656794.6587	346150	0.01%	N
lseu	36	1030.0000	0.11	1120.0000	763	0.01	Y
m20-75-1	618	-51174.1673	138.80	-49213.0000	237900	2.8%	N
m20-75-2	700	-51950.5450	153.20	-50322.0000	127188	36.47	Y
m20-75-3	904	-53141.6722	78.94	-51158.0000	199597	2.91%	N
m20-75-4	413	-54683.5128	114.77	-52752.0000	280224	1.24%	N
m20-75-5	363	-53108.7897	131.67	-51349.0000	116062	27.76	Y
manna81	0	-13297.0000	0.38	-13163.0000	876680	1.02%	N
markshare1	4	0.0000	0.02	7.0000	9999999	100%	N
markshare1_1	7	0.0000	0.02	0.0000	158715	0.34	Y
markshare2	6	0.0000	0.00	17.0000	9999999	100%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
markshare2_1	12	0.0000	0.02	0.0000	7844144	22.10	Y
mas74	25	10583.2346	0.95	11801.1857	6708307	3.51%	N
mas76	23	39010.8237	0.81	40005.0541	1159854	8.12	Y
misc03	0	1910.0000	0.00	3360.0000	603	0.01	Y
misc06	29	12846.2683	0.11	12851.0763	19	0.01	Y
misc07	0	1415.0000	0.05	2810.0000	37474	0.56	Y
mitre	979	115119.0000	5.75	115155.0000	1171	0.21	Y
mkc	410	-603.9839	15.89	-556.8700	432956	8.2%	N
mod008	49	304.0000	0.80	307.0000	2142	0.06	Y
mod010	4	6535.0000	0.64	6548.0000	18	0.01	Y
mod011	823	-56535258.9904	4.05	-54558535.0142	3228	1.14	Y
modglob	188	20725154.3888	0.38	20740508.0863	668	0.02	Y
momentum1	747	96249.2890	46.41	-	53	-	N
momentum2	1056	10716.6411	117.70	-	276	-	N
momentum3	2533	92031.1707	3117.00	n.a.	0	n.a.	N
msc98-ip	523	19699455.1058	13.70	-	3664	-	N
multiA	65	3557.1627	0.05	3774.7600	75797	1.58	Y
multiB	73	3627.8472	0.06	3987.8600	2064174	8.57%	N
multiC	57	1490.9417	0.08	2088.4200	2444970	24.68%	N
multiD	96	3893.9351	0.33	6349.6200	1558295	37.2%	N
multiE	231	2294.8776	0.53	2710.5925	2519888	11.83%	N
multiF	191	2065.1180	0.47	2428.9300	2999632	12.83%	N
mzzv11	179	-22643.0000	74.89	-21648.0000	47937	2.34%	N
mzzv42z	75	-21450.0000	56.03	-19308.0000	8861	9.03%	N
neos1	87	7.0000	0.28	19.0000	1893208	57.89%	N
neos10	81	-1182.0000	214.31	-1135.0000	32	3.67	Y
neos11	10	6.0000	0.75	9.0000	30044	16.48	Y
neos12	6	9.4116	0.75	13.0000	11810	16.22%	N
neos13	4	-126.1784	37.56	-84.2047	132764	49.85%	N
neos2	93	-3965.8024	12.67	454.8647	155362	13.68	Y
neos20	323	-474.8940	0.84	-434.0000	42015	1.78	Y
neos21	0	3.0000	0.05	7.0000	30130	2.40	Y
neos22	198	778990.4286	1.36	779715.0000	0	0.02	Y
neos23	201	62.2901	1.47	137.0000	3073022	46.5%	N
neos3	174	-5694.4573	16.34	369.6544	694536	408.11%	N
neos4	0	-49463016984.6474	2.50	-48603440750.5898	1305	0.62	Y
neos5	0	13.0000	0.02	15.0000	7943964	3.33%	N
neos6	4	83.0000	3.42	83.0000	4721	7.35	Y
neos648910	347	16.0000	0.47	32.0000	365736	8.49	Y
neos671048	3	2999.0000	2.41	5001.0000	873	0.43	Y
neos7	307	686826.5159	1.53	721934.0000	714502	59.89	Y
neos8	23	-3725.0000	177.62	-3719.0000	0	2.98	Y
neos9	35	794.0000	13.12	798.0000	12903	0.5%	N
net12	450	78.0000	14.06	-	55686	-	N
noswot	10	-43.0000	0.03	-41.0000	9999999	4.88%	N
nsrand-ipx	255	50185.0000	1.83	53600.0000	126193	6.32%	N
nug08	0	204.0000	0.22	214.0000	151	0.10	Y
nw04	0	16311.0000	1.02	16862.0000	1638	1.12	Y
opt1217	31	-19.3221	0.08	-16.0000	5235754	20.76%	N
p0033	22	2942.0000	0.11	3089.0000	75	0.01	Y
p0201	8	7125.0000	0.41	7615.0000	1099	0.02	Y
p0282	102	255636.0000	0.20	258411.0000	48	0.01	Y
p0548	147	8688.0000	0.03	8691.0000	0	0.01	Y
p2756	260	3120.0000	2.12	3124.0000	316	0.08	Y
pk1	0	0.0000	0.02	11.0000	529908	1.79	Y
pp08a	215	7210.6498	0.45	7350.0000	942	0.03	Y
pp08aCUTS	147	7205.6813	0.53	7350.0000	1028	0.03	Y
prod1	129	-81.3771	0.62	-56.0000	1442337	20.30	Y
prod2	128	-85.2228	2.86	-61.0000	2720687	18.39%	N
protfold	0	-41.0000	0.22	-	5	-	N
qap10	0	333.0000	0.62	358.0000	43	6.98%	N
qiu	0	-931.6389	0.05	-132.8731	15640	1.34	Y
qnet1	76	15438.7245	0.45	16029.6927	298	0.03	Y
qnet1_o	90	15624.5847	0.50	16030.9927	230	0.02	Y
ran10x26	179	4095.8019	1.02	4270.0000	28218	0.69	Y
ran12x21	199	3460.4177	6.20	3664.0000	74456	2.18	Y
ran13x13	188	3062.9010	0.86	3252.0000	54276	1.05	Y
rd-rplusc-21	606	100.0000	1308.59	-	25382	-	N
rentacar	30	29274325.2003	0.20	30356760.9841	25	0.03	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
rgn	101	79.9392	0.17	82.2000	1304	0.02	Y
rgna	0	48.8000	0.00	82.2000	2504	0.01	Y
roll3000	341	11486.5552	4.83	13428.0000	527802	11.84%	N
rout	29	982.1729	0.14	1077.5600	599083	9.59	Y
set1ch	424	54516.2870	0.56	54537.7500	106	0.02	Y
seymour	8	406.0000	1.42	434.0000	74405	6.22%	N
seymour1	16	405.4473	4.58	410.7637	25409	17.98	Y
sp97ar	282	653445845.1576	5.61	673207925.0400	62607	2.87%	N
stein27	7	13.0000	0.02	18.0000	4240	0.01	Y
stein45	0	22.0000	0.00	30.0000	62605	0.34	Y
stp3d	6	481.9510	211.38	-	12	-	N
swath	66	379.4261	688.06	507.0972	372678	25.11%	N
swath2	24	334.4969	1.53	385.1997	444677	41.53	Y
swath3	24	334.4969	1.70	399.6350	637990	11.39%	N
t1717	0	134532.0000	7.62	-	2505	-	N
timtab1	264	270007.9118	0.66	780218.0000	3750603	56.77%	N
timtab2	408	384296.5910	1.33	1239779.0000	2405659	65.42%	N
tr12-15	376	73533.1984	0.80	74634.0000	76847	1.59	Y
tr12-30	855	129026.2631	2.39	130600.0000	1291701	0.68%	N
tr24-15	745	135305.5117	2.36	136509.0000	1169048	46.59	Y
tr24-30	984	238708.3737	1.06	295201.0000	1223357	18.94%	N
tr6-15	223	36966.9071	0.28	37721.0000	8118	0.10	Y
tr6-30	366	60708.9797	0.70	61746.0000	3777388	0.19%	N
vpm1	62	20.0000	0.05	20.0000	3	0.01	Y
vpm2	157	13.0128	1.81	13.7500	15030	0.21	Y
vpm2a	135	13.1195	0.44	13.7500	7914	0.09	Y
vpm5	129	3002.7242	0.86	3003.3000	1491	0.11	Y

Table D.27.: Results for a 1-hour test with the cMIR cut and the cPMC generator. For the instance `momentum3`, MOPS returned an invalid result.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.27	924.0000	235	0.04	Y
30_05_100	0	9.0000	13.84	14.0000	180392	35.71%	N
30_95_100	1	3.0000	16.99	3.0000	3897	7.29	Y
30_95_98	0	12.0000	12.21	13.0000	85198	7.69%	N
a1c1s1	738	5883.2336	4.03	11910.9242	733784	45.76%	N
acc0	7	0.0000	0.41	0.0000	236	0.06	Y
acc1	14	0.0000	0.95	0.0000	1076	0.33	Y
acc2	9	0.0000	0.78	0.0000	4454	1.97	Y
acc3	0	0.0000	0.31	0.0000	4630	7.36	Y
acc4	0	0.0000	0.38	-	22201	-	N
acc5	0	0.0000	1.33	0.0000	3664	25.06	Y
aflow30a	208	1070.0000	2.66	1158.0000	40885	2.39	Y
aflow40b	344	1072.0000	30.86	1398.0000	70496	22.68%	N
air03	2	338864.2500	0.12	340160.0000	0	0.01	Y
air04	0	55536.0000	2.39	56138.0000	5473	2.45	Y
air05	0	25878.0000	0.89	26374.0000	16466	3.60	Y
arki001	76	7579849.3344	24.08	7581315.6319	1074265	0.02%	N
atlanta-ip	924	81.3039	58.72	-	3416	-	N
b4-10	176	13310.6454	1.95	14050.8397	37731	1.45	Y
b4-10b	105	13905.5206	4.28	14050.8397	510	0.12	Y
b4-12	279	14706.7246	1.97	16103.8837	691107	30.79	Y
b4-12b	108	15724.1846	9.42	16103.8837	39391	5.26	Y
b4-20b	100	22133.1330	11.38	23369.3847	90621	1.89%	N
BASF6-10	241	20962.7272	3.25	21267.5689	224330	38.74	Y
BASF6-5	233	11898.3140	2.12	12071.5772	68880	8.33	Y
bc1	34	2.5983	118.42	3.3384	10877	2.62%	N
bell3a	11	873172.1171	0.05	878430.3160	42135	0.12	Y
bell5	22	8918967.5473	0.06	8966406.4915	3278174	9.68	Y
bienst1	110	14.1033	0.76	46.7500	26664	1.72	Y
bienst2	127	14.9318	0.73	54.6000	257216	14.76	Y
binkar10_1	95	6702.1432	0.67	6742.2000	343205	9.84	Y
blend2	33	7.0991	0.64	7.5990	2902	0.04	Y
cap6000	7	-2451535.0000	0.16	-2451377.0000	37770	4.39	Y
ches1	60	73.7992	0.11	74.3405	0	0.01	Y
ches2	38	-2891.6578	0.11	-2889.6909	5758071	0.07%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
ches3	44	-1303896.9248	0.09	-1303896.9248	9	0.01	Y
ches4	36	-647403.5167	0.03	-647403.5167	4	0.01	Y
ches5	74	-7369.1212	0.05	-7342.8188	2096	0.03	Y
clorox	197	13579.2260	1.47	21217.8144	120	0.05	Y
Con-12	141	4670.3883	0.17	7593.3400	157020	2.75	Y
con-24	243	17972.6666	0.56	25804.9600	2016655	0.89%	N
dano3mip	382	576.5330	74.85	757.8400	4107	23.91%	N
dano3_3	47	576.2436	7.88	576.3964	9	1.55	Y
dano3_4	65	576.2535	2.24	576.4352	26	1.78	Y
dano3_5	131	576.2953	7.48	576.9452	260	5.33	Y
danoint	129	62.7159	0.78	65.6667	472242	4.02%	N
dcmulti	153	187342.6064	0.50	188182.0000	184	0.01	Y
disktom	0	-5000.0000	0.47	-	135946	-	N
dlsp	25	370.7681	0.41	613.0000	107254	2.19	Y
ds	0	57.2346	5.39	-	2212	-	N
dsbmip	80	-305.1982	0.16	-305.1982	36	0.02	Y
egout	23	567.6318	0.02	568.1007	0	0.01	Y
enigma	2	0.0000	0.00	0.0000	21176	0.04	Y
fast0507	2	173.0000	5.41	176.0000	12490	1.7%	N
fiber	84	388306.7843	0.69	405935.1800	164	0.01	Y
fixnet6	196	3753.1593	3.67	3983.0000	132	0.08	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0.01	Y
gen	44	112312.9529	0.09	112313.3627	0	0.01	Y
gesa2	187	25764287.6762	0.67	25779856.3717	3642	0.11	Y
gesa2_o	145	25611988.8505	0.42	25779856.3717	240493	5.16	Y
gesa3	146	27952285.0004	0.08	27991042.6484	236	0.02	Y
gesa3_o	158	27939906.2518	0.12	27991042.6484	167	0.02	Y
glass4	63	800002400.0000	0.09	1675016325.0000	4983728	52.24%	N
gt2	31	20726.0000	0.01	21166.0000	193	0.01	Y
harp2	103	-74231352.0000	2.36	-73899798.0000	1967668	31.49	Y
khb05250	112	106901882.2620	0.14	106940226.0000	16	0.01	Y
l152lav	0	4657.0000	0.16	4722.0000	10637	0.22	Y
liu	851	560.0000	0.61	1256.0000	481425	55.41%	N
lrn	788	44405303.0832	12.00	44479255.1273	340398	0.02%	N
lseu	34	1034.0000	0.14	1120.0000	785	0.01	Y
m20-75-1	512	-51216.6831	27.49	-49213.0000	272358	3.08%	N
m20-75-2	546	-52018.8768	48.21	-50322.0000	109119	24.74	Y
m20-75-3	659	-53268.2748	68.26	-51158.0000	269097	1.88%	N
m20-75-4	420	-54700.0284	9.84	-52752.0000	315228	57.37	Y
m20-75-5	504	-53080.0171	29.08	-51349.0000	94176	20.59	Y
manna81	0	-13297.0000	0.41	-13163.0000	876256	1.02%	N
markshare1	3	0.0000	0.22	8.0000	9999999	100%	N
markshare1_1	7	0.0000	0.02	0.0000	274158	0.55	Y
markshare2	6	0.0000	0.01	17.0000	9999999	100%	N
markshare2_1	13	0.0000	0.01	0.0000	6697817	19.69	Y
mas74	24	10579.8173	0.88	11801.1857	6791493	1.46%	N
mas76	23	39010.8237	0.83	40005.0541	1159854	8.13	Y
misc03	0	1910.0000	0.02	3360.0000	603	0.01	Y
misc06	17	12844.1486	0.08	12850.8607	184	0.02	Y
misc07	0	1415.0000	0.05	2810.0000	37474	0.56	Y
mitre	970	115107.0000	5.81	115155.0000	2488	0.35	Y
mkc	346	-605.9688	8.16	-511.7520	596643	18.2%	N
mod008	48	304.0000	0.69	307.0000	3398	0.07	Y
mod010	5	6535.0000	0.51	6548.0000	18	0.01	Y
mod011	772	-56633263.7842	4.38	-54558535.0142	2538	0.87	Y
modglob	176	20713903.3328	0.36	20740508.0863	234	0.01	Y
momentum1	507	96245.4753	52.94	-	439	-	N
momentum2	762	10698.2912	112.59	-	318	-	N
momentum3	3566	97254.0679	1962.64	-	22	-	N
msc98-ip	516	19702877.0058	16.38	-	3636	-	N
multiA	29	3512.7778	0.03	3774.7600	502086	8.38	Y
multiB	77	3627.4746	1.25	4059.7764	2172055	10.16%	N
multiC	81	1504.9114	0.52	2088.4200	2213356	24.16%	N
multiD	64	3813.8779	0.19	6117.4027	1715906	37.07%	N
multiE	176	2287.9500	0.53	2718.2050	3107198	12.03%	N
multiF	97	2054.0180	0.14	2429.4000	4373249	13.63%	N
mzzv11	152	-22655.0000	47.75	-21648.0000	77894	1.98%	N
mzzv42z	75	-21450.0000	56.19	-19308.0000	8812	9.03%	N
neos1	125	7.0000	0.45	19.0000	1886782	57.89%	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
neos10	81	-1182.0000	216.14	-1135.0000	40	3.71	Y
neos11	3	6.0000	0.69	9.0000	32624	17.82	Y
neos12	7	9.4116	0.64	13.0000	13196	15.89%	N
neos13	6	-126.1784	241.03	-95.0012	129381	23.59%	N
neos2	74	-4093.7435	12.28	454.8647	134309	9.46	Y
neos20	357	-474.8940	1.17	-434.0000	24823	1.11	Y
neos21	0	3.0000	0.06	7.0000	30130	2.41	Y
neos22	472	777191.4286	0.47	779715.0000	25974	1.58	Y
neos23	105	58.7023	0.80	137.0000	3489035	37.23%	N
neos3	75	-5781.7582	14.95	369.3519	827125	483.34%	N
neos4	0	-49463016984.6474	2.52	-48603440750.5898	1305	0.62	Y
neos5	0	13.0000	0.02	15.0000	7905334	3.33%	N
neos6	4	83.0000	3.33	83.0000	4721	7.38	Y
neos648910	367	16.0000	1.00	32.0000	90509	2.42	Y
neos671048	13	2999.0000	4.95	5001.0000	7246	5.47	Y
neos7	161	622879.9907	1.47	721934.0000	1054335	5.94%	N
neos8	23	-3725.0000	178.83	-3719.0000	0	3.00	Y
neos9	35	794.0000	13.92	798.0000	12846	0.5%	N
net12	452	78.0000	14.51	-	20860	-	N
noswot	15	-43.0000	0.02	-41.0000	8411531	4.88%	N
nsrand-ipx	305	50187.0000	0.91	51680.0000	165044	2.85%	N
nug08	0	204.0000	0.23	214.0000	151	0.10	Y
nw04	0	16311.0000	1.02	16862.0000	1638	1.13	Y
opt1217	27	-19.4900	0.08	-16.0000	4761545	21.81%	N
p0033	22	2942.0000	0.06	3089.0000	85	0.01	Y
p0201	8	7125.0000	0.42	7615.0000	1099	0.02	Y
p0282	109	255708.0000	0.22	258411.0000	713	0.01	Y
p0548	170	8691.0000	0.09	8691.0000	0	0.01	Y
p2756	250	3121.0000	0.97	3124.0000	422	0.07	Y
pk1	0	0.0000	0.00	11.0000	529908	1.79	Y
pp08a	205	7164.5789	0.47	7350.0000	1396	0.03	Y
pp08aCUTS	105	7188.4437	0.19	7350.0000	1250	0.02	Y
prod1	137	-81.3751	0.88	-56.0000	3974094	11.27%	N
prod2	130	-85.2228	2.98	-62.0000	2601440	9.84%	N
protfold	0	-41.0000	0.22	-	5	-	N
qap10	0	333.0000	0.61	358.0000	43	6.98%	N
qiu	0	-931.6389	0.06	-132.8731	15640	1.35	Y
qnet1	72	15274.7876	0.48	16029.6927	280	0.03	Y
qnet1_o	90	15624.5847	0.51	16030.9927	230	0.02	Y
ran10x26	234	4100.0864	0.91	4270.0000	26752	0.89	Y
ran12x21	244	3453.9374	1.41	3664.0000	91567	2.63	Y
ran13x13	214	3040.6160	0.95	3252.0000	41938	0.87	Y
rd-rplusc-21	219	100.0000	363.95	-	1172	-	N
rentacar	47	29232562.5002	0.22	30356760.9841	25	0.06	Y
rgn	120	76.7749	0.33	82.2000	1496	0.03	Y
rgna	0	48.8000	0.00	82.2000	2504	0.01	Y
roll3000	398	12174.3206	5.41	13347.0000	497942	8.79%	N
rout	36	982.1729	1.11	1077.5600	584574	9.57	Y
set1ch	362	54523.2518	0.42	54537.7500	47	0.02	Y
seymour	8	406.0000	1.52	434.0000	74178	6.22%	N
seymour1	4	404.6459	1.75	410.7919	47923	33.30	Y
sp97ar	253	653445845.1576	5.52	672355872.3000	67765	2.72%	N
stein27	7	13.0000	0.02	18.0000	4240	0.01	Y
stein45	0	22.0000	0.01	30.0000	62605	0.34	Y
stp3d	6	481.9510	214.52	-	12	-	N
swath	33	335.1868	33.30	-	262616	-	N
swath2	19	334.4969	1.53	385.1997	421028	38.14	Y
swath3	19	334.4969	1.70	399.8501	656488	12.35%	N
t1717	0	134532.0000	7.73	-	2493	-	N
timtab1	279	272473.6428	0.67	799106.0000	3903111	55.21%	N
timtab2	468	372453.0933	2.30	-	2363791	-	N
tr12-15	353	73846.0438	0.81	74634.0000	14792	0.31	Y
tr12-30	818	130150.1161	2.39	130596.0000	1337346	50.40	Y
tr24-15	673	136179.4765	2.59	136509.0000	36830	1.33	Y
tr24-30	984	238527.8255	1.19	294061.0000	1254242	18.88%	N
tr6-15	195	37217.7194	0.14	37721.0000	5112	0.06	Y
tr6-30	359	60934.6510	0.97	61746.0000	2878575	44.43	Y
vpm1	36	20.0000	0.03	20.0000	0	0.01	Y
vpm2	141	12.9249	0.38	13.7500	25148	0.26	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
vpm2a	137	13.0685	0.67	13.7500	10248	0.13	Y
vpm5	113	3002.6463	6.72	3003.2000	89	0.12	Y

Table D.28.: Results for a 1-hour test with the SOTA configuration.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.26	924.0000	235	0.04	Y
30_05_100	0	9.0000	14.06	14.0000	180475	35.71%	N
30_95_100	1	3.0000	16.97	3.0000	3897	7.27	Y
30_95_98	0	12.0000	12.16	13.0000	85289	7.69%	N
a1c1s1	785	6143.6765	3.38	11684.7273	597048	39.86%	N
acc0	7	0.0000	0.39	0.0000	236	0.06	Y
acc1	14	0.0000	0.94	0.0000	1076	0.33	Y
acc2	9	0.0000	0.70	0.0000	4454	1.97	Y
acc3	0	0.0000	0.31	0.0000	4630	7.35	Y
acc4	0	0.0000	0.33	-	22212	-	N
acc5	0	0.0000	1.27	0.0000	3664	25.04	Y
aflow30a	210	1079.0000	1.31	1158.0000	25519	1.34	Y
aflow40b	387	1085.0000	14.31	1448.0000	63223	24.31%	N
air03	2	338864.2500	0.12	340160.0000	0	0.01	Y
air04	0	55536.0000	2.39	56138.0000	5473	2.45	Y
air05	0	25878.0000	0.86	26374.0000	16466	3.59	Y
arki001	140	7579880.1818	14.91	-	1189	-	N
atlanta-ip	969	81.3034	61.00	-	372	-	N
b4-10	202	13346.1515	1.52	-	799639	-	N
b4-10b	136	13979.8264	4.00	14050.8397	368	0.09	Y
b4-12	290	14756.7958	1.72	16103.8837	1383965	0.46%	N
b4-12b	212	15854.5963	11.17	16103.8837	2282	0.55	Y
b4-20b	335	22466.6166	29.59	-	54547	-	N
BASF6-10	217	20963.1859	2.61	21267.5689	59017	8.65	Y
BASF6-5	206	11898.2924	1.47	12071.5772	25296	3.19	Y
bc1	67	2.5955	114.36	3.3663	10393	3.31%	N
bell3a	16	873196.5787	0.08	878430.3160	45716	0.13	Y
bell5	29	8922311.1807	0.06	8966406.4915	2145218	6.39	Y
bienst1	100	14.0824	0.62	46.7500	39019	2.31	Y
bienst2	122	14.9288	0.62	54.6000	251493	12.31	Y
binkar10_1	95	6702.1432	0.67	6742.2000	318139	9.21	Y
blend2	37	7.0952	0.59	7.5990	2997	0.04	Y
cap6000	7	-2451535.0000	0.16	-2451377.0000	37770	4.39	Y
ches1	102	73.8056	0.55	74.3405	20	0.01	Y
ches2	66	-2891.6536	0.17	-2889.5569	2758190	32.56	Y
ches3	28	-1303896.9248	0.03	-1303896.9248	13	0	Y
ches4	32	-647403.5167	0.02	-647403.5167	13	0	Y
ches5	78	-7370.5310	0.09	-7342.8188	7948	0.14	Y
clorox	165	17303.6051	0.30	21218.8920	212	0.02	Y
Con-12	197	4588.7899	0.53	7593.1000	184070	3.43	Y
con-24	291	18036.2928	0.42	25804.9600	1971078	1.72%	N
dano3mip	540	576.5720	51.55	758.0000	5952	23.92%	N
dano3_3	32	576.2384	0.88	576.3964	9	1.77	Y
dano3_4	69	576.2556	3.11	576.4352	17	2.12	Y
dano3_5	135	576.3037	10.84	576.9249	200	4.81	Y
danoint	113	62.6996	0.66	65.6667	507242	4.39%	N
dcmulti	146	187470.6083	0.44	188182.0000	271	0.01	Y
disktom	0	-5000.0000	0.47	-	136143	-	N
dlsp	31	375.3100	0.42	613.0000	103209	2.19	Y
ds	0	57.2346	5.56	-	2218	-	N
dsbmip	101	-305.1982	0.11	-305.1982	45	0.02	Y
egout	22	567.9932	0.03	568.1007	0	0	Y
enigma	2	0.0000	0.00	0.0000	21176	0.04	Y
fast0507	2	173.0000	5.41	176.0000	12490	1.7%	N
fiber	84	388306.7843	0.70	405935.1800	164	0.02	Y
fixnet6	206	3807.4818	1.78	3983.0000	128	0.05	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0	Y
gen	44	112312.9529	0.09	112313.3627	0	0	Y
gesa2	165	25771293.5544	0.62	25779856.3717	340	0.02	Y
gesa2_o	209	25775432.4183	0.69	25779856.3717	230	0.02	Y
gesa3	192	27973351.6108	0.95	27991042.6484	48	0.03	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
gesa3_o	234	27963406.7044	0.91	27991042.6484	93	0.03	Y
glass4	63	800002400.0000	0.11	1675016325.0000	4985255	52.24%	N
gt2	31	20726.0000	0.02	21166.0000	193	0	Y
harp2	103	-74231352.0000	2.31	-73899798.0000	1967668	31.51	Y
khb05250	124	106915722.2610	0.58	106940226.0000	22	0.02	Y
l152lav	0	4657.0000	0.16	4722.0000	10637	0.22	Y
liu	834	560.0000	1.14	1360.0000	555019	58.82%	N
lrn	918	44545173.2447	11.53	44688241.3258	297668	0.05%	N
lseu	34	1034.0000	0.14	1120.0000	785	0	Y
m20-75-1	389	-51161.8537	6.31	-49113.0000	300838	3.35%	N
m20-75-2	605	-52005.4722	62.11	-50322.0000	68392	16.76	Y
m20-75-3	653	-53238.0531	118.67	-51158.0000	262537	2.79%	N
m20-75-4	436	-54665.1474	67.38	-52752.0000	307130	0.8%	N
m20-75-5	523	-53017.6993	16.14	-51349.0000	163488	33.94	Y
manna81	0	-13297.0000	0.41	-13163.0000	883667	1.02%	N
markshare1	3	0.0000	0.06	8.0000	9999999	100%	N
markshare1_1	7	0.0000	0.01	0.0000	823388	1.71	Y
markshare2	6	0.0000	0.01	17.0000	9999999	100%	N
markshare2_1	11	0.0000	0.06	0.0000	7287843	19.98	Y
mas74	25	10583.2346	0.91	11801.1857	6709642	3.51%	N
mas76	23	39010.8237	0.83	40005.0541	1159854	8.13	Y
misc03	0	1910.0000	0.02	3360.0000	603	0	Y
misc06	29	12846.2683	0.11	12851.0763	19	0.01	Y
misc07	0	1415.0000	0.05	2810.0000	37474	0.56	Y
mitre	970	115107.0000	5.88	115155.0000	2488	0.35	Y
mkc	346	-605.9688	8.27	-511.7520	598235	18.2%	N
mod008	48	304.0000	0.70	307.0000	3398	0.07	Y
mod010	5	6535.0000	0.52	6548.0000	18	0.01	Y
mod011	823	-56654368.7838	4.41	-54558535.0142	4280	1.43	Y
modglob	166	20728790.8264	0.27	20740508.0863	64	0.01	Y
momentum1	690	96249.1959	46.55	-	98	-	N
momentum2	1158	10722.4703	100.28	-	76	-	N
momentum3	4558	94945.1449	3442.72	-	1	-	N
msc98-ip	470	19695288.0058	17.53	-	2794	-	N
multiA	73	3558.1537	0.05	3774.7600	332012	6.86	Y
multiB	112	3629.2792	1.77	3995.5200	1759362	8.83%	N
multiC	56	1491.5315	0.28	2088.4200	2360085	25.16%	N
multiD	77	3832.1012	0.08	6161.7000	1675797	36.55%	N
multiE	255	2298.9004	0.51	2710.5925	2359011	11.89%	N
multiF	205	2058.5601	0.44	2428.9300	2865979	13.82%	N
mzzv11	103	-22689.0000	55.28	-19040.0000	45645	17.66%	N
mzzv42z	75	-21450.0000	56.30	-19308.0000	8873	9.03%	N
neos1	125	7.0000	0.44	19.0000	1892071	57.89%	N
neos10	81	-1182.0000	214.66	-1135.0000	40	3.69	Y
neos11	10	6.0000	0.72	9.0000	30044	16.5	Y
neos12	6	9.4116	0.73	13.0000	11801	16.22%	N
neos13	6	-126.1784	42.47	-87.0062	137894	45.02%	N
neos2	110	-3986.4225	11.42	454.8647	132551	11.31	Y
neos20	357	-474.8940	0.94	-434.0000	24823	1.1	Y
neos21	0	3.0000	0.06	7.0000	30130	2.41	Y
neos22	237	779485.8333	0.36	779715.0000	48	0.03	Y
neos23	136	59.3098	0.95	137.0000	3301133	38.69%	N
neos3	174	-5674.9374	15.78	368.8428	788559	304.2%	N
neos4	0	-49463016984.6474	2.50	-48603440750.5898	1305	0.62	Y
neos5	0	13.0000	0.02	15.0000	7936097	3.33%	N
neos6	4	83.0000	3.31	83.0000	4721	7.35	Y
neos648910	364	16.0000	0.69	32.0000	2056753	50%	N
neos671048	4	2999.0000	2.74	5001.0000	15868	13.59	Y
neos7	291	688625.5509	1.52	721934.0000	694868	54.71	Y
neos8	23	-3725.0000	178.80	-3719.0000	0	3	Y
neos9	35	794.0000	13.53	798.0000	12993	0.5%	N
net12	467	78.0000	14.80	-	40820	-	N
noswot	14	-43.0000	0.03	-40.0000	9624835	7.5%	N
nsrand-ipx	305	50187.0000	0.92	51680.0000	165515	2.85%	N
nug08	0	204.0000	0.23	214.0000	151	0.1	Y
nw04	0	16311.0000	1.02	16862.0000	1638	1.13	Y
opt1217	31	-19.3221	0.08	-16.0000	5035540	20.76%	N
p0033	22	2942.0000	0.42	3089.0000	85	0.02	Y
p0201	8	7125.0000	0.39	7615.0000	1099	0.02	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
p0282	109	255708.0000	0.25	258411.0000	713	0.01	Y
p0548	170	8691.0000	0.06	8691.0000	0	0.01	Y
p2756	250	3121.0000	0.98	3124.0000	422	0.06	Y
pk1	0	0.0000	0.00	11.0000	529908	1.78	Y
pp08a	227	7207.0073	0.39	7350.0000	910	0.02	Y
pp08aCUTS	151	7219.4614	0.41	7350.0000	1250	0.03	Y
prod1	137	-81.3751	0.83	-56.0000	3982789	11.27%	N
prod2	130	-85.2228	2.94	-62.0000	2608981	9.84%	N
protfold	0	-41.0000	0.22	-	5	-	N
qap10	0	333.0000	0.59	358.0000	43	6.98%	N
qiu	0	-931.6389	0.06	-132.8731	15640	1.34	Y
qnet1	76	15438.7245	0.47	16029.6927	298	0.03	Y
qnet1_o	90	15624.5847	0.50	16030.9927	230	0.02	Y
ran10x26	218	4094.8122	1.01	4270.0000	40140	1.15	Y
ran12x21	264	3462.6667	1.75	3664.0000	50435	1.64	Y
ran13x13	211	3065.7297	0.77	3252.0000	34533	0.75	Y
rd-rplusc-21	314	100.0000	603.44	-	21878	-	N
rentacar	24	29274325.2003	0.58	30356760.9841	32	0.04	Y
rgn	110	81.8363	0.25	82.2000	354	0.01	Y
rgna	0	48.8000	0.00	82.2000	2504	0	Y
roll3000	377	11512.1280	6.02	13240.0000	478480	11.44%	N
rout	39	982.1729	0.58	1077.5600	824442	16.04	Y
set1ch	439	54476.1056	0.31	54537.7500	128	0.02	Y
seymour	8	406.0000	1.53	434.0000	74222	6.22%	N
seymour1	16	405.4473	4.31	410.7637	25409	18.05	Y
sp97ar	253	653445845.1576	5.34	672355872.3000	67795	2.72%	N
stein27	7	13.0000	0.01	18.0000	4240	0.01	Y
stein45	0	22.0000	0.02	30.0000	62605	0.34	Y
stp3d	6	481.9510	214.42	-	12	-	N
swath	66	379.2683	305.02	494.7899	452319	23.33%	N
swath2	19	334.4969	1.55	385.1997	421028	38.1	Y
swath3	19	334.4969	1.70	399.8501	657364	12.35%	N
t1717	0	134532.0000	7.92	-	2497	-	N
timtab1	270	255646.2133	0.55	792722.0000	4033675	65.78%	N
timtab2	401	377197.6991	1.42	-	2642857	-	N
tr12-15	374	73493.5888	0.70	74634.0000	48741	1.06	Y
tr12-30	874	129037.6381	2.09	130596.0000	1284495	0.55%	N
tr24-15	795	135210.2040	2.17	136509.0000	1405248	0.26%	N
tr24-30	984	237510.7790	1.03	295262.0000	1224750	19.31%	N
tr6-15	224	37185.7433	0.22	37721.0000	5582	0.07	Y
tr6-30	382	60745.2225	0.44	61746.0000	3675519	0.21%	N
vpm1	41	20.0000	0.03	20.0000	0	0	Y
vpm2	170	12.9692	0.30	13.7500	20559	0.23	Y
vpm2a	129	13.0633	0.25	13.7500	9948	0.1	Y
vpm5	134	3002.7436	0.81	3003.3000	1165	0.09	Y

Table D.29.: Results for a 1-hour test with the improved SOTA configuration but without flow path cuts.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
10teams	4	924.0000	0.27	924.0000	820	0.11	Y
30_05_100	0	9.0000	13.55	14.0000	180834	35.71%	N
30_95_100	1	3.0000	16.80	3.0000	42589	41.22	Y
30_95_98	0	12.0000	11.92	13.0000	85610	7.69%	N
a1c1s1	756	5986.2630	1.80	11931.4245	730741	47.53%	N
acc0	9	0.0000	0.33	0.0000	298	0.06	Y
acc1	6	0.0000	0.45	0.0000	209	0.08	Y
acc2	11	0.0000	0.81	0.0000	1172	0.56	Y
acc3	0	0.0000	0.24	0.0000	4630	7.34	Y
acc4	0	0.0000	0.25	-	22238	-	N
acc5	0	0.0000	1.17	0.0000	3664	24.97	Y
aflow30a	213	1061.0000	1.02	1158.0000	58210	3.07	Y
aflow40b	279	1072.0000	10.70	1243.0000	81424	10.38%	N
air03	2	338864.2500	0.12	340160.0000	0	0.01	Y
air04	0	55536.0000	2.36	56138.0000	5473	2.45	Y
air05	0	25878.0000	0.84	26374.0000	16466	3.59	Y
arki001	148	7579800.4059	5.26	-	290	-	N

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
atlanta-ip	792	81.2902	43.17	-	208	-	N
b4-10	140	13262.8578	0.69	14050.8397	7525	0.26	Y
b4-10b	68	13912.3120	1.75	14050.8397	769	0.09	Y
b4-12	211	14594.6857	0.91	16103.8837	1345886	3.92%	N
b4-12b	94	15702.9874	2.16	16103.8837	11057	1.50	Y
b4-20b	79	22116.4441	7.78	24357.9852	90490	7.43%	N
BASF6-10	179	20958.3110	3.62	21268.0456	410554	0.08%	N
BASF6-5	177	11892.3102	2.42	12071.5772	32508	4.67	Y
bc1	34	2.5906	46.56	3.3665	10805	2.83%	N
bell3a	12	873351.5153	0.02	878430.3160	40577	0.12	Y
bell5	24	8918901.7504	0.03	8966406.4915	103482	0.26	Y
bienst1	98	14.0719	0.38	46.7500	30310	2.20	Y
bienst2	122	14.9408	0.36	54.6000	203634	9.98	Y
binkar10_1	47	6666.5748	0.03	6742.8000	2787816	0.6%	N
blend2	41	7.1597	0.33	7.5990	1985	0.03	Y
cap6000	7	-2451535.0000	0.09	-2451377.0000	37770	4.38	Y
ches1	30	72.9641	0.05	74.3405	18	0.01	Y
ches2	37	-2891.6621	0.08	-2889.6786	5290334	0.07%	N
ches3	43	-1303896.9248	0.03	-1303896.9248	9	0.01	Y
ches4	35	-647403.5167	0.03	-647403.5167	9	0.01	Y
ches5	96	-7371.6793	0.09	-7342.8188	2423	0.04	Y
clorox	235	13579.7651	0.38	21217.8144	186	0.02	Y
Con-12	151	3936.7016	0.09	7593.0700	684498	9.10	Y
con-24	228	15864.1531	0.20	25839.0200	2286701	5.25%	N
dano3mip	21	576.2906	41.23	801.5556	6553	28.09%	N
dano3_3	4	576.2325	3.03	576.3964	9	1.38	Y
dano3_4	6	576.2326	0.78	576.4352	26	1.51	Y
dano3_5	7	576.2327	0.88	576.9249	230	4.22	Y
danoint	91	62.7003	0.41	65.6667	537584	3.96%	N
dcmulti	120	186852.1775	0.17	188182.0000	284	0.01	Y
disktom	0	-5000.0000	0.45	-	136183	-	N
dlsp	24	373.4638	0.17	613.0000	79257	1.52	Y
ds	0	57.2346	5.61	-	2222	-	N
dsbmip	64	-305.1982	0.05	-305.1982	56	0.01	Y
egout	23	567.4596	0.02	568.1007	0	0.01	Y
enigma	2	0.0000	0.00	0.0000	22368	0.04	Y
fast0507	2	173.0000	5.38	177.0000	2889	2.26%	N
fiber	107	383707.1861	0.36	405935.1800	260	0.01	Y
fixnet6	158	3661.2995	0.58	3983.0000	82	0.02	Y
flugpl	0	1167185.7256	0.00	1201500.0000	141	0.01	Y
gen	30	112312.5959	0.00	112313.3627	0	0.01	Y
gesa2	149	25670385.9950	0.36	25779856.3717	2616	0.06	Y
gesa2_o	165	25590653.7169	0.36	25779856.3717	456850	8.97	Y
gesa3	176	27960739.3395	0.25	27991042.6484	154	0.02	Y
gesa3_o	150	27938654.5304	0.36	27991042.6484	283	0.01	Y
glass4	430	800003554.9149	0.24	1750015220.0000	5299462	54.29%	N
gt2	49	20050.0000	0.02	21166.0000	2002	0.01	Y
harp2	154	-74080227.0000	2.59	-73872399.4600	2081271	47.67	Y
khb05250	89	106786405.2814	0.31	106940226.0000	24	0.02	Y
l152lav	0	4657.0000	0.14	4722.0000	17798	0.38	Y
liu	781	560.0000	0.16	1362.0000	491640	58.88%	N
lrn	1032	44420488.5128	8.23	44497156.9440	284047	0.07%	N
lseu	27	1035.0000	0.14	1120.0000	856	0.01	Y
m20-75-1	395	-51188.3495	0.26	-49113.0000	310110	3.01%	N
m20-75-2	494	-52096.9781	0.47	-50322.0000	144516	29.25	Y
m20-75-3	731	-53218.3058	4.62	-51158.0000	265760	2.76%	N
m20-75-4	463	-54681.0879	2.12	-52752.0000	306050	1.36%	N
m20-75-5	574	-53019.2220	1.14	-51349.0000	49692	10.90	Y
manna81	0	-13297.0000	0.25	-13163.0000	882747	1.02%	N
markshare1	2	0.0000	0.36	5.0000	9999999	100%	N
markshare1_1	9	0.0000	0.01	0.0000	819831	1.72	Y
markshare2	2	0.0000	0.00	21.0000	9999999	100%	N
markshare2_1	11	0.0000	0.02	0.0019	9999999	100%	N
mas74	23	10576.3415	0.23	11801.1857	6700750	1.24%	N
mas76	24	39014.1735	0.24	40005.0541	737878	5.28	Y
misc03	0	1910.0000	0.00	3360.0000	603	0.01	Y
misc06	8	12844.1977	0.03	12850.8607	89	0.01	Y
misc07	0	1415.0000	0.05	2810.0000	37474	0.56	Y
mitre	943	115091.0000	4.73	115155.0000	7164	0.82	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
mkc	98	-611.8247	1.25	-542.6260	844386	12.55%	N
mod008	23	298.0000	0.01	307.0000	5356	0.05	Y
mod010	3	6535.0000	0.48	6548.0000	18	0.01	Y
mod011	738	-57302072.4948	2.95	-54558535.0142	3174	0.97	Y
modglob	182	20707301.9002	0.14	20740508.0863	2748	0.03	Y
momentum1	480	93691.5982	32.16	-	478	-	N
momentum2	678	10696.9097	66.69	-	196	-	N
momentum3	2303	91961.4444	348.80	-	93	-	N
msc98-ip	486	19699455.1058	17.48	-	3493	-	N
multiA	29	3512.7778	0.02	3774.7600	338016	6.02	Y
multiB	53	3625.8934	0.08	4003.1800	2161513	9.05%	N
multiC	43	1487.5455	0.06	2083.2867	2403226	25.63%	N
multiD	96	3844.7799	0.33	6178.0000	1658401	37.18%	N
multiE	175	2283.5447	0.16	2720.0150	3010863	14.02%	N
multiF	171	2057.0161	0.17	2428.9300	3289482	12.68%	N
mzzv11	115	-22685.0000	63.86	-	48854	-	N
mzzv42z	53	-21450.0000	51.30	-19478.0000	3943	8.17%	N
neos1	499	11.0000	1.38	19.0000	1212893	31.58%	N
neos10	168	-1177.0000	259.86	-1135.0000	24	4.42	Y
neos11	3	6.0000	2.41	9.0000	31680	19.08	Y
neos12	2	9.4116	0.51	13.0000	10467	12.89%	N
neos13	7	-126.1784	19.17	-89.0612	133548	41.68%	N
neos2	73	-3922.4872	11.05	454.8697	123787	9.67	Y
neos20	146	-475.0000	0.27	-434.0000	293336	10.57	Y
neos21	0	3.0000	0.03	7.0000	30130	2.40	Y
neos22	472	777191.4286	0.17	779715.0000	27270	1.66	Y
neos23	62	58.7023	0.24	137.0000	3788197	44.53%	N
neos3	83	-5721.9596	14.22	368.8428	616757	174.53%	N
neos4	15	-49456451695.1585	5.62	-48603440750.5898	842	0.36	Y
neos5	0	13.0000	0.02	15.0000	7940849	3.33%	N
neos6	4	83.0000	3.34	83.0000	4721	7.34	Y
neos648910	394	16.0000	0.30	32.0000	215300	4.99	Y
neos671048	0	2001.0000	1.76	5001.0000	36178	32.37	Y
neos7	181	631874.7339	0.73	721934.0000	1093507	4.75%	N
neos8	22	-3725.0000	177.08	-3719.0000	0	2.97	Y
neos9	42	794.0000	12.19	798.0000	18535	0.5%	N
net12	419	78.0000	10.81	-	20903	-	N
noswot	13	-43.0000	0.03	-41.0000	9187538	4.88%	N
nsrand-ipx	279	50230.0000	0.86	52640.0000	132061	4.58%	N
nug08	0	204.0000	0.19	214.0000	151	0.10	Y
nw04	0	16311.0000	0.95	16862.0000	1638	1.13	Y
opt1217	13	-19.3076	0.01	-16.0000	5678297	20.67%	N
p0033	16	2917.0000	0.08	3089.0000	61	0.02	Y
p0201	8	7125.0000	0.39	7615.0000	999	0.02	Y
p0282	96	255872.0000	0.12	258411.0000	51	0.01	Y
p0548	156	8670.0000	0.28	8691.0000	163	0.01	Y
p2756	281	3119.0000	2.25	3124.0000	381	0.08	Y
pk1	0	0.0000	0.02	11.0000	529908	1.78	Y
pp08a	217	7157.7096	0.17	7350.0000	2612	0.04	Y
pp08aCUTS	160	7195.3451	0.20	7350.0000	1576	0.03	Y
prod1	27	-81.5018	0.27	-56.0000	1760454	15.77	Y
prod2	45	-85.2768	1.06	-62.0000	3603022	5.31%	N
protfold	0	-41.0000	0.16	-	5	-	N
qap10	0	333.0000	0.53	358.0000	43	6.98%	N
qiu	0	-931.6389	0.08	-132.8731	16190	1.21	Y
qnet1	87	15716.7009	0.52	16030.9927	114	0.03	Y
qnet1_o	83	15544.8601	0.28	16029.6927	257	0.02	Y
ran10x26	202	4063.4891	0.59	4270.0000	66827	2.12	Y
ran12x21	233	3442.4730	0.95	3664.0000	133578	4.09	Y
ran13x13	193	3047.1686	0.61	3252.0000	52535	1.35	Y
rd-rplusc-21	234	100.0000	33.03	-	24207	-	N
rentacar	51	29203605.8781	0.11	30356760.9841	20	0.03	Y
rgn	194	78.0074	0.25	82.2000	1484	0.02	Y
rgna	0	48.8000	0.00	82.2000	2504	0.01	Y
roll3000	389	12018.4998	4.27	13241.0000	418915	8.88%	N
rout	23	982.1729	0.17	1077.5600	624964	9.58	Y
set1ch	376	54521.6938	0.25	54537.7500	26	0.01	Y
seymour	10	406.0000	1.03	434.0000	65011	6.22%	N
seymour1	3	404.5658	0.36	410.7637	64017	42.05	Y

continued on the next page

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
sp97ar	265	653464964.5058	7.05	680733250.5000	53673	4.01%	N
stein27	7	13.0000	0.00	18.0000	4240	0.01	Y
stein45	0	22.0000	0.00	30.0000	62605	0.34	Y
stp3d	6	481.9510	217.02	-	12	-	N
swath	42	342.3662	60.06	522.6150	449859	33.4%	N
swath2	21	334.6370	3.03	385.1997	429314	38.10	Y
swath3	13	334.4969	1.70	400.2202	617880	11.43%	N
t1717	0	134532.0000	7.56	-	2513	-	N
timtab1	275	278086.5075	0.22	788365.0000	3753418	58.04%	N
timtab2	407	325584.7630	0.66	-	2828682	-	N
tr12-15	409	73406.2583	0.42	74634.0000	220955	5.81	Y
tr12-30	891	129108.0395	1.19	130600.0000	980827	0.81%	N
tr24-15	839	135199.3995	1.25	136538.0000	1148823	0.41%	N
tr24-30	984	239198.4771	0.70	296045.0000	1156157	18.98%	N
tr6-15	198	37018.4489	0.17	37721.0000	10044	0.11	Y
tr6-30	365	60675.8455	0.08	61746.0000	3246973	0.39%	N
vpm1	60	20.0000	0.00	20.0000	0	0.01	Y
vpm2	128	12.9493	0.42	13.7500	30910	0.34	Y
vpm2a	87	12.9431	0.05	13.7500	32617	0.27	Y
vpm5	99	3002.6108	0.55	3003.2000	28	0.01	Y

Table D.30.: Results for a 1-hour test with the OLD configuration.

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
mopsMIB001	27	101861.4029	4.23	114594.0000	264	10.85%	N
mopsMIB002	779	1417130.2195	169.33	-	3038	-	N
mopsMIB003	324	1841585.2575	14.67	2526650.5000	299238	26.66%	N
mopsMIB004	476	3596694.4652	70.75	-	24605	-	N
mopsMIB005	223	-18602919.6633	510.27	36464706.4948	10	149.99%	N
mopsMIB006	1643	4677977035.0694	124.11	4798143828.2267	340	2.21%	N
mopsMIB007	1809	71712307.6923	55.05	71716621.1059	39102	0.01%	N
mopsMIB008	26904	-58325.7050	2469.55	-58325.6924	4	59.61	Y
mopsMIB009	2609	74278213.1923	28.31	-	57310	-	N
mopsMIB010	4597	56000492.2175	66.20	-	18222	-	N
mopsMIB011	1665	40983605.1066	15.61	-	92459	-	N
mopsMIB012	3721	-60715398.4461	416.78	-	10351	-	N
mopsMIB013	78	-961717.3608	2.34	-959180.7489	607193	0.21%	N
mopsMIB014	91	-962356.0230	1.14	-961513.0374	566245	0.02%	N
mopsMIB015	1348	5306.4191	0.09	6361.5801	1195860	8.75%	N
mopsMIB016	977	380.6941	141.83	-	8256	-	N
mopsMIP001	18	11601.7517	0.06	20675.0000	327091	0.90	Y
mopsMIP002	608	5494.6720	0.55	6521.2513	2004481	15.5%	N
mopsMIP003	48	2694.5429	0.73	2726.9766	208471	1.28	Y
mopsMIP004	131	1665.0000	1.39	1854.0000	1690031	9.76%	N
mopsMIP005	1603	1864.5158	6.03	2536.8133	228664	37.99	Y
mopsMIP006	1463	80.7753	11.11	208.6757	1530	0.52	Y
mopsMIP007	160	278.7540	0.45	694.0186	6682950	10.75%	N
mopsMIP008	483	1533.0094	2.19	4649.3997	1208131	67.02%	N
mopsMIP009	1009	7198.3547	9.08	47846.3633	23089	84.9%	N

Table D.31.: Results for a 1-hour test with the improved SOTA configuration (MOPSLIB test set).

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
mopsMIB001	27	101861.4029	2.83	114594.0000	261	10.85%	N
mopsMIB002	829	1416118.3396	168.80	-	4211	-	N
mopsMIB003	343	1842193.6192	15.58	2528110.5000	305962	26.84%	N
mopsMIB004	476	3596694.4652	68.56	-	24090	-	N
mopsMIB005	223	-18601587.6662	442.66	36464706.4948	12	149.98%	N
mopsMIB006	1643	4677977035.0694	114.67	4798143828.2267	340	2.21%	N
mopsMIB007	1785	71712308.8534	60.91	71716249.4508	31508	0.01%	N
mopsMIB008	26431	-58325.6140	2337.95	-58325.6014	4	57.57	Y
mopsMIB009	2455	74278367.7008	26.91	-	66554	-	N
mopsMIB010	4363	56001366.3554	64.23	-	7499	-	N
mopsMIB011	1689	40981283.8792	15.25	-	112484	-	N
mopsMIB012	3721	-60715398.4461	410.47	-	10251	-	N
mopsMIB013	78	-961717.3608	2.08	-959180.7489	605407	0.21%	N
mopsMIB014	91	-962356.0230	1.08	-961513.0374	564190	0.02%	N
mopsMIB015	1348	5306.4191	0.11	6365.6621	1231221	8.48%	N
mopsMIB016	963	383.2553	137.44	563.7795	11868	31.77%	N
mopsMIP001	18	11601.7517	0.05	20675.0000	326346	0.90	Y
mopsMIP002	608	5494.6720	0.53	6521.2513	1966976	15.5%	N
mopsMIP003	48	2694.5429	0.97	2726.9766	208471	1.30	Y
mopsMIP004	125	1666.0000	1.23	1842.0000	1200205	9.34%	N
mopsMIP005	1603	1864.5158	5.89	2536.8133	185810	31.17	Y
mopsMIP006	1435	80.7627	11.00	208.6757	1314	0.48	Y
mopsMIP007	160	278.7540	0.45	694.0186	6669866	10.75%	N
mopsMIP008	483	1533.0094	2.09	4649.3997	1248293	67.02%	N
mopsMIP009	943	7118.0502	7.02	29764.7672	164350	76.09%	N

Table D.32.: Results for a 1-hour test with the improved SOTA configuration but without flow cover cuts (MOPSLIB test set).

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
mopsMIB001	19	101861.6784	4.25	109562.0000	308	6.54%	N
mopsMIB002	1139	1438465.4557	109.14	-	4918	-	N
mopsMIB003	247	1840184.8567	10.08	2524630.5000	283571	26.74%	N
mopsMIB004	495	3604085.1196	56.86	-	38399	-	N
mopsMIB005	226	-19242051.8639	335.47	36464706.4948	12	151.84%	N
mopsMIB006	171	4677563372.5603	11.09	4798378828.2267	574	2.12%	N
mopsMIB007	1619	71713827.8174	43.45	71713837.8803	27445	0%	N
mopsMIB008	23953	-58325.7050	2077.74	-58325.6924	4	53.24	Y
mopsMIB009	1229	74267667.9657	24.05	-	78896	-	N
mopsMIB010	1902	56014795.5920	65.70	-	26432	-	N
mopsMIB011	1034	40984764.1660	17.64	-	133874	-	N
mopsMIB012	3434	-60715466.8691	408.83	-	9694	-	N
mopsMIB013	75	-961718.8721	2.09	-959090.5222	584542	0.22%	N
mopsMIB014	76	-962412.7350	1.33	-961513.0374	578769	0.02%	N
mopsMIB015	1348	5313.3296	0.08	6383.4203	1242773	8.27%	N
mopsMIB016	977	380.6941	143.08	-	8293	-	N
mopsMIP001	22	11601.7017	0.05	20675.0000	229367	0.63	Y
mopsMIP002	520	5442.2829	0.44	6473.7947	2110443	15.56%	N
mopsMIP003	33	2693.9557	1.03	2726.8040	78907	0.49	Y
mopsMIP004	131	1665.0000	1.42	1854.0000	1688617	9.76%	N
mopsMIP005	1601	1864.5158	6.14	2536.8133	250205	41.52	Y
mopsMIP006	1437	80.7596	10.83	208.6757	1766	0.61	Y
mopsMIP007	123	257.8680	0.05	694.6643	7931509	34.38%	N
mopsMIP008	425	1575.8944	1.86	4662.9110	1281830	66.04%	N
mopsMIP009	891	10981.5584	11.12	31234.9882	901527	64.84%	N

Table D.33.: Results for a 1-hour test with the SOTA configuration (MOPSLIB test set).

name	cuts	xLP	time in SNP (s)	best IP	nodes	time (m) / gap	solved?
mopsMIB001	23	101866.0693	2.83	109940.0000	298	6.86%	N
mopsMIB002	988	1446884.2603	203.47	-	7370	-	N
mopsMIB003	275	1841393.7353	6.97	2511837.5000	255280	21.76%	N
mopsMIB004	491	3617189.3797	48.33	4955218.0000	41262	26.87%	N
mopsMIB005	187	-20437405.8900	245.73	36464706.4948	15	155.09%	N
mopsMIB006	609	4677707915.5063	42.42	4720566885.8610	532	0.49%	N
mopsMIB007	1431	71713827.8174	22.80	n.a.	5174	n.a.	N
mopsMIB008	24037	-57949.5261	16286.94	-	1	-	N
mopsMIB009	1350	74264296.2064	14.02	-	90896	-	N
mopsMIB010	2501	56013290.6471	32.08	-	29246	-	N
mopsMIB011	1088	40984715.4149	8.62	-	97540	-	N
mopsMIB012	3389	-60714456.0496	417.95	-	8122	-	N
mopsMIB013	75	-961717.3750	1.38	-959175.5865	572480	0.23%	N
mopsMIB014	77	-962418.2383	1.00	-961513.0374	585923	0.01%	N
mopsMIB015	1348	5323.0827	0.11	6383.3125	1267579	9.21%	N
mopsMIB016	881	381.5353	124.19	666.5375	9615	42.7%	N
mopsMIP001	19	11601.7017	0.02	20675.0000	341065	0.94	Y
mopsMIP002	211	5119.5965	0.30	6464.0864	2185619	18.41%	N
mopsMIP003	18	2692.8930	0.19	2726.8040	139113	0.79	Y
mopsMIP004	83	1665.0000	1.05	1842.0000	1236490	9.01%	N
mopsMIP005	1598	1864.5120	5.51	2536.8133	213216	35.75	Y
mopsMIP006	1433	80.7515	10.20	208.6757	1385	1.62	Y
mopsMIP007	104	245.0265	0.02	693.8405	7897978	9.99%	N
mopsMIP008	371	1467.4566	0.77	4693.4696	1346379	68.73%	N
mopsMIP009	905	7734.0693	9.00	32352.7360	578447	76.09%	N

Table D.34.: Results for a 1-hour test with the OLD configuration (MOPSLIB test set). For the instance `mopsMIB007`, MOPS returned an invalid result.

Bibliography

[1] COIN-OR – COmputational INfrastructure for Operations Research. See http://www.coin-or.org.

[2] GLPK – GNU Linear Programming Kit. Free Software Foundation. See http://www.gnu.org/software/glpk/.

[3] MINTO – Mixed INTeger Optimizer. See http://coral.ie.lehigh.edu/minto/.

[4] SCIP – Solving Constraint Integer Programs. See http://scip.zib.de/.

[5] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33:42 – 54, 2005.

[6] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006. See http://miplib.zib.de.

[7] AMPL Optimization LLC. AMPL – A modeling language for mathematical programming. See http://www.ampl.com.

[8] Giuseppe Andreello, Alberto Caprara, and Matteo Fischetti. Embedding $\{0, \frac{1}{2}\}$ - cuts in a branch-and-cut framework: A computational study. *INFORMS Journal on Computing*, 19(2):229–238, 2007.

[9] David L. Applegate, William Cook, Sanjeeb Dash, and Daniel G. Espinoza. Exact solutions to linear programming problems. *Operations Research Letters*, 35:693 – 699, 2007.

[10] Robert Ashford. Mixed integer programming: A historical perspective with Xpress-MP. *Annals of Operations Research*, 149(1):5–17, 2007.

[11] Alper Atamtürk. Flow pack facets of the single node fixed-charge flow polytope. *Operations Research Letters*, 29(3):107 – 114, October 2001.

[12] Alper Atamtürk and Oktay Günlük. Mingling: Mixed-integer rounding with bounds. Technical Report BCOL.07.03, IEOR, University of California-Berkeley, September 2007.

[13] Alper Atamtürk, George L. Nemhauser, and Martin W. P. Savelsbergh. Valid inequalities for problems with additive variable upper bounds. *Mathematical Programming Series A*, 91:145 – 162, 2001.

[14] Alper Atamtürk and Martin W. P. Savelsbergh. Integer-programming software systems. *Annals of Operations Research*, 140(1):67–124, 2005.

[15] Egon Balas, Sebastian Ceria, Gérard Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[16] Egon Balas and Michael Perregaard. Lift-and-project for mixed 0-1 programming: recent progress. *Discrete Applied Mathematics*, 123(1):129–154, 2002.

[17] Egon Balas and Michael Perregaard. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. *Mathematical Programming Series B*, 94(2–3):221–245, 2003.

[18] Egon Balas and Anureet Saxena. Optimizing over the split closure. *Mathematical Programming Series A*, 113(2):219–240, 2008.

[19] I. Barany, Tony J. van Roy, and Laurence A. Wolsey. Uncapacitated lot-sizing: the convex hull of solutions. *Mathematical Programming Study*, 22:32–43, 1984.

[20] Gaetan Belveaux and Laurence A. Wolsey. Lotsizelib: A library of models and matrices for lot-sizing problems. Technical report, Center for Operations Research and Econometrics, Universite Catholique de Louvain, 1999. See `http://www.core.ucl.ac.be/wolsey/lotsizel.htm`.

[21] Timo Berthold. Primal heuristics for mixed integer programs. Master's thesis, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Technischen Universität Berlin, 2006.

[22] Dimitris Bertsimas, Christopher Darnell, and Robert Soucy. Portfolio construction through mixed-integer programming at grantham, mayo, van otterloo and company. *Interfaces*, 29:49–66, 1999.

[23] Robert E. Bixby, Sebastián Ceria, Cassandra M. McZeal, and Martin W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[24] Robert E. Bixby, Mary Fenelon, Zonghao Gu, Edward Rothberg, and Roland Wunderling. MIP: Theory and practice - closing the gap. In M. J. D. Powell and

S. Scholtes, editors, *System Modelling and Optimization: Methods, Theory and Applications*, pages 19–49. Kluwer, 2000.

[25] Robert E. 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.

[26] Alberto Caprara and Matteo Fischetti. $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts. *Mathematical Programming*, 74(3):221–235, 1996.

[27] Alberto Caprara and Adam N. Letchford. On the separation of split cuts and related inequalities. *Mathematical Programming Series B*, 94(2-3):279–294, 2003.

[28] Sebastian Ceria. A brief history of lift-and-project. *Annals of Operations Research*, 149(1):57–61, 2007.

[29] Vašek Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983.

[30] Cécile Cordier, Hugues Marchand, Richard Laundy, and Laurence A. Wolsey. bc – opt : a branch-and-cut code for mixed integer programs. *Mathematical Programming*, 86(2):335 – 353, November 1999.

[31] Gérard Cornuéjols. Revival of the Gomory cuts in the 1990's. *Annals of Operations Research*, 149(1):63–66, 2007.

[32] Gérard Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming Series B*, 112(1):3–44, 2008.

[33] Gérard Cornuéjols and Milind Dawande. A class of hard small 0-1 programs. In *Integer Programming and Combinatorial Optimization*, volume 1412/1998 of *Lecture Notes in Computer Science*, pages 284 – 293. Springer Berlin / Heidelberg, 1998.

[34] Harlan Crowder, Ron S. Dembo, and John M. Mulvey. Reporting computational experiments in mathematical programming. *Mathematical Programming*, 15(1):316 – 329, 1978.

[35] Sanjeeb Dash, Marcos Goycoolea, and Oktay Günlük. Two step MIR inequalities for mixed-integer programs. Technical report, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, 2006.

[36] Sanjeeb Dash and Oktay Günlük. Valid inequalities based on simple mixed-integer sets. *Mathematical Programming Series A*, 105(1):29 – 53, January 2006.

[37] Sanjeeb Dash and Oktay Günlük. On mixing inequalities: rank, closure and cutting plane proofs. Technical report, IBM Research, 2008.

[38] Sanjeeb Dash, Oktay Günlük, and Andrea Lodi. MIR closures of polyhedral sets. *to appear in Mathematical Programming Series A*, 2008.

[39] Dash Optimization Inc. Xpress-MP. See `http://www.dashoptimization.com`.

[40] Brian T. Denton, John Forrest, and R. John Milne. IBM solves mixed-integer program to optimize its semiconductor supply chain. *Interfaces*, 36(5):386 – 399, September – October 2006.

[41] Santanu S. Dey. A note on split rank of intersection cuts. Technical report, CORE, UCL, Belgium, September 2008.

[42] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming Series A*, 91::201–213, 2002.

[43] Matteo Fischetti, Andrea Lodi, and Domenico Salvagnin. Just MIP it! In V. Maniezzo, T. Stautzle, and S. Voss, editors, *Hybridizing metaheuristics and mathematical programming*, Operations Research/Computer Science Interfaces Series. Springer, 2008. (to appear).

[44] Bernhard Fleischmann, Sonja Ferber, and Peter Heinrich. Strategic planning of BMW's global production network. *Interfaces*, 36(3):194–208, May–June 2006.

[45] Swantje Friedrich. *Algorithmische Verbesserungen für die Lösung diskreter Optimierungsmodelle*. PhD thesis, Freie Universität Berlin, 2007.

[46] Ricardo Fukasawa and Marcos Goycoolea. On the exact separation of mixed integer knapsack cuts. In *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, pages 225 – 239. Springer, June 2007.

[47] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5 – 48, March 1991.

[48] João Gonçalves and Laszlo Ladanyi. An implementation of a separation procedure for mixed integer rounding inequalities. IBM Research Report RC23686, IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, N.Y., August 2005.

[49] Stephen C. Graves. Using lagrangean techniques to solve hierarchical production planning problems. *Management Science*, 28(3):260 – 275, March 1982.

[50] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS Journal on Computing*, 10(4):427–437, 1998.

[51] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85(3):439–467, 1999.

[52] Oktay Günlük and Yves Pochet. Mixing mixed-integer inequalities. *Mathematical Programming Series A*, 90(3):429–457, 2001.

[53] J. N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42(2):201 – 212, 1994.

[54] ILOG Inc., An IBM Company. ILOG CPLEX. See `http://www.cplex.com`.

[55] David S. Johnson. A theoretician's guide to the experimental analysis of algorithms, 2001.

[56] Ellis L. Johnson, George L. Nemhauser, and Martin W. P. Savelsbergh. Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

[57] Kiavash Kianfar and Yahya Fathi. Generalized mixed integer rounding inequalities: facets for infinite group polyhedra. *to appear in: Mathematical Programming*, 2008.

[58] Diego Klabjan and George L. Nemhauser. A polyhedral study of integer variable upper bounds. *Mathematics of Operations Research*, 27(4):711 – 739, 2002.

[59] Achim Koberstein. *The Dual Simplex Method, Techniques for a fast and stable implementation*. PhD thesis, Universität Paderborn, 2005.

[60] Achim Koberstein. Progress in the dual simplex method for solving large scale LP problems: techniques for a fast and stable implementation. *Computational Optimization and Applications*, 41(2):185–204, November 2008.

[61] A. H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, July 1960.

[62] Adam N. Letchford and Andrea Lodi. Strengthening Chvátal-Gomory cuts and Gomory fractional cuts. *Operations Research Letters*, 30(2):74–82, 2002.

[63] Benjamin W. Lin and Ronald L. Rardin. Controlled experimental design for statistical comparison of integer programming algorithms. *Management Science*, 25(12):1258 – 1271, December 1979.

[64] Jeff T. Linderoth. A simplicial branch-and-bound algorithm for solving quadratically constrained quadratic programs. *Mathematical Programming Series B*, 103(2):251 – 282, June 2005.

[65] Jeff T. Linderoth and Martin W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173 – 187, February 1999.

[66] Quentin Louveaux and Laurence A. Wolsey. Lifting, superadditivity, mixed integer rounding and single node flow sets revisited. *4OR: A Quarterly Journal of Operations Research*, 1(3):173–207, 2003.

[67] Thomas L. Magnanti, Prakash Mirchandani, and Rita Vachani. The convex hull of two core capacitated network design problems. *Mathematical Programming*, 60(1 – 3):233 – 250, June 1993.

[68] Hugues Marchand. *A Polyhedral Study of the Mixed Knapsack Set and its Use to Solve Mixed Integer Programs*. PhD thesis, Faculté des Sciences Appliquées, Université catholique de Louvain, 1998.

[69] Hugues Marchand and Laurence A. Wolsey. The 0–1 knapsack problem with a single continuous variable. *Mathematical Programming*, 85(1):15 – 33, May 1999.

[70] Hugues Marchand and Laurence A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001.

[71] François Margot. Testing cut generators for mixed-integer linear programming. Technical report, Tepper School of Business, Carnegie Mellon University, 2007.

[72] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. Wiley-Interscience Series In Discrete Mathematics And Optimization. John Wiley & Sons Inc., 1990.

[73] Maximal Software Inc. MPL – Mathematical Programming Language. See `http://www.maximal-usa.com/`.

[74] Catherine C. McGeoch. Experimental analysis of algorithms. In *Handbook of Global Optimization*, volume 2, pages 489 – 513. Kluwer, 2002.

[75] Hans Mittelmann. Decision tree for optimization software: Benchmarks for optimization software, 2008. See `http://plato.asu.edu/bench.html`.

[76] MOPS Optimierungssysteme GmbH & Co. KG. MOPS – Mathematical Optimization System. See `http://www.mops-optimizer.com`.

[77] MOPS Optimierungssysteme GmbH & Co. KG. *mops - Mathematical Optimization System - White Paper*, September 2008. See `http://www.mops-optimizer.com`.

[78] MOPS Optimierungssysteme GmbH & Co. KG. *The mops User Manual*, Januar 2008. MOPS Version 9.x, 27.01.2008, See `http://www.mops-optimizer.com`.

[79] George L. Nemhauser and Laurence A. Wolsey. A recursive procedure to generate all cuts for 0–1 mixed integer programs. *Mathematical Programming*, 46(1–3):379–390, 1990.

[80] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience Series In Discrete Mathematics And Optimization. John Wiley & Sons Inc., 2nd edition, 1999.

[81] Arnold Neumaier and Oleg Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming Series A*, 99:283–296, 2004.

[82] Manfred W. Padberg, Tony J. van Roy, and Laurence A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33(4):842 – 861, July 1985.

[83] Yves Pochet and Laurence A. Wolsey. Lot-sizing with constant batches: Formulation and valid inequalities. *Mathematics of Operations Research*, 18(4):767 – 785, 1993.

[84] Yves Pochet and Laurence A. Wolsey. Polyhedra for lot-sizing with wagner – whitin costs. *Mathematical Programming*, 67(1 – 3):297 – 323, October 1994.

[85] Yves Pochet and Laurence A. Wolsey. *Production planning by mixed integer programming*. Springer, 2006.

[86] Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[87] Jan I. A. Stallaert. The complementary class of generalized flow cover inequalities. *Discrete Applied Mathematics*, 77:73 – 80, 1997.

[88] Uwe H. Suhl. MOPS - Mathematical OPtimization System. *European Journal of Operational Research*, 72:312–322, 1994.

[89] Uwe H. Suhl and Ralf Szymanski. Supernode processing of mixed-integer models. *Computational Optimization and Applications*, 3(4):317–331, 1994.

[90] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2nd edition, 2001.

[91] Tony J. van Roy and Laurence A. Wolsey. Valid inequalities and separation for uncapacitated fixed charge networks. *Operations Research Letters*, 4:105 – 213, 1985.

[92] Tony J. van Roy and Laurence A. Wolsey. Valid inequalities for mixed 0-1 programs. *Discrete Applied Mathematics*, 14:199–213, 1986.

[93] Tony J. van Roy and Laurence A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35(1):45–57, 1987.

[94] Veronika Waue. *Entwicklung von Software zur Lösung von gemischt-ganzzahligen Optimierungsmodellen mit einem Branch-and-Cut-Ansatz*. PhD thesis, Freie Universität Berlin, 2007.

[95] Franz Wesselmann. Implementation of a separation heuristic for mixed integer rounding inequalities. Master's thesis, DS&OR Lab, University of Paderborn, February 2006.

[96] Franz Wesselmann and Uwe H. Suhl. Implementation techniques for cutting plane management and selection. *submitted to Optimization Methods and Software*, 2008.

[97] Laurence A. Wolsey. Valid inequalities for 0-1 knapsacks and MIPs with generalized upper bound constraints. *Discrete Applied Mathematics*, 29(2 – 3):251 – 162, June 1990.

[98] Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience Series In Discrete Mathematics And Optimization. John Wiley & Sons Inc., 1998.

[99] Kati Wolter. Implementation of cutting plane separators for mixed integer programs. Master's thesis, Technische Universität Berlin, 2006.