Computational Optimization and Applications manuscript No.
(will be inserted by the editor)

Strong-Branching Inequalities for Convex Mixed Integer Nonlinear Programs

Mustafa Kılınç · Jeff Linderoth · James Luedtke · Andrew Miller

April 17, 2014

Abstract Strong branching is an effective branching technique that can significantly reduce the size of the branch-and-bound tree for solving Mixed Integer Nonlinear Programming (MINLP) problems. The focus of this paper is to demonstrate how to effectively use "discarded" information from strong branching to strengthen relaxations of MINLP problems. Valid inequalities such as branching-based linearizations, various forms of disjunctive inequalities, and mixing-type inequalities are all discussed. The inequalities span a spectrum from those that require almost no extra effort to compute to those that require the solution of an additional linear program. In the end, we perform an extensive computational study to measure the impact of each of our proposed techniques. Computational results reveal that existing algorithms can be significantly improved by leveraging the information generated as a byproduct of strong branching in the form of valid inequalities.

Keywords Mixed-Integer Nonlinear Programming, Strong-Branching, Disjunctive Inequalities, Mixing Inequalities

1 Introduction

In this work, we study valid inequalities derived from strong branching for solving the convex mixed integer nonlinear programming (MINLP) problem

$$\begin{aligned} z_{\text{MINLP}} = \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_j(x) \leq 0, \quad \forall j \in J, \\ & x \in X, \quad x_I \in \mathbb{Z}^{|I|}. \end{aligned} \tag{1}$$

M. Kılınç
Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, E-mail: mrk46@pitt.edu

J. Linderoth
Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI 53706, E-mail: linderoth@wisc.edu

J. Luedtke
Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI 53706, E-mail: jrluedt1@wisc.edu

A. Miller
United Parcel Service, Atlanta, GA 30328, E-mail: foresomenteneikona@gmail.com

The functions $f : X \to \mathbb{R}$ and $g_j : X \to \mathbb{R}\ \forall j \in J$ are smooth, convex functions, and the set $X \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n_+ \mid Ax \leq b\}$ is a polyhedron. The set $I \subseteq \{1, \ldots, n\}$ contains the indices of discrete variables, and we define $B \subseteq I$ as the index set of binary variables.

In order to have a linear objective function an auxiliary variable η is introduced, and the nonlinear objective function is moved to the constraints, creating the equivalent problem

$$z_{\text{MINLP}} = \text{minimize}\{\eta : (\eta, x) \in \mathcal{S},\ x_I \in \mathbb{Z}^{|I|}\} \tag{2}$$

where

$$\mathcal{S} = \{(\eta, x) \in \mathbb{R} \times X \mid\ f(x) \leq \eta,\ g_j(x) \leq 0 \quad \forall j \in J\ \}.$$

We define the set $\mathcal{P} = \{(\eta, x) \in \mathcal{S} \mid x_I \in \mathbb{Z}^{|I|}\}$ to be the set of feasible solutions to (2).

Branch and bound forms a significant component of most algorithms for solving MINLP problems. In NLP-based branch and bound, the lower bound on the value z_{MINLP} comes from the solution value of the nonlinear program (NLP) that is the *continuous relaxation* of (2):

$$z_{\text{NLPR}} = \min\{\eta : (\eta, x) \in \mathcal{S}\}. \tag{3}$$

In linearization-based approaches, such as outer-approximation [18] or the LP/NLP branch-and-bound algorithm [31], the lower bound comes from solving a linear program (LP), often called *the master problem* that is based on a polyhedral outer-approximation of $\mathcal{P}$:

$$\begin{aligned} z_{\text{MP}(\mathcal{K})} = \min \quad & \eta \\ \text{s.t. } & \eta \geq f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x}) \quad \forall \bar{x} \in \mathcal{K}, \\ & g_j(\bar{x}) + \nabla g_j(\bar{x})^T(x - \bar{x}) \leq 0 \quad \forall j \in J, \forall \bar{x} \in \mathcal{K}, \\ & x \in X, \end{aligned} \tag{4}$$

where $\mathcal{K}$ is a set of points about which linearizations of the convex functions $f(\cdot)$ and $g_j(\cdot)$ are taken. For more details on algorithms for solving convex MINLP problems, the reader is referred to the surveys [11, 12, 22].

Regardless of the bound employed by the branch-and-bound algorithm, algorithms are required to branch. By far the most common branching approach is branching on individual integer variables. In this approach, branching involves selecting a single branching variable $x_i, i \in I$ such that in the solution $\hat{x}$ to the relaxation (3) or (4), $\hat{x}_i \notin \mathbb{Z}$. Based on the branching variable, the problem is recursively divided, imposing the constraint $x_i \leq \lfloor \hat{x}_i \rfloor$ for one child subproblem and $x_i \geq \lceil \hat{x}_i \rceil$ for the other. The relaxation solution $\hat{x}$ may have many candidates for the branching variable, and the choice of branching variable can have a very significant impact on the size of the search tree [28, 3]. Ideally, the selection of the branching variable would lead to the smallest resulting enumeration tree. However, without explicitly enumerating the trees coming from all possible branching choices, choosing the best variable is difficult to do exactly. A common heuristic is to select the branching variable that is likely to lead to the largest improvement in the children nodes' lower bounds. The reasoning behind this heuristic is that nodes of the branch-and-bound tree are fathomed when the lower bound for the node is larger than the current upper bound, so one should select branching variables to increase the children nodes' lower bounds by as much as possible.

In the context of solving the Traveling Salesman Problem, Applegate *et al.* [4] propose to explicitly calculate the lower bound changes for many candidate branching variables and choose the branching variable that results in the largest change for the resulting child nodes. This method has come to be known as *strong branching*. Strong branching or variants of strong branching, such as reliability branching [3], have been implemented in state-of-the-art solvers for solving mixed integer linear programs, the special case of MINLP where all functions are linear.

For MINLP, one could equally well impose the extra bound constraint on the candidate branching variable in the nonlinear continuous relaxation (3). We call this type of strong branching, *NLP-based strong branching*. In particular, for a fractional solution $\hat{x}$, NLP-based strong branching is performed by solving the two continuous nonlinear programming problems

$$\hat{\eta}_i^0 = \text{minimize}\{\eta : (\eta, x) \in \mathcal{S}_i^0\} \tag{NLP_i^0}$$

and

$$\hat{\eta}_i^1 = \text{minimize}\{\eta : (\eta, x) \in \mathcal{S}_i^1\} \tag{NLP_i^1}$$

for each fractional variable index $i \in F \stackrel{\text{def}}{=} \{i \in I \mid \hat{x}_i \notin \mathbb{Z}\}$, where $\mathcal{S}_i^0 = \{(\eta, x) \in \mathcal{S} \mid x_i \leq \lfloor \hat{x}_i \rfloor\}$ and $\mathcal{S}_i^1 = \{(\eta, x) \in \mathcal{S} \mid x_i \geq \lceil \hat{x}_i \rceil\}$. The optimal values of the subproblems $(\eta_i^0, \eta_i^1 \ \forall i \in F)$ are used to choose a branching variable [2,28].

In the LP/NLP branch-and-bound algorithm, the NLP continuous relaxation (3) is not solved at every node in the branch-and-bound tree, although it is typically solved at the root node. Instead, the polyhedral outer-approximation (4) is used throughout the branch-and-bound tree. The outer-approximation is refined when an integer feasible solution to the current linear relaxation is obtained. Since the branch-and-bound tree is based on a *linear* master problem, it is not obvious whether strong branching should be based on solving the *nonlinear* subproblems (NLP_i^0) and (NLP_i^1) or based on solving the LP analogues to these where the nonlinear constraints are replaced by the current linear outer approximation. However, our computational experience in Section 4.3 is that even when using a linearization-based method, a strong-branching approach based on solving NLP subproblems can yield significant reduction in the number of nodes in a branch-and-bound tree. Bonami *et al.* [13] have also given empirical evidence of the effectiveness of NLP-based strong branching for solving convex MINLP problems.

On the other hand, using NLP subproblems for strong branching is computationally more intensive than using LP subproblems, so it makes sense to attempt to use information obtained from NLP-based strong branching in ways besides simply choosing a branching variable. In this work, we describe a variety of ways to transfer strong-branching information into the child node relaxations. The focus of our work will be on improving the implementation of the LP/NLP branch-and-bound algorithm in the software package FilMINT [1]. The information may be transferred to the child relaxations by adding additional linearizations to the master problem (4) or through the addition of simple disjunctive inequalities. The idea of applying disjunctive programming ideas to solve MINLP problems has been previously employed by many authors [36,35]. We demonstrate the relation of the simple disjunctive inequalities we derive to standard disjunctive inequalities. We derive and discuss many different techniques by which these simple disjunctive strong-branching inequalities may be strengthened. The strengthening methods range from methods that require almost no extra computation to methods that require the solution of a linear program. In the end, we perform an extensive computational study to measure the impact of each of our methods. Incorporating these changes in the solver FilMINT results in a significant reduction in CPU time on the instances in our test suite.

The remainder of the paper is divided into 4 sections. Section 2 describes some simple methods for using inequalities generated as an immediate byproduct of the strong-branching process. Section 3 concerns methods for strengthening inequalities obtained from strong branching. Section 4 reports on our computational experience with all of our described methods, and Section 5 offers some conclusions of our work.

2 Simple Strong-Branching Inequalities

In this section, we describe elementary ways that information obtained from the strong-branching procedure can be recorded and used in the form of valid inequalities for solving MINLP problems. The simplest scheme

is to use linearizations from the NLP subproblems. Alternatively, valid inequalities may be produced from the disjunction, and these inequalities may be combined by mixing.

2.1 Linearizations

When using a linearization-based approach for solving MINLP problems, a simple idea for obtaining more information from the NLP strong-branching subproblems (NLP_i^0) and (NLP_i^1) is to add the solutions to these subproblems to the linearization point set $\mathcal{K}$ of the master problem (4).

There are a number of reasons why using linearizations about solutions to (NLP_i^0) and (NLP_i^1) may yield significant computational benefit. First, the inequalities are trivial to obtain once the NLP subproblems have been solved; one simply has to evaluate the gradient of the nonlinear functions at the optimal solutions of (NLP_i^0) and (NLP_i^1). Second, the inequalities are likely to improve the lower bound in the master problem (4) after branching. In fact, if these linearizations are added to (4), then after branching on the variable x_i, the lower bound $z_{\text{MP}(\mathcal{K})}$ will be at least as large as the bound obtained by an NLP-based branch-and-bound algorithm. Third, optimal solutions to (NLP_i^0) and (NLP_i^1) satisfy the nonlinear constraints of the MINLP problem. Computational experience with different linearization approaches for solving MINLP problems in [1] suggests that the most important linearizations to add to the master problem (4) are those obtained at points that are feasible to the NLP relaxation. Finally, depending on the branching strategy employed, using these linearizations may lead to improved branching decisions. For example, our branching strategy, described in detail in Section 4.1, is based on pseudocosts that are initialized using NLP strong-branching information, but are updated based on the current polyhedral outer approximation after a variable is branched on. Thus, the improved polyhedral outer approximation derived from these linearizations may lead to improved pseudocosts, and hence better branching decisions.

2.2 Simple disjunctive inequalities

Another approach to collecting information from the strong-branching subproblems (NLP_i^0) and (NLP_i^1) is to combine information from the two subproblems using *disjunctive* arguments. We call the first very simple inequality a *strong-branching cut* (SBC). We omit the simple proof of its validity.

Proposition 1 *The strong-branching cut (SBC)*

$$\eta \geq \hat{\eta}_i^0 + (\hat{\eta}_i^1 - \hat{\eta}_i^0)x_i \tag{5}$$

is valid for the MINLP (2)*, where* $i \in B$*, and* $\hat{\eta}_i^0$*,* $\hat{\eta}_i^1$ *are the optimal solution values to* (NLP_i^0) *and* (NLP_i^1)*, respectively.*

Similar inequalities can be written for other common branching disjunctions, such as the GUB constraint

$$\sum_{i \in S} x_i = 1, \tag{6}$$

where $S \subseteq B$ is subset of binary variables.

Proposition 2 *Let* (6) *be a constraint for the MINLP problem* (2)*. The GUBSBC inequality*

$$\eta \geq \sum_{i \in S} \hat{\eta}_i^1 x_i \tag{7}$$

is valid for (2)*, where* $\hat{\eta}_i^1$ *is the optimal solution value to* (NLP_i^1) *for* $i \in S$*.*

If the instance contains a constraint of the form $\sum_{i \in S} x_i \leq 1$, then a slack binary variable can be added to convert it to the form of (6), so that (7) may be used in this case as well.

The simple SBC (5) can be generalized to disjunctions based on general integer variables. The following result follows by using a convexity argument and a disjunctive argument based on the disjunction $x_i \leq \lfloor \hat{x}_i \rfloor$ or $x_i \geq \lceil \hat{x}_i \rceil$, for some integer variable x_i whose relaxation value $\hat{x}_i$ is fractional. A complete proof of Proposition 3 can be found in the Ph.D. thesis of Kılınç [26].

Proposition 3 *For $i \in I$, the strong-branching cut*

$$\eta \geq \hat{\eta}^0 + (\hat{\eta}^1 - \hat{\eta}^0)(x_i - \lfloor \hat{x}_i \rfloor)$$

is valid for (2), where $\hat{\eta}^0$ and $\hat{\eta}^1$ are the optimal solution values to (NLP_i^0) and (NLP_i^1), respectively.

2.3 Mixing Strong-Branching Cuts

Mixing sets arose in the study of a lot-sizing problem by Pochet and Wolsey [30] and were systematically studied by Günlük and Pochet [24]. A similar set was introduced as a byproduct of studying the mixed vertex packing problem by Atamtürk, Nemhauser, and Savelsbergh [5].

A collection of strong-branching inequalities (5) can be transformed into a mixing set in a straightforward manner. Specifically, let $\hat{B} \subseteq B$ be the index set of binary variables on which strong branching has been performed, and let $\delta_i = \hat{\eta}_i^1 - \hat{\eta}_i^0$ be the difference in objective values between the two NLP subproblems (NLP_i^0) and (NLP_i^1). Proposition 1 states that the SBC inequalities

$$\eta \geq \hat{\eta}_i^0 + \delta_i x_i \quad \forall i \in \hat{B} \tag{8}$$

are valid for the MINLP problem (2). Without loss of generality, assume that $\delta_i \geq 0$, for otherwise, one can define $\tilde{x}_i = 1 - x_i$, $\tilde{\delta}_i = -\delta_i$, and write (8) as

$$\eta \geq \hat{\eta}_i^1 + \tilde{\delta}_i \tilde{x}_i,$$

which has $\tilde{\delta}_i \geq 0$. Since $\delta_i \geq 0$, the value

$$\underline{\eta} \stackrel{\text{def}}{=} \max_{\forall i \in \hat{B}} \hat{\eta}_i^0 \leq \eta$$

is a valid lower bound for the objective function variable η. Furthermore, by definition, the inequalities

$$\eta \geq \underline{\eta} + \sigma_i x_i \quad \forall i \in \bar{B} \tag{9}$$

are valid for (MINLP), where $\sigma_i = \hat{\eta}_i^1 - \underline{\eta}$ and $\bar{B} = \{i \mid \sigma_i > 0, i \in \hat{B}\}$. The inequalities (9) define a mixing set

$$\mathcal{M} = \{(\eta, x) \in \mathbb{R} \times \{0,1\}^{|\bar{B}|} \mid \eta \geq \underline{\eta} + \sigma_i x_i \quad \forall i \in \bar{B}\}. \tag{10}$$

Proposition 4 is a straightforward application of the *mixing inequalities* of [24] or the *star inequalities* of [5] and demonstrates that the inequalities, which we call MIXSBC, are valid for $\mathcal{M}$, thus valid for the feasible region $\mathcal{P}$ of the MINLP problem.

Proposition 4 ([5, 24]) *Let $T = \{i_1, \ldots, i_t\}$ be a subset of $\bar{B}$ such that $\sigma_{i_{(j-1)}} < \sigma_{i_j}$ for $j = 2, \ldots, t$. Then the MIXSBC inequality*

$$\eta \geq \underline{\eta} + \sum_{i_j \in T} \theta_{i_j} x_{i_j} \tag{11}$$

is valid for $\mathcal{M}$, where $\theta_{i_1} = \sigma_{i_1}$ and $\theta_{i_j} = \sigma_{i_j} - \sigma_{i_{j-1}}$ for $j = 2, \ldots, t$.

If a MIXSBC inequality (11) is violated by a fractional solution $\hat{x}$, it may be identified in polynomial time using a separation algorithm given in [5] or [24].

3 Strengthened Strong-Branching Inequalities

The valid inequalities introduced in Section 2 can be obtained almost "for free" using strong-branching information. In this section, we explore methods for strengthening and combining simple disjunctive inequalities. By doing marginally more work, we hope to obtain more effective valid inequalities. The section begins by examining the relationship between the simple strong-branching cut (5) and general disjunctive inequalities. A byproduct of the analysis is a simple mechanism for strengthening the inequalities (5) by using the optimal Lagrange multipliers from the NLP strong-branching subproblems. The analysis also suggests the construction of a cut-generating linear program (CGLP) to further improve the (weak) disjunctive inequality generated by strong branching.

3.1 SBC and Disjunctive Inequalities

The SBC (5) is a disjunctive inequality. For ease of presentation, we describe the relationship only for disjunctions of binary variables. The extension to disjunctions on integer variables is straightforward and can be found in [26]. Let $(\hat{\eta}^0, \hat{x}^0)$ and $(\hat{\eta}^1, \hat{x}^1)$ be optimal solutions to the NLP subproblems (NLP_i^0) and (NLP_i^1), respectively. Since $f(\cdot)$ and $g_j(\cdot)$ are convex, linearizing the nonlinear inequalities about the points $(\hat{\eta}^0, \hat{x}^0)$ and $(\hat{\eta}^1, \hat{x}^1)$ gives two polyhedra

$$\mathcal{X}_i^0 = \left\{ (\eta, x) \;\middle|\; \begin{array}{rcl} c^0 x - \eta & \leq & b^0 \\ D^0 x & \leq & d^0 \\ Ax & \leq & b \\ x_i & \leq & 0 \\ x \in \mathbb{R}^n_+ & & \end{array} \right\}, \quad \mathcal{X}_i^1 = \left\{ (\eta, x) \;\middle|\; \begin{array}{rcl} c^1 x - \eta & \leq & b^1 \\ D^1 x & \leq & d^1 \\ Ax & \leq & b \\ -x_i & \leq & -1 \\ x \in \mathbb{R}^n_+ & & \end{array} \right\} \tag{12}$$

that outer-approximate the feasible region of the two strong-branching subproblems. In the description of the polyhedra (12), we use the following notation for the gradient $\nabla f(x) \in \mathbb{R}^{n \times 1}$, and Jacobian $\nabla g(x) \in \mathbb{R}^{n \times |J|}$ of the objective and constraint functions at various points:

$$\begin{aligned} c^0 &= \nabla f(\hat{x}^0)^T, & c^1 &= \nabla f(\hat{x}^1)^T, \\ b^0 &= \nabla f(\hat{x}^0)^T \hat{x}^0 - \hat{\eta}^0, & b^1 &= \nabla f(\hat{x}^1)^T \hat{x}^1 - \hat{\eta}^1, \\ D^0 &= \nabla g(\hat{x}^0)^T, & D^1 &= \nabla g(\hat{x}^1)^T, \\ d^0 &= \nabla g(\hat{x}^0)^T \hat{x}^0 - g(\hat{x}^0) & d^1 &= \nabla g(\hat{x}^1)^T \hat{x}^1 - g(\hat{x}^1). \end{aligned}$$

We may assume that the sets $\mathcal{X}_i^0$ and $\mathcal{X}_i^1$ are non-empty, for if one of the sets is empty, the bound on the variable x_i may be fixed to its alternative value. Since $\mathcal{X}_i^0$ and $\mathcal{X}_i^1$ are polyhedra, we may apply known disjunctive theory to obtain the following theorem.

Theorem 1 *[7] The disjunctive inequality*

$$\alpha x - \sigma \eta \leq \beta \tag{13}$$

is valid for $\text{conv}(\mathcal{X}_i^0 \cup \mathcal{X}_i^1)$ *and hence for the MINLP problem* (2) *if there exists* $\lambda^0, \lambda^1 \in \mathbb{R}^{1 \times |J|}_+, \mu^0, \mu^1 \in \mathbb{R}^{1 \times m}_+, \theta^0, \theta^1 \in \mathbb{R}_+$ *and* $\sigma \in \mathbb{R}_+$ *such that*

$$\alpha \leq \sigma c^0 + \lambda^0 D^0 + \mu^0 A + \theta^0 e_i, \tag{14a}$$
$$\alpha \leq \sigma c^1 + \lambda^1 D^1 + \mu^1 A - \theta^1 e_i, \tag{14b}$$
$$\beta \geq \sigma b^0 + \lambda^0 d^0 + \mu^0 b, \tag{14c}$$
$$\beta \geq \sigma b^1 + \lambda^1 d^1 + \mu^1 b - \theta^1, \tag{14d}$$
$$\lambda^0, \lambda^1, \mu^0, \mu^1, \theta^0, \theta^1, \sigma \geq 0. \tag{14e}$$

One specific choice of multipliers $\lambda^0, \lambda^1, \mu^0, \mu^1, \theta^0, \theta^1, \sigma$ in (14) leads to the strong-branching inequality (5).

Proposition 5 *Let $(\hat{\eta}^0, \hat{x}^0)$ and $(\hat{\eta}^1, \hat{x}^1)$ be optimal solutions to the NLP subproblems (NLP_i^0) and (NLP_i^1), respectively, satisfying a constraint qualification. Then,*

$$\eta \geq \hat{\eta}^0 + (\hat{\eta}^1 - \hat{\eta}^0)x_i$$

is a disjunctive inequality (13)*.*

Proof.

Since both $(\hat{\eta}^0, \hat{x}^0)$ and $(\hat{\eta}^0, \hat{x}^0)$ satisfy a constraint qualification, there exists Lagrange multiplier vectors $\hat{\lambda}^h, \hat{\mu}^h, \hat{\phi}^h \geq 0$ and a Lagrange multiplier $\hat{\theta}^h \geq 0$, for each $h \in \{0, 1\}$ satisfying the Karush-Kuhn-Tucker (KKT) conditions

$$\nabla f(\hat{x}^h)^T + \hat{\lambda}^h \nabla g(\hat{x}^h)^T + \hat{\mu}^h A - \hat{\phi}^h + \hat{\theta}^h e_i = 0, \tag{15a}$$

$$\hat{\lambda}^h g(\hat{x}^h) = 0, \tag{15b}$$

$$\hat{\mu}^h (A\hat{x}^h - b) = 0, \tag{15c}$$

$$\hat{\phi}^h (\hat{x}^h - h) = 0, \tag{15d}$$

$$\hat{x}_i^h \hat{\theta}^h = 0 \tag{15e}$$

We assign multipliers $\sigma^0 = 1$, $\lambda^0 = \hat{\lambda}^0$, $\mu^0 = \hat{\mu}^0$, $\theta^0 = \hat{\theta}^0 - \hat{\eta}^0 + \hat{\eta}^1$ into (14a) and (14c) and $\sigma^1 = 1$, $\lambda^1 = \hat{\lambda}^1$, $\mu^1 = \hat{\mu}^1$, $\theta^1 = \hat{\theta}^1 + \hat{\eta}^0 - \hat{\eta}^1$ into (14b) and (14d) in Theorem 1. Substituting these multipliers into (14) and simplifying the resulting inequalities using the KKT conditions (15) demonstrates that the SBC (5) is a disjunctive inequality. The algebraic details of the proof can be found in [26].

□

3.2 Multiplier Strengthening

The analogy between the strong-branching inequality (5) and disjunctive inequality (13) leads immediately to simple ideas for strengthening the strong-branching inequality using Lagrange multiplier information. Specifically, a different choice of multipliers for the disjunctive cut (13) leads immediately to a stronger inequality.

Theorem 2 *Let $(\hat{\eta}^0, \hat{x}^0)$ be the optimal (primal) solution to (NLP_i^0) with associated Lagrange multipliers $(\hat{\lambda}^0, \hat{\mu}^0, \hat{\phi}^0, \hat{\theta}^0)$. Likewise, let $(\hat{\eta}^1, \hat{x}^1)$ be the optimal (primal) solution to (NLP_i^1) with associated Lagrange multipliers $(\hat{\lambda}^1, \hat{\mu}^1, \hat{\phi}^1, \hat{\theta}^1)$. Define $\hat{\mu}^* = \min\{\hat{\mu}^0, \hat{\mu}^1\}$, and $\phi^* = \min\{\phi^0, \phi^1\}$. If (NLP_i^0) and (NLP_i^1) both satisfy a constraint qualification, then the strengthened strong-branching cut (SSBC)*

$$\hat{\eta}^0 + \hat{\mu}^*(b - Ax) + \hat{\phi}^* x + (\hat{\eta}^1 - \hat{\eta}^0)x_i \leq \eta \tag{16}$$

is a disjunctive inequality (13).

Proof. We substitute the multipliers $\lambda^h = \hat{\lambda}^h$, $\mu^h = \hat{\mu}^h - \hat{\mu}^*$, $h \in \{0, 1\}$, $\sigma = 1$, $\theta^0 = \hat{\theta}^0 - \hat{\eta}^0 + \hat{\eta}^1$, $\theta^1 = \hat{\theta}^1 + \hat{\eta}^0 - \hat{\eta}^1$ into (14) in Theorem 1. Simplifying the resulting expressions using the KKT conditions (15) demonstrates the result. Details of the algebraic steps required are given in the Ph.D. thesis of Kılınç [26].

□

3.3 Strong-Branching CGLP

In Theorem 1, we gave necessary conditions for the validity of a disjunctive inequality for the set $\text{conv}(\mathcal{X}_i^0 \cup \mathcal{X}_i^1)$. A most violated disjunctive inequality can be found by solving *Cut Generating Linear Program* (CGLP) that maximizes the violation of the resulting cut with respect to a given point $(\hat{\eta}, \hat{x})$:

$$\begin{aligned} \text{maximize} \quad & \beta - \alpha\hat{x} + \sigma\hat{\eta} \\ \text{subject to} \quad & \alpha \leq \sigma c^0 + \lambda^0 D^0 + \mu^0 A + \theta^0 e_i, \\ & \alpha \leq \sigma c^1 + \lambda^1 D^1 + \mu^1 A - \theta^1 e_i, \\ & \beta \geq \sigma b^0 + \lambda^0 d^0 + \mu^0 b, \\ & \beta \geq \sigma b^1 + \lambda^1 d^1 + \mu^1 b - \theta^1, \\ & \lambda^0, \mu^0, \theta^0, \lambda^1, \mu^1, \theta^1, \sigma \geq 0. \end{aligned} \tag{17}$$

A feasible solution to (17) with a positive objective function corresponds to a disjunctive inequality violated at $(\hat{\eta}, \hat{x})$. However, the set of feasible solutions to CGLP is a cone and needs to be truncated to produce a bounded optimal solution value in case a violated cut exists. The choice of the normalization constraint used to truncate the cone can be a crucial factor in the effectiveness of disjunctive cutting planes. One normalization constraint studied in [8,9] is the α-normalization:

$$\sum_{i=1}^{n} |\alpha_i| + \sigma = 1. \tag{αNORM}$$

The most widely used normalization constraint was proposed by [6] and is called the *Standard Normalization Condition* (SNC)[20]:

$$\sum_{h \in \{0,1\}} \Big(\sum_{j=1}^{|J|} \lambda_j^h + \sum_{j=1}^{m} \mu_j^h + \theta^h + \sigma \Big) = 1. \tag{SNC}$$

The SNC normalization is criticized by Fischetti, Lodi, and Tramontani [20] for its dependence on the relative scaling of the constraints. To overcome this drawback, they proposed the *Euclidean Normalization*

$$\sum_{h \in \{0,1\}} \Big(\sum_{j=1}^{|J|} \|D_{j\bullet}^h\| \lambda_j^h + \sum_{j=1}^{m} \|A_{j\bullet}\| \mu_j^h + \theta^h + \|c^h\| \sigma \Big) = 1, \tag{EN}$$

instead of (SNC). In (EN), $D_{j\bullet}^0$ and $D_{j\bullet}^1$ are the j^{th} row of the matrices D^0 and D^1 respectively, and $A_{j\bullet}$ is the j^{th} row of A. We refer reader to [20] for further discussion on normalization constraints and their impact on the effectiveness of disjunctive inequalities. In Section 4.8 we will report on the effect of different normalization constraints on our disjunctive inequalities.

3.4 Monoidal Strengthening

Disjunctive cuts can be further strengthened by exploiting integrality requirements of variables. This method was introduced by Balas and Jeroslow, where they call it *monoidal strengthening* [10]. In any disjunctive cut (13) the coefficient of x_k, $k \in I \setminus \{i\}$ can be strengthened to take the value

$$\tilde{\alpha}_k = \max\{\alpha_k^0 - \theta^0 \lceil \hat{m}_k \rceil, \alpha_k^1 + \theta^1 \lfloor \hat{m}_k \rfloor\}$$

where

$$\alpha_k^0 = \sigma c_k^0 + \lambda^0 D_{\bullet k}^0 + \mu^0 A_{\bullet k},$$
$$\alpha_k^1 = \sigma c_k^1 + \lambda^1 D_{\bullet k}^1 + \mu^1 A_{\bullet k},$$
$$\hat{m}_k = \frac{\alpha_k^0 - \alpha_k^1}{\theta^0 + \theta^1},$$

and $\lambda_k^0, \mu_k^0, \theta^0, \lambda_k^1, \mu_k^1, \theta^1, \sigma$ satisfy the requirements (14) for multipliers in a disjunctive inequality. The notation $D_{\bullet k}, A_{\bullet k}$ represents the k^{th} column of the associated matrix.

3.5 Lifting

The disjunctive inequality (13) can be lifted to become globally valid if generated at a node of the branch-and-bound tree. Assume that the inequality is generated at a node of the branch-and-bound tree where the variables in the set F_0 are fixed to zero and the variables in the set F_1 are fixed to one. Without loss of generality, we can assume that F_1 is empty by complementing all variables before formulating the CGLP (17).

Let R be the set of unfixed variables and $(\alpha, \beta, \lambda^0, \mu^0, \lambda^1, \mu^1, \sigma)$ be a solution to (17) in the subspace where the variables in the set F_0 are fixed to zero so that $\alpha \in \mathbb{R}^{|R|}$. The lifting coefficient for the fixed variables is given by Balas *et al.* [8,9] as

$$\gamma_j = \min\{\sigma c_j^0 + \lambda^0 D_{\bullet j}^0 + \mu^0 A_{\bullet j}, \sigma c_j^1 + \lambda^1 D_{\bullet j}^1 + \mu^1 A_{\bullet j}\}.$$

Thus, the inequality

$$\sum_{j \in R} \alpha_j x_j + \sum_{j \in F_0} \gamma_j x_j - \sigma\eta \leq \beta$$

is valid for the MINLP problem (2).

4 Computational Experience

In this section, we report on a collection of experiments designed to test the ideas presented in Sections 2 and 3 with the end goal of deducing how to most effectively exploit information obtained when solving NLP subproblems in a strong-branching scheme. Our implementation is done using the FilMINT solver for convex MINLP problems.

4.1 FilMINT

FilMINT is an implementation of the LP/NLP-Based Branch-and-Bound algorithm of Quesada and Grossmann [31], which uses the outer-approximation master problem (4). In our experiments, all strong-branching inequalities are added directly to (4). FilMINT uses MINTO [29] to enforce integrality of the master problem via branching and filterSQP [21] for solving nonlinear subproblems that are both necessary for convergence of the method and used in this work to obtain NLP-based strong-branching information. In our experiments, FilMINT used the CPLEX (v12.2) software to solve linear programs.

FilMINT by default employs nearly all of MINTO's enhanced MILP features, such as cutting planes, primal heuristics, row management, and enhanced branching and node selection rules. FilMINT uses the best estimate method for node selection [28].

FilMINT uses a reliability branching approach [3], where strong branching based on the current master *linear program* is performed a limited number of times for each variable. The feasible region of the linear master problem (4) may be significantly strengthened by MINTO's preprocessing and cutting plane

mechanisms, and these formulation improvements are extremely difficult to communicate to the nonlinear solver Filter-SQP. Our approach for communicating NLP-based strong-branching information to the master problem was implemented in the following manner. For each variable, we perform NLP-based strong branching by solving (NLP_i^0) and (NLP_i^1) the first time the variable is fractional in a relaxation solution. Regardless of the inequalities we add to the master problem, we solve (NLP_i^0) and (NLP_i^1) only once per variable to limit the computational burden from solving NLP subproblems, which is appropriate in the context of the linearization-based LP/NLP branch-and-bound algorithm that is used in FilMINT. We then simply add the strong-branching inequalities under consideration to the master problem and then let FilMINT make its branching decisions using its default mechanism. This affects the bounds in a manner similar to NLP-based strong branching. For example, for a fractional variable x_i, after adding a simple SBC (5) or linearizations about solutions to (NLP_i^0) and (NLP_i^1), when FilMINT performs LP-based strong branching on x_i, the bound obtained from fixing $x_i \leq \lfloor \hat{x}_i \rfloor$ will be at least $\hat{\eta}_i^0$, and likewise the bound obtained from fixing $x_i \geq \lceil \hat{x}_i \rceil$ will be at least $\hat{\eta}_i^1$. Note however, that adding inequalities will also likely affect the value of the relaxation, so the pseudocosts, which measure the *rate of change* of the objective function per unit change in variable bound, may also be affected.

4.2 Computational Setup

Our test suite consists of convex MINLPs collected from the MacMINLP collection [27], the GAMS MINLP World [15], the collection on the website of the IBM-CMU research group [34], and instances that we created ourselves. The test suite consists of 40 convex instances covering a wide range of practical applications such as multi-product batch plant design problems [32,38], layout design problems [33,16], synthesis design problems [18,37], retrofit planning [33], stochastic service system design problems [19], cutting stock problems [25], uncapacitated facility location problems [23] and network design problems [14]. Characteristics of the instances are given in Table 6, which lists whether or not the instance has a nonlinear objective function, the total number of variables, the number of integer variables, the number of constraints, how many of the constraints are nonlinear, and the number of GUB constraints. We chose the instances so that no one family of instances is overrepresented in the group and so that each of the instances is not "too easy" or "too hard." To accomplish this, we chose instances so that the default version of FilMINT is able to solve each of these instances using CPU time in the range of 30 seconds to 3 hours.

The computational experiments were run on a cluster of machines equipped with Intel Xeon microprocessors clocked at 2.00 GHz and 256 GB of RAM, using only one thread for each run. In order to concisely display the relative performance of different solution techniques, we make use of performance profiles (see [17]). A performance profile is a graph of the relative performance of different solvers on a fixed set of instances. In a performance profile graph, the x-axis is used for the performance factor. The y-axis gives the fraction of instances for which the performance of that solver is within a factor of x of the best solver for that instance. In our experiments, we use both the number of nodes in the branch and bound tree and the CPU solution time as performance metrics.

We often use the "extra" gap closed at the root node as a measure to assess the strength of a class of valid inequalities. The extra gap closed measures the relative improvement in lower bound at the root node over the lower bound found without adding the valid inequalities. Specifically, the extra percentage gap closed is

$$100 \left(\frac{z_{CUTS} - z_{\text{MP}(\mathcal{K})}}{z_{MINLP} - z_{\text{MP}(\mathcal{K})}} \right),$$

where z_{CUTS} is the value of LP relaxation after adding inequalities, $z_{\text{MP}(\mathcal{K})}$ is the value of LP relaxation of reduced master problem after preprocessing and default set of cuts of MINTO, and z_{MINLP} is the optimal solution value.

We summarize computational results in small tables that list the (arithmetic) average extra gap closed, the number of nodes, and the CPU solution time.

Strong-branching inequalities are added in rounds. After adding cuts at a node of the branch-and-bound tree, the linear program is resolved, and a new solution to the relaxation of the master problem (4) is obtained. The strong-branching subproblems (NLP_i^0) and (NLP_i^1) are solved for all fractional variables in the new solution that have not yet been initialized, and associated strong-branching inequalities are added. If inequalities are generated at a non-root node, they are lifted to make them globally valid, as explained in Section 3.5. Recall that NLP-based strong branching is performed at most once for each variable.

We are primarily interested in the impact of using strong-branching information to improve the lower bound of a linearization-based algorithm. Therefore, to eliminate variability in solution time induced by the effect of finding improved upper bounds during the search, in our experiments we input the optimal solution value to FilMINT as a cutoff value and disable primal heuristics.

4.3 Performance of SBC Inequalities and Linearizations

Our first experiment was aimed at comparing *elementary* methods for exploiting information from NLP-based strong-branching subproblems. The methods chosen for comparison in this experiment were

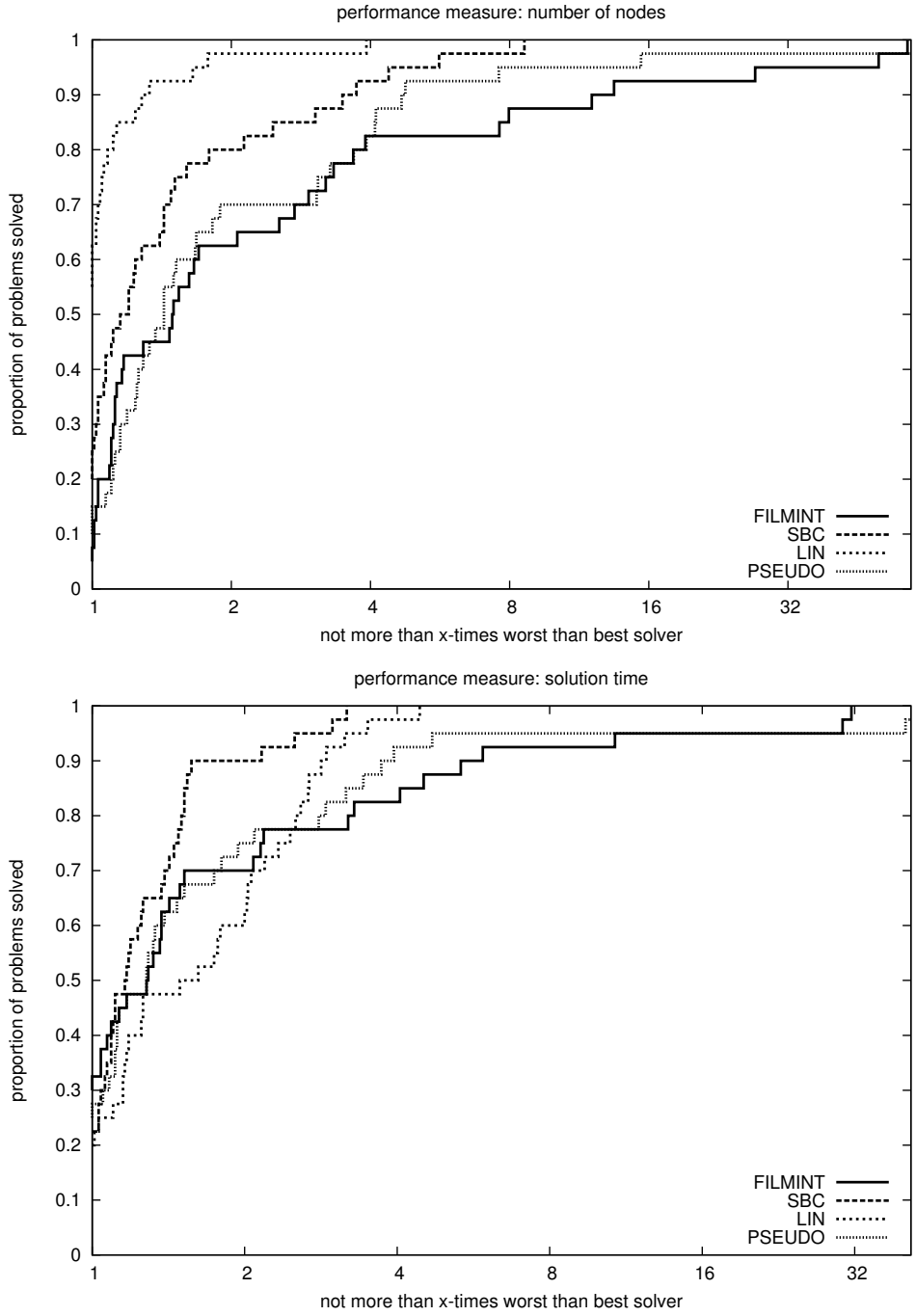
- `FILMINT`: The default version of FilMINT.
- `LIN`: FilMINT, but with the master problem augmented with linearizations from NLP-based branching subproblems, as described in Section 2.1.
- `SBC`: FilMINT, but with the master problem augmented with the simple strong-branching cuts (5).
- `PSEUDO`: FilMINT, but with an NLP-based strong-branching strategy in which strong-branching inequalities *are not added*. Rather, only the pseudocost value are initialized using the NLP-based strong branching information.

We included the method PSEUDO to test whether or not using valid inequalities derived from NLP-based strong branching can yield improvement beyond simply using the information for branching. After initializing the pseudocosts based on the solution values of (NLP_i^0) and (NLP_i^1), the pseudocosts are then updated based on FilMINT's default update strategy. FilMINT is based on the integer programming solver MINTO, and the pseudocost update strategy for MINTO is described in [28]. Since FilMINT is a linearization-based solver, updates to the pseudocosts are dependent on the outer approximation that has been obtained in the master problem.

Tables 7 and 8 in the appendix give the performance of each of these methods on each of the instances in our test suite. The tables are summarized in Figure 1, which consists of two performance profiles. The first profile uses the the number of nodes in the branch and bound tree as the solution metric. This profile indicates that *all* methods that incorporate NLP-based strong-branching information are useful for reducing the size of the branch and bound tree, but also that using strong branching information to derive valid inequalities in addition to making branching decisions can further reduce the size. The most effective method in terms of number of nodes is `LIN`. The second profile uses CPU time as the quality metric. In this measure, `SBC` is the best method.

The two profiles together paint the picture that simple strong branching cuts (5) can be an effective mechanism for improving performance of a linearization-based convex MINLP solver. The SBC inequalities are not as strong as adding all linearizations, but this is not a surprising result, as the SBC inequalities aggregate the linearization information into a single inequality. From the results of this experiment, we also conclude that a well-engineered mechanism for incorporating "useful" linearizations from points suggested by NLP-based strong branching, while not overwhelming the linear master problem (4) is likely to be the most effective "elementary" mechanism for making use of information from NLP-based strong-branching subproblems. We return to this idea in Section 4.6.

Fig. 1 Performance Profile of Elementary NLP-based Strong-Branching Inequalities



4.4 Performance of GUB-SBC Inequalities

A second experiment was designed to test the effectiveness of performing NLP-based strong branching on the GUB disjunction (6) and using the resulting GUBSBC inequality (7). Of primary interest is how the method performs compared to using only the disjunction on the individual binary variables via the simple SBC inequality (5).

In this experiment, if at least one of the variables in a GUB constraint is fractional at a solution to the master problem (4), then strong branching on the GUB constraint is performed, and a GUBSBC inequality (7) is generated. In order to generate a GUBSBC inequality, nonlinear subproblems (NLP_i^1) are solved for each of the variables in the GUB constraint, regardless of whether the variable value is fractional. In our implementation, at most one GUBSBC inequality is generated for each GUB, and the GUBSBC inequalities are generated at the root node only. If we encounter a fractional binary variable that is not in any GUB constraint, or we are not at the root node, then a simple SBC inequality (5) is generated for that variable.

Table 1 Solution Statistics Comparing SBC versus GUBSBC inequalities

	Extra gap closed (%)	**Average # of nodes**		**Average time**	
		Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
SBC	8.4	367291.8	57656.1	1557.9	431.2
GUBSBC	17.8	409115.7	59489.7	1558.7	470.1

In Table 1, we give computational results comparing the relative strength of SBC inequalities (5) and the GUBSBC inequalities (7). The detailed performance of methods on each instance is given in Table 9. The comparison is done for 31 instances from our test set of 40 problems for which there exists at least one GUB constraint in the problem. On average, adding GUBSBC inequalities closed 17.8% of the gap at root node, and adding only SBC inequalities closed 8.4%. It is then somewhat surprising that the number of nodes required for the two methods is approximately equal. For some reason, FilMINT seems to select less effective branching variables when GUBSBC inequalities are introduced to the master problem (4).

While adding GUBSBC inequalities can make a significant positive impact on solving some instances, our primary conclusion from this experiment is that the GUBSBC inequalities do not improve the performance of FilMINT more than the SBC inequalities, thus we focused our remaining computational experiments on evaluating only enhanced versions of SBC inequalities.

4.5 Performance of Mixing Strong-Branching Inequalities

We next compared the effectiveness of the mixed strong-branching inequalities (MIXSBC) (11) against the unmixed version (5). There may be exponentially many mixed strong branching inequalities, so we use the following strategy for adding them to the master problem. First, as in all of our methods, the NLP subproblems (NLP_i^0) and (NLP_i^1) are solved for each fractional variable x_i in the solution to the relaxed master problem (4). The fractional variables for which (NLP_i^0) and (NLP_i^1) have been solved define the mixing set $\bar{B}$. Next, for each variable in the mixing set, we add the *sparsest* MIXSBC inequality for that variable:

$$\eta \geq \underline{\eta} + \sigma_i x_i + (\sigma_h - \sigma_i) x_h \quad \forall i \in \bar{B}, \tag{18}$$

where $h = \text{argmax}_{i \in \bar{B}} \sigma_i$. Note that the sparsest MIXSBC inequality (18) already dominates the SBC inequality (5). Finally, after obtaining a fractional solution from the relaxation of the master problem, (after adding the inequalities (18)), the two most violated mixing inequalities are added and the relaxation is resolved. The MIXSBC inequalities are added in rounds until none are violated or until the inequalities do not change the relaxation solution by a sufficient amount. Specifically, if $\sum_{i \in B} |x_i' - x_i''| < 0.1$ for consecutive relaxation solutions x', x'', no further MIXSBC inequalities are added.

In Table 2, we summarize computational results comparing the effect of adding MIXSBC inequalities (11) with adding only SBC inequalities (5). The detailed performance of each method on each instance is given in Table 10. The MIXSBC inequalities are significantly stronger than the SBC inequalities. On average, MIXSBC closed 17.7% of the optimality gap at root node, and SBC closed only 6.9% of the

Table 2 Solution Statistics Comparing Mixing SBC versus SBC

	Extra gap closed (%)	**Average # of nodes**		**Average time**	
		Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
SBC	6.9	394689.0	54788.4	1585.6	504.9
MIXSBC	17.7	412063.4	55650.9	1689.2	521.4

gap on our test set. Despite this, MIXSBC inequalities perform *worse* than SBC in terms of average number of nodes and solution time. An explanation for this counterintuitive behavior is that the addition of the mixed strong-branching inequalities (11) results in MINTO (and hence FilMINT) performing "poor" updates on the pseudocost values for integer variables. That is, in subsequent branches, the pseudocosts do not accurately reflect the true change in objective value if a variable is branched on. Therefore, MIXSBC makes poor branching decisions, which in turn leads to a larger search tree. For example, MIXSBC closed 62.3% of the gap at the root node for the instance *SLay09M* and SBC closed only 23.6%. However, 85 seconds and 8545 nodes are required to prove optimality when using MIXSBC, compared to only 33 seconds and 4063 nodes for SBC alone.

4.6 Linearization Strategies

In our computational experiments we were not able to significantly improve the performance of strong-branching inequalities by exploiting GUB disjunctions or by mixing them. We therefore conclude that, among the strategies for obtaining strong-branching inequalities with minimal additional computational effort, adding linearizations from NLP strong-branching subproblems has the most potential as a computational technique. The performance profiles in Figure 1 indicate that linearizations are very effective in reducing the number of nodes, but often lead to unacceptable solution times due to the large number of linearizations added to the master problem. Two simple ideas to improve the performance of linearizations are to add only violated linearizations and to quickly remove linearizations that are not binding in the solution of the LP relaxation of the master problem.

In Table 3, we summarize computational results comparing our original linearization scheme `LIN` with an improved version denoted by `BESTLIN`. In `BESTLIN`, only linearization inequalities that are violated by the current relaxation solution are added, and if the dual variable for a linearization inequality has value zero for five consecutive relaxation solutions, the inequality is removed from the master problem. Full results of the performance of the two methods on each instance can be found in Table 11. The results show that without degrading the performance of `LIN` in terms of the number of nodes, `BESTLIN` can improve the average solution time from 2319.9 seconds to 1744.2 seconds.

Table 3 Solution Statistics Comparing LIN versus BESTLIN

	Average # of nodes		**Average time**	
	Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
LIN	353538.5	41021.2	2319.9	640.4
BESTLIN	376940.9	41245.3	1744.2	495.5

4.7 Performance of Multiplier-Strengthened SBC Inequalities

Initial computational experience with the multiplier-strengthened cuts SSBC (16), introduced in Section 3.2 suggested that the inequalities in general were far too dense to be effectively used in a linearization-based

scheme. Very quickly, the LP relaxation of the master problem (4) became prohibitively expensive to solve. Our remedy for these dense cuts was to strengthen only the coefficients of integer variables. Additionally, after the inequality was generated, the monoidal strengthening step described in Section 3.4 was performed on the inequalities.

An experiment was done to compare the performance of using the SBC inequalities against the SSBC inequalities on instances in our test set, and a summary of the results are given in Table 4. The computational results indicate a slight improvement in both the number of nodes and the solution time by strengthening the SBC inequalities using multipliers of the NLP strong-branching subproblems. Full results of the performance of the two methods on each instance can be found in Table 12.

Table 4 Solution Statistics Comparing Strengthened SBC versus SBC

	Extra gap closed (%)	**Average # of nodes**		**Average time**	
		Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
SBC	6.9	394689.0	54788.4	1585.6	504.9
SSBC	6.9	332509.4	51591.5	1535.4	490.9

4.8 Normalizations for CGLP

In Section 3.3, we described three different normalization constraints that are commonly used for the CGLP (17). We performed a small experiment to compare the relative effectiveness of each in our context. The performance profiles of Figure 2 summarize the results of comparing disjunctive inequalities generated by solving the CGLP (17) with the normalization constraints (αNORM) denoted by SBCGLP-INF, (SNC) denoted by SBCGLP-SNC, and (EN) denoted by SBCGLP-EN. The performance profiles show that both the standard normalization condition (SNC) and the Euclidean normalization (EN) perform significantly better than the α-normalization. The results are consistent with the literature. The performance of (SNC) and (EN) are comparable with each other, suggesting that the constraints of the instances in our test suite are well-scaled. The detailed performance of methods on each instance is given in Table 13.

4.9 CGLP Without Strong Branching

The CGLP described in Section 3.1 embeds linearization information from the solution of the two strong-branching subproblems (NLP_i^0) and (NLP_i^1). Specifically, the parameters $c^0, c^1, b^0, b^1, D^0, D^1, d^0, d^1$ defining the polyhedra $\mathcal{X}_i^0$ and $\mathcal{X}_i^1$ in (12) are defined in terms of the optimal solutions $(\hat{\eta}^0, \hat{x}^0)$ and $(\hat{\eta}^1, \hat{x}^1)$ to the NLP strong-branching subproblems (NLP_i^0) and (NLP_i^1). A different strategy, that does not use information from strong branching, is to define the polyhedra $\mathcal{X}_i^0$ and $\mathcal{X}_i^1$ using only linearization information from the solution of the current relaxation $(\hat{\eta}^{\text{LP}}, \hat{x}^{\text{LP}})$. That is, we could define the polyhedra using

$$\begin{aligned} c^0 &= c^1 = \nabla f(\hat{x}^{\text{LP}})^T, \\ b^0 &= b^1 = \nabla f(\hat{x}^{\text{LP}})^T \hat{x}^{\text{LP}} - \hat{\eta}^{\text{LP}}, \\ D^0 &= D^1 = \nabla g(\hat{x}^{\text{LP}})^T, \text{ and} \\ d^0 &= d^1 = \nabla g(\hat{x}^{\text{LP}})^T \hat{x}^{\text{LP}} - g(\hat{x}^{\text{LP}}). \end{aligned}$$

We performed an experiment aimed at quantifying the effect of using linearization information obtained from strong branching to create disjunctive cuts by comparing the performance of FilMINT augmented with these two types of disjunctive cuts.

Fig. 2 Performance Profile of CGLP with Different Normalization Constraints

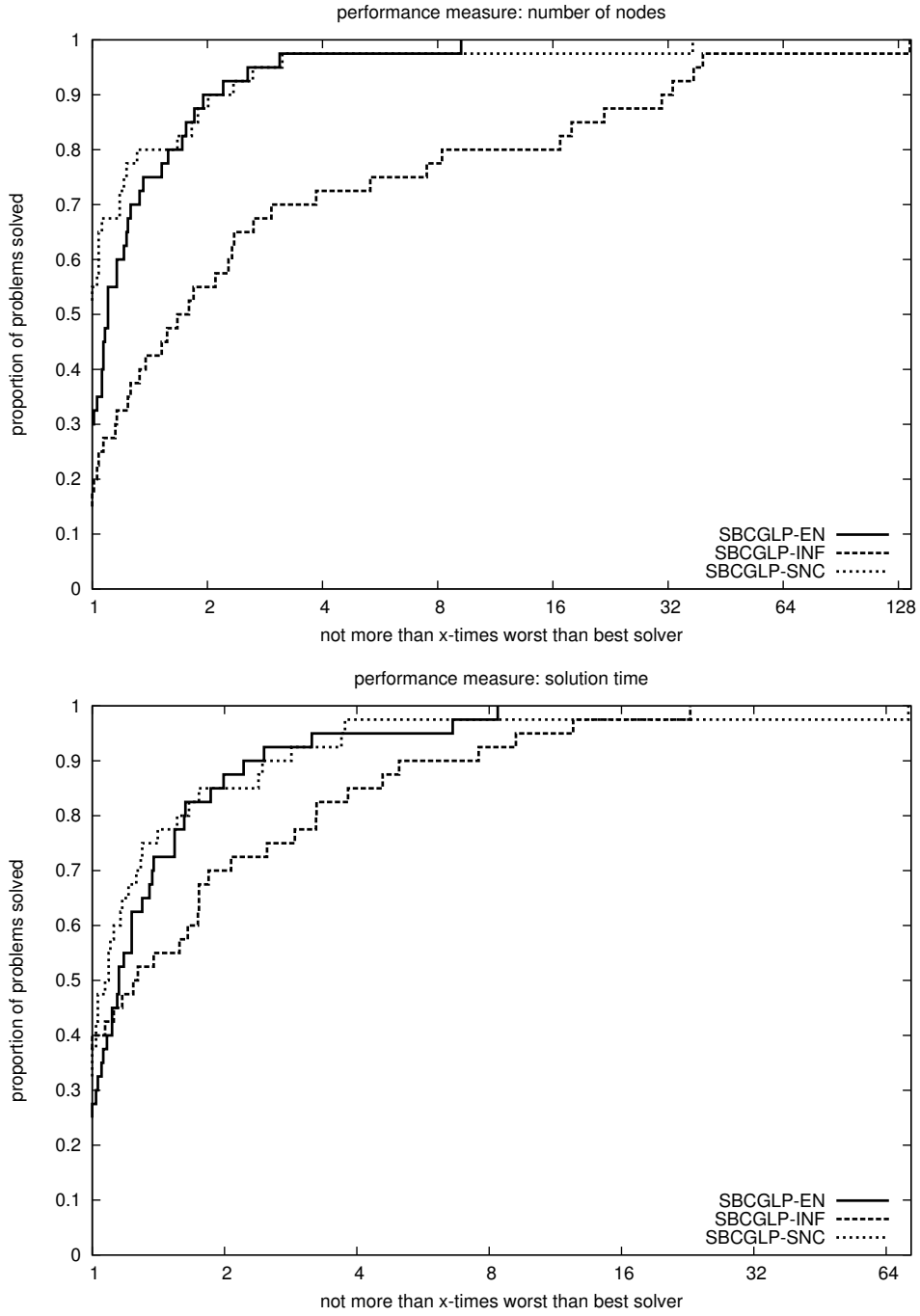


Table 14 reports the instance-specific results of this experiment. A summary of the results of this experiment are given in the form of two performance profiles in Figure 3. The profiles compare the performance of FilMINT using disjunctive cuts with linearizations from strong branching (`SBCGLP-SNC`) and FilMINT using disjunctive cuts with linearization from the current relaxation only (`NOSB-CGLP-SNC`). The top profile of the figure measures the performance in terms of number of nodes, and we can see clearly that by this measure, there is significant benefit to including linearizations obtained from NLP-based strong

branching in the CGLP. However, obtaining these linearizations by solving NLPs comes at some significant computational cost. This conclusion can be drawn by examining the second profile of Figure 3, where the performance measure is CPU time. In this measure, `NOSB-CGLP-SNC` outperforms `SBCGLP-SNC`. However, if the CPU time taking to perform strong branching is removed from the solution time calculation for the method `SBCGLP-SNC`, we obtained the results given by `SBCGLP-SNC-WO-SBTIME`, which dominates `NOSB-CGLP-SNC`. We conclude that *if* a branch-and-bound based algorithm performs NLP-based strong branching to determine the branching variables, then there is is significant positive effect in using the linearization information in the CGLP. However, one should *not* solve the strong-branching NLPs simply to obtain stronger disjunctive inequalities. The computational effort required in solving the NLPs outweighs the benefit obtained from stronger inequalities in terms of CPU time.

4.10 Comparison of All Methods

We make a final comparison of the methods introduced in the paper. In this experiment, we compare the methods that performed best in earlier experiments with the default version of FilMINT. The methods we compare are the following:

- `FilMINT`: The default version of FilMINT.
- `BESTLIN`: FilMINT, with the master problem augmented with linearizations from NLP-based branching subproblems, as described in Section 2.1. The linearization management strategy introduced in Section 4.6 is employed.
- `SSBC`: FilMINT, with the master problem augmented with the multiplier-strengthened strong-branching cuts (16).
- `SBCGLP-SNC`: FilMINT, adding disjunctive inequalities based on solving the CGLP (17) using the standard normalization condition (SNC).

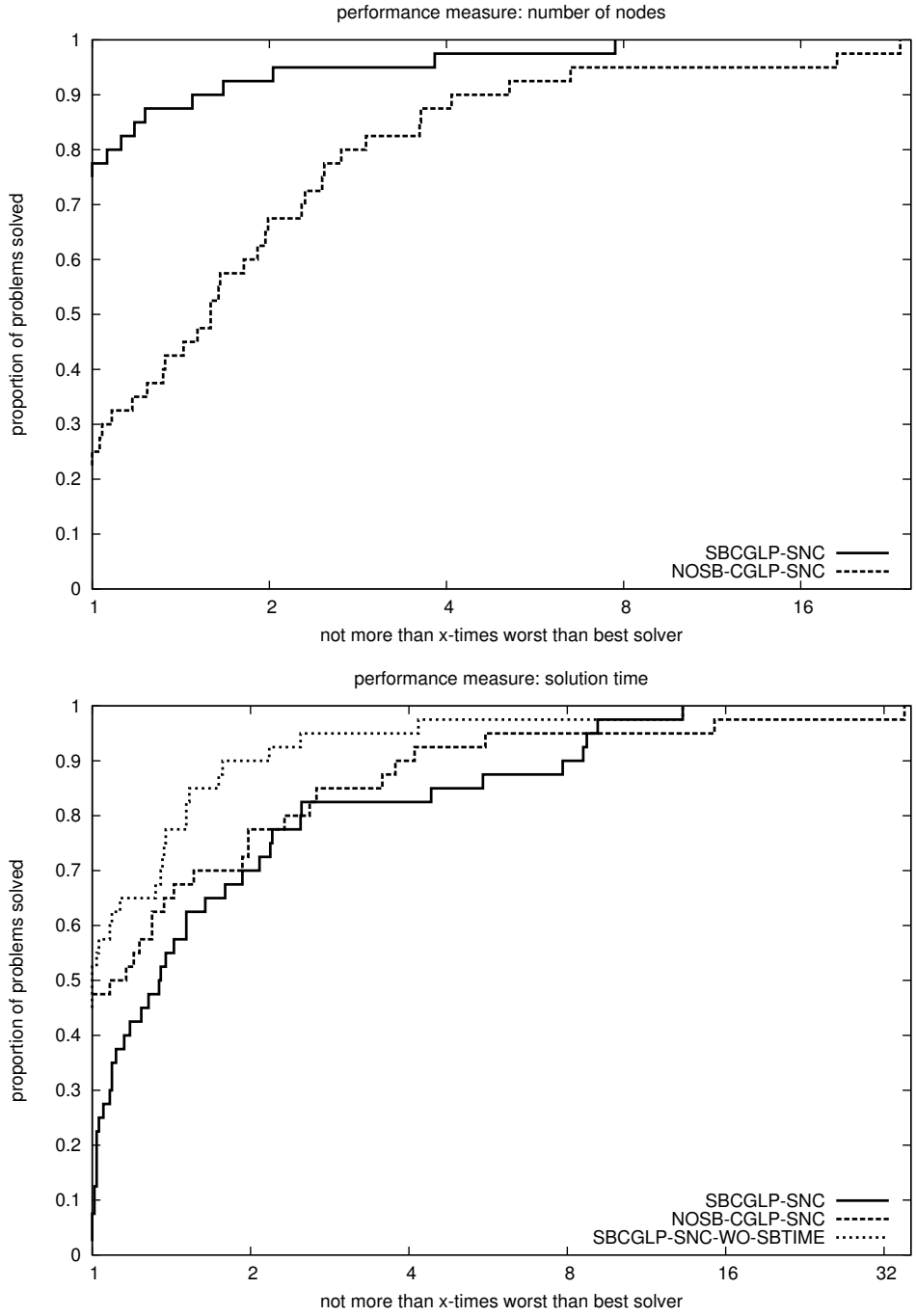
The monoidal strengthening step described in Section 3.4 was applied to the inequalities generated by methods `SSBC` and `SBCGLP-SNC`. In Table 5, we list the average number of nodes and solution time taken for the instances in our test set. The table shows that the method `SBCGLP-SNC` is the best for search tree size and solution time in the geometric mean, but the *worst* in both measures by the arithmetic mean. We conclude that the method `SBCGLP-SNC` can be a very effective method but some care must be taken in its use. For a small number of instances in our test set, in particular *o7*, *Safety3*, and *sssd-20-8-3*, the performance of `SBCGLP-SNC` is quite bad, requiring a very large number of nodes and large CPU time that significantly shifts the arithmetic mean measure.

A more holistic view is given by the performance profiles in Figure 4. These profiles show that in general creating disjunctive inequalities by solving the CGLP (17) with the standard normalization condition significantly outperforms the other methods. Creating disjunctive inequalities by an extra solve of (17) pays dividends both in terms of number of nodes and solution time. We experienced similar positive effects with other normalization constraints introduced in Section 3.3 as well. The profiles also indicate that both linearizations and SSBC inequalities improve the performance of default FilMINT substantially. The detailed performance of methods on each instance is given in Table 15 and Table 16.

Table 5 Solution Statistics Comparing Best Methods and FILMINT

	Average # of nodes		**Average time**	
	Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
FILMINT	762760.6	87019.1	2378.7	712.6
BESTLIN	376940.9	41245.3	1744.2	495.5
SSBC	332509.4	51591.5	1535.4	490.9
SBCGLP-SNC	833065.8	15922.6	2595.5	294.3

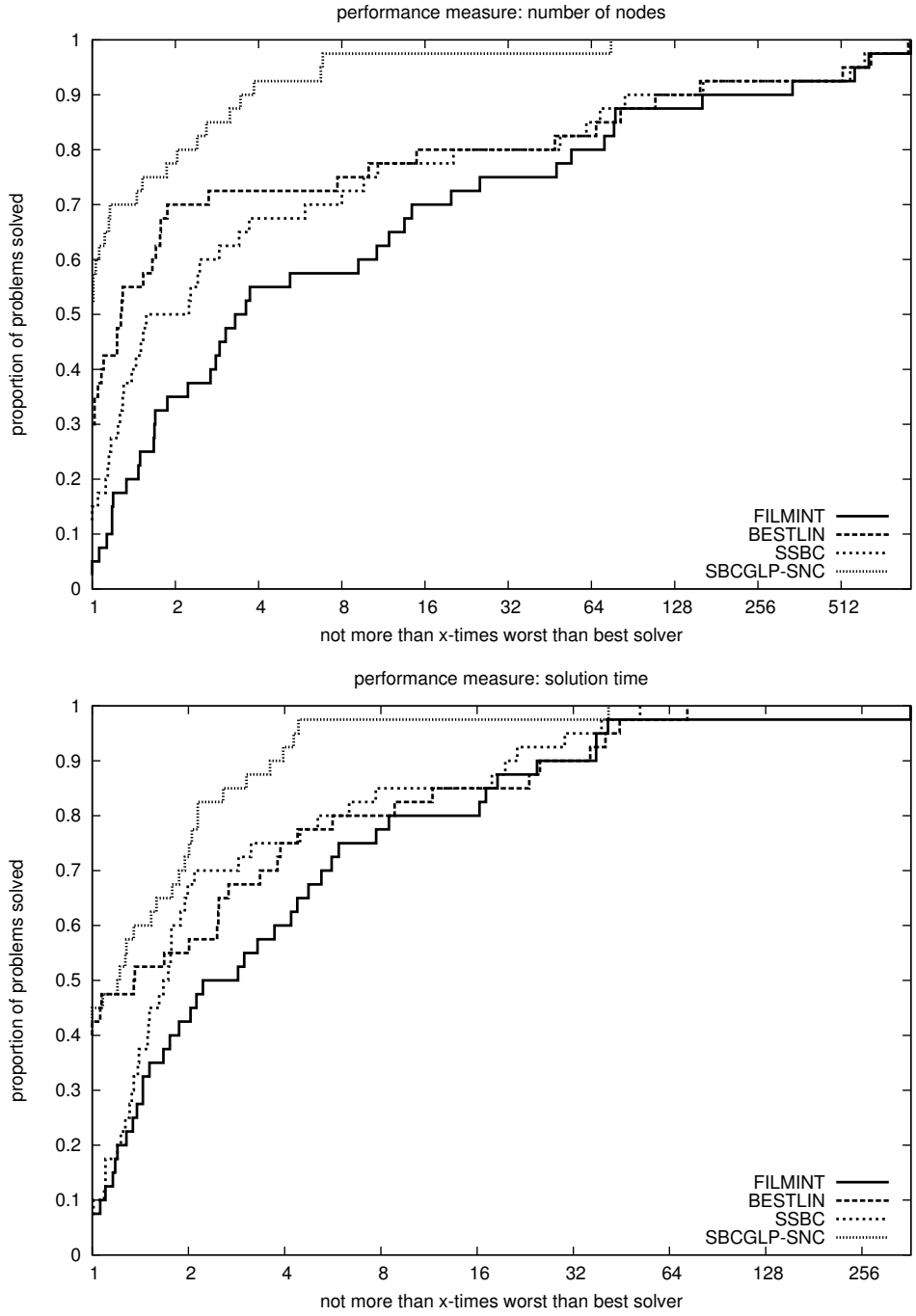
Fig. 3 Performance Profile of CGLP with and without Strong Branching



5 Conclusions

In this work, we demonstrate how to use "discarded" information generated from NLP-based strong branching to strengthen relaxations of MINLP problems. We first introduced strong-branching cuts, and we demonstrated the relation of strong-branching cuts we derive with other well-known disjunctive inequalities in the literature. We improved these basic cuts by using Lagrange multipliers and the integrality

Fig. 4 Performance Profile of Best Methods and FILMINT



of variables. We combined strong-branching cuts via mixing. We demonstrated that simple disjunctive inequalities can be improved by additional linearizations generated from strong-branching subproblems. Finally, the methods explained in this paper significantly improve the performance of FilMINT, justifying the use of strong branching based on nonlinear subproblems for solving convex MINLP problems.

Acknowledgements The authors would like to thank two anonymous referees for their useful comments and patience. This research was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy under Grant DE-FG02-08ER25861 and by the U.S. National Science Foundation under Grant CCF-0830153.

References

1. K. Abhishek, S. Leyffer, and J. T. Linderoth. FilMINT: An outer-approximation-based solver for nonlinear mixed integer programs. *INFORMS Journal on Computing*, 22:555–567, 2010.
2. T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technischen Universtät Berlin, 2007.
3. T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2004.
4. D. Applegate, R. Bixby, W. Cook, and V. Chvátal. *The Traveling Salesman Problem, A Computational Study*. Princeton University Press, 2006.
5. A. Atamtürk, G. Nemhauser, and M. W. P. Savelsbergh. Conflict graphs in solving integer programming problems. *European J. Operational Research*, 121:40–55, 2000.
6. E. Balas. A modified lift-and-project procedure. *Math. Program.*, 79:19–31, 1997.
7. E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3–44, 1998.
8. E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.
9. E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229–1246, 1996.
10. E. Balas and R. G. Jeroslow. Strengthening cuts for mixed integer programs. *European Journal of Operational Research*, 4:224–234, 1980.
11. P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.
12. P. Bonami, M. Kılınç, and J. Linderoth. Algorithms and software for solving convex mixed integer nonlinear programs. In *IMA Volumes in Mathematics and its Applications*, volume 54, pages 1–40, 2012.
13. P. Bonami, J. Lee, S. Leyffer, and A. Wächter. More branch-and-bound experiments in convex nonlinear integer programming. Preprint ANL/MCS-P1949-0911, Argonne National Laboratory, Mathematics and Computer Science Division, September 2011.
14. R R Boorstyn and H Frank. Large-scale network topological optimization. *IEEE Trans. Commun*, pages 29–47, 1997.
15. M. R. Bussieck, A. S. Drud, and A.Meeraus. MINLPLib – a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1), 2003.
16. I. Castillo, J. Westerlund, S. Emet, and T. Westerlund. Optimization of block layout deisgn problems with unequal areas: A comparison of MILP and MINLP optimization methods. *Computers and Chemical Engineering*, 30:54–69, 2005.
17. E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.
18. M. A. Duran and I. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.
19. S. Elhedhli. Service System Design with Immobile Servers, Stochastic Demand, and Congestion. *Manufacturing & Service Operations Management*, 8(1):92–97, 2006.
20. M. Fischetti, A. Lodi, and A. Tramontani. On the separation of disjunctive cuts. *Mathematical Programming*, pages 1–26, 2009.
21. R. Fletcher and S. Leyffer. User manual for filterSQP, 1998. University of Dundee Numerical Analysis Report NA-181.
22. I. E. Grossmann. Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, 3:227–252, 2002.
23. O. Günlük, J. Lee, and R. Weismantel. MINLP strengthening for separaable convex quadratic transportation-cost ufl. Technical Report RC24213 (W0703-042), IBM Research Division, March 2007.
24. O. Günlük and Y. Pochet. Mixing mixed-integer inequalities. *Mathematical Programming*, 90(3):429 – 457, 2001.
25. I. Harjunkoski, R. Pörn, and T. Westerlund. MINLP: Trim-loss problem. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 2190–2198. Springer, 2009.
26. M. Kılınç. *Disjunctive Cutting Planes and Algorithms for Convex Mixed Integer Nonlinear Programming,*. PhD thesis, University of Wisconsin-Madison, 2011.
27. S. Leyffer. MacMINLP: Test problems for mixed integer nonlinear programming, 2003. `http://www.mcs.anl.gov/~leyffer/macminlp`.
28. J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies in mixed integer programming. *INFORMS Journal on Computing*, 11:173–187, 1999.
29. G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.
30. Y. Pochet and L. Wolsey. Lot sizing with constant batches: Formulation and valid inequalities. *Mathematics of Operations Research*, 18:767–785, 1993.

31. I. Quesada and I. E. Grossmann. An LP/NLP based branch–and–bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.
32. D. E. Ravemark and D. W. T. Rippin. Optimal design of a multi-product batch plant. *Computers & Chemical Engineering*, 22(1-2):177 – 183, 1998.
33. N. Sawaya. *Reformulations, relaxations and cutting planes for generalized disjunctive programming*. PhD thesis, Chemical Engineering Department, Carnegie Mellon University, 2006.
34. N. Sawaya, C. D. Laird, L. T. Biegler, P. Bonami, A. R. Conn, G. Cornuéjols, I. E. Grossmann, J. Lee, A. Lodi, F. Margot, and A. Wächter. CMU-IBM open source MINLP project test set, 2006. `http://egon.cheme.cmu.edu/ibm/page.htm`.
35. A. Saxena, P. Bonami, and J. Lee. Convex relaxations of non-convex mixed integer quadratically constrained programs: Extended formulations. *Mathematical Programming, Series B*, 124:383–411, 2010.
36. R. Stubbs and S. Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86:515–532, 1999.
37. M. Türkay and I. E. Grossmann. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers & Chemical Engineering*, 20(8):959 – 978, 1996.
38. A. Vecchietti and I. E. Grossmann. LOGMIP: a disjunctive 0-1 non-linear optimizer for process system models. *Computers and Chemical Engineering*, 23(4-5):555 – 565, 1999.

Appendix

Table 7: Number of Nodes for `FILMINT`, `SBC`, `LIN` and `PSEUDO`. Experiment is described in Section 4.3

Problem	FILMINT	SBC	LIN	PSEUDO
BatchS151208M	4097	2853	1989	2187
BatchS201210M	4001	2521	1025	1407
Batch_Storage10_BM_10_10_6_4	18647	12675	10999	16731
Batch_Storage_BM_10_10_6_4	21433	11015	7293	9407
CLay0305H	10595	10039	9727	11153
FLay05H	99681	99693	99953	98449
FLay05M	93765	83409	81017	86295
fo7_2	84269	75711	100381	93917
fo7	265115	381401	257719	1053587
m7	587399	77263	303051	146123
nd-12	25139	10765	13429	7541
nd-13	22953	10007	8691	8365
nd-14	98507	97573	102023	97537
o7_2	1653741	1109651	1172195	2022459
o7	4025281	2743271	3398533	8962473
RSyn0810M02M	122077	113667	110969	124563
RSyn0810M03M	93807	106807	83659	99647
RSyn0810M04M	131657	101741	112685	135623
RSyn0815M02M	441187	458867	286589	430839
RSyn0820M02M	2126481	2341249	2138451	1892201
RSyn0830M02M	798717	881683	797105	1139617
Safety3	4211361	1317425	2177111	1877557
safety_no_rotation_CH	23267	5239	2919	3635
SLay07H	3615	1145	133	521
SLay08H	9989	2517	829	3411
SLay09H	117219	8839	2019	15301
SLay09M	15729	4063	1169	5563
sssd-16-8-3	2026245	2885523	1353391	6321719
sssd-17-7-3	1314097	518221	557645	1598469
sssd-18-7-3	602883	924233	164115	2527359
sssd-20-8-3	10870313	807023	216121	1525875
Syn20M04M	90763	89253	91343	149685
Syn30M03M	49627	53087	50895	182599
Syn30M04M	257063	251043	250663	766969
Syn40M02M	153035	147023	138941	139131

Continued on next page

Table 7 – continued from previous page

Problem	FILMINT	SBC	LIN	PSEUDO
trimloss4	19159	26849	24515	24061
uflquad-15-60	2415	2275	2061	2369
uflquad-15-80	3577	3413	3179	3527
uflquad-20-40	4517	3351	2783	3981
uflquad-25-40	7001	5177	4225	7057
Arithmetic mean	762760.6	394689	353538.5	789972.8
Geometric mean	87019.1	54788.4	41021.2	70326.6

Table 8: Solution Time for FILMINT, SBC, LIN and PSEUDO. Experiment is described in Section 4.3

Problem	FILMINT	SBC	LIN	PSEUDO
BatchS151208M	55.7	51.9	61.9	49.2
BatchS201210M	65.3	61.1	50.4	43.8
Batch_Storage10_BM_10_10_6_4	189.7	162.8	138.4	176.8
Batch_Storage_BM_10_10_6_4	208.5	141.5	98.5	95.8
CLay0305H	59.6	42.1	62.8	53.8
FLay05H	2501.6	2515.0	2902.7	2298.7
FLay05M	1161.1	1095.1	1155.6	994.6
fo7_2	151.4	167.8	331.3	210.5
fo7	454.1	690.5	498.6	2129.4
m7	1325.7	123.2	545.1	238.9
nd-12	2423.8	875.8	2381.3	756.5
nd-13	529.1	1577.2	1852.7	921.0
nd-14	4948.2	9666.7	10317.9	3848.9
o7_2	3714.9	2719.0	4399.9	4133.6
o7	8929.3	6772.7	7955.8	21451.6
RSyn0810M02M	364.4	403.0	650.8	394.6
RSyn0810M03M	477.4	724.8	1276.1	637.1
RSyn0810M04M	1008.4	1042.4	2078.7	1176.8
RSyn0815M02M	1531.4	1671.0	2519.6	1426.3
RSyn0820M02M	9170.8	10553.1	16558.4	6743.0
RSyn0830M02M	4142.6	5102.7	9651.1	4636.5
Safety3	18232.8	5540.1	11065.3	7152.5
safety_no_rotation_CH	122.0	25.9	20.7	21.8
SLay07H	47.4	16.2	10.5	11.8
SLay08H	181.5	53.4	44.8	66.1
SLay09H	3031.1	215.9	100.1	394.6
SLay09M	123.4	33.5	23.1	48.4
sssd-16-8-3	1517.9	2156.5	3078.1	4392.1
sssd-17-7-3	854.2	397.4	1024.9	1363.7
sssd-18-7-3	430.1	659.0	207.0	8343.4
sssd-20-8-3	19030.5	621.9	604.0	43200.4
Syn20M04M	220.6	277.5	640.2	396.4
Syn30M03M	204.9	281.1	580.3	761.7
Syn30M04M	1623.7	1762.2	3278.0	4542.6
Syn40M02M	465.1	498.1	1169.1	464.4
trimloss4	47.2	71.0	82.0	52.7
uflquad-15-60	607.2	586.2	689.3	652.4
uflquad-15-80	2017.3	2003.5	2412.1	1932.9
uflquad-20-40	831.9	651.3	656.2	721.6
uflquad-25-40	2146.9	1413.7	1624.0	1865.9
Arithmetic mean	2378.7	1585.6	2319.9	3220.1
Geometric mean	712.6	504.9	640.4	693.2

Table 6 Test Set Statistics

Problem	NL Obj	Vars	Ints	Cons	NL Cons	GUBs
BatchS151208M	√	446	203	1780	1	24
BatchS201210M	√	559	251	2326	1	24
BatchStorage10BM101064	√	239	89	798	1	20
BatchStorageBM101064	√	239	89	798	1	20
CLay0305H		276	55	336	60	15
FLay05H		383	40	461	5	10
FLay05M		63	40	61	5	10
fo7_2		115	42	198	14	0
fo7		115	42	198	14	0
m7		115	42	198	14	0
nd-12		601	40	290	40	4
nd-13		641	40	317	40	2
nd-14		817	48	370	48	2
o7_2		115	42	198	14	0
o7		115	42	198	14	0
RSyn0810M02M		411	168	855	12	200
RSyn0810M03M		616	252	1435	18	435
RSyn0810M04M		821	336	2117	24	772
RSyn0815M02M		471	188	960	22	236
RSyn0820M02M		511	208	1047	28	266
RSyn0830M02M		621	248	1233	40	322
Safety3	√	260	98	294	0	28
safety_no_rotation_CH	√	409	60	456	6	15
SLay07H	√	477	84	609	0	21
SLay08H	√	633	112	812	0	28
SLay09H	√	811	144	1044	0	36
SLay09M	√	235	144	324	0	36
sssd-16-8-3		185	152	57	24	24
sssd-17-7-3		169	140	53	21	24
sssd-18-7-3		176	147	54	21	25
sssd-20-8-3		217	184	61	24	28
Syn20M04M		421	160	997	56	462
Syn30M03M		481	180	982	60	386
Syn30M04M		641	240	1489	80	684
Syn40M02M		421	160	757	56	244
trimloss4		106	85	61	4	20
uflquad-15-60	√	916	15	960	0	0
uflquad-15-80	√	1216	15	1280	0	0
uflquad-20-40	√	821	20	840	0	0
uflquad-25-40	√	1026	25	1040	0	0

Table 9: Computational Results Comparing `SBC` and `GUBSBC` Inequalities. Experiment is described in Section 4.4

Problem	SBC		GUBSBC	
	Node	Time	Node	Time
BatchS151208M	2853	51.9	2637	64.6
BatchS201210M	2521	61.1	3431	107.9
Batch_Storage10_BM_10_10_6_4	12675	162.8	12327	135.2
Batch_Storage_BM_10_10_6_4	11015	141.5	10749	122.2
CLay0305H	10039	42.1	15231	39.2
FLay05H	99693	2515.0	103261	2732.8
FLay05M	83409	1095.1	88833	1191.8
nd-12	10765	875.8	10957	979.1
nd-13	10007	1577.2	6909	744.7
nd-14	97573	9666.7	103817	10344.2

Continued on next page

Table 9 – continued from previous page

Problem	SBC		GUBSBC	
	Node	**Time**	**Node**	**Time**
RSyn0810M02M	113667	403.0	111769	421.1
RSyn0810M03M	106807	724.8	90209	843.5
RSyn0810M04M	101741	1042.4	114027	1321.6
RSyn0815M02M	458867	1671.0	449381	1668.3
RSyn0820M02M	2341249	10553.1	1388191	6714.8
RSyn0830M02M	881683	5102.7	929581	5332.2
Safety3	1317425	5540.1	1517789	6782.6
safety_no_rotation_CH	5239	25.9	5549	29.5
SLay07H	1145	16.2	1019	15.3
SLay08H	2517	53.4	3947	72.0
SLay09H	8839	215.9	14227	346.9
SLay09M	4063	33.5	13211	122.0
sssd-16-8-3	2885523	2156.5	5407233	3613.6
sssd-17-7-3	518221	397.4	435421	301.0
sssd-18-7-3	924233	659.0	390357	283.4
sssd-20-8-3	807023	621.9	1031651	893.8
Syn20M04M	89253	277.5	89589	379.4
Syn30M03M	53087	281.1	47673	309.5
Syn30M04M	251043	1762.2	117043	1776.0
Syn40M02M	147023	498.1	138897	559.0
trimloss4	26849	71.0	27671	73.4
Arithmetic mean	367291.8	1557.9	409115.7	1558.7
Geometric mean	57656.1	431.2	59489.7	470.1

Table 10: Computational Results Comparing SBC versus MIXSBC. Experiment is described in Section 4.5

Problem	SBC		MIXSBC	
	Node	**Time**	**Node**	**Time**
BatchS151208M	2853	51.9	2981	85.5
BatchS201210M	2521	61.1	3107	104.8
Batch_Storage10_BM_10_10_6_4	12675	162.8	9825	114.7
Batch_Storage_BM_10_10_6_4	11015	141.5	8091	96.6
CLay0305H	10039	42.1	12423	60.5
FLay05H	99693	2515.0	99445	2460.5
FLay05M	83409	1095.1	91981	1198.0
fo7_2	75711	167.8	75711	159.3
fo7	381401	690.5	381401	657.7
m7	77263	123.2	77263	114.4
nd-12	10765	875.8	8765	628.0
nd-13	10007	1577.2	7597	680.5
nd-14	97573	9666.7	107587	12390.7
o7_2	1109651	2719.0	1109651	2591.4
o7	2743271	6772.7	2743271	6177.0
RSyn0810M02M	113667	403.0	116547	402.8
RSyn0810M03M	106807	724.8	91953	647.3
RSyn0810M04M	101741	1042.4	109603	1030.2
RSyn0815M02M	458867	1671.0	389003	1472.3
RSyn0820M02M	2341249	10553.1	2021433	7995.5
RSyn0830M02M	881683	5102.7	881683	4644.3
Safety3	1317425	5540.1	2282213	10623.0
safety_no_rotation_CH	5239	25.9	4949	30.4
SLay07H	1145	16.2	649	14.2
SLay08H	2517	53.4	1931	47.1
SLay09H	8839	215.9	16157	344.0

Continued on next page

Table 10 – continued from previous page

Problem	SBC		MIXSBC	
	Node	**Time**	**Node**	**Time**
SLay09M	4063	33.5	8545	84.6
sssd-16-8-3	2885523	2156.5	2859127	2010.0
sssd-17-7-3	518221	397.4	339627	229.2
sssd-18-7-3	924233	659.0	1290693	791.8
sssd-20-8-3	807023	621.9	751119	583.0
Syn20M04M	89253	277.5	93665	251.5
Syn30M03M	53087	281.1	53087	260.7
Syn30M04M	251043	1762.2	251043	1659.5
Syn40M02M	147023	498.1	136903	537.0
trimloss4	26849	71.0	26905	63.9
uflquad-15-60	2275	586.2	2509	732.6
uflquad-15-80	3413	2003.5	3473	2021.0
uflquad-20-40	3351	651.3	4207	962.4
uflquad-25-40	5177	1413.7	6411	2611.6
Arithmetic mean	394689.0	1585.6	412063.4	1689.2
Geometric mean	54788.4	504.9	55650.9	521.4

Table 11: Computational Results Comparing LIN versus BESTLIN. Experiment is described in Section 4.6

Problem	LIN		BESTLIN	
	Node	**Time**	**Node**	**Time**
BatchS151208M	1989	61.9	2193	52.7
BatchS201210M	1025	50.4	1111	45.5
Batch_Storage10_BM_10_10_6_4	10999	138.4	10307	108.4
Batch_Storage_BM_10_10_6_4	7293	98.5	10335	109.3
CLay0305H	9727	62.8	11077	58.2
FLay05H	99953	2902.7	101823	2083.9
FLay05M	81017	1155.6	85797	981.5
fo7_2	100381	331.3	133599	344.5
fo7	257719	498.6	278535	391.4
m7	303051	545.1	187299	265.6
nd-12	13429	2381.3	12611	1604.4
nd-13	8691	1852.7	8377	1412.3
nd-14	102023	10317.9	102169	6723.5
o7_2	1172195	4399.9	1891701	5564.0
o7	3398533	7955.8	3511795	6184.1
RSyn0810M02M	110969	650.8	102967	470.0
RSyn0810M03M	83659	1276.1	78845	874.1
RSyn0810M04M	112685	2078.7	98627	1479.6
RSyn0815M02M	286589	2519.6	435263	2192.2
RSyn0820M02M	2138451	16558.4	1984789	10858.7
RSyn0830M02M	797105	9651.1	843825	6789.1
Safety3	2177111	11065.3	2446389	9328.3
safety_no_rotation_CH	2919	20.7	2173	14.4
SLay07H	133	10.5	143	9.1
SLay08H	829	44.8	741	32.4
SLay09H	2019	100.1	1501	73.8
SLay09M	1169	23.1	1097	16.0
sssd-16-8-3	1353391	3078.1	1312505	2716.1
sssd-17-7-3	557645	1024.9	444949	775.9
sssd-18-7-3	164115	207.0	206693	373.3
sssd-20-8-3	216121	604.0	200725	504.7
Syn20M04M	91343	640.2	89111	410.3
Syn30M03M	50895	580.3	50557	402.6

Continued on next page

Table 11 – continued from previous page

Problem	LIN		BESTLIN	
	Node	**Time**	**Node**	**Time**
Syn30M04M	250663	3278.0	251895	2204.6
Syn40M02M	138941	1169.1	138493	984.8
trimloss4	24515	82.0	25919	50.4
uflquad-15-60	2061	689.3	1809	474.2
uflquad-15-80	3179	2412.1	3041	1337.0
uflquad-20-40	2783	656.2	2679	498.6
uflquad-25-40	4225	1624.0	4169	967.0
Arithmetic mean	353538.5	2319.9	376940.9	1744.2
Geometric mean	41021.2	640.4	41245.3	495.5

Table 12: Computational Results Comparing SBC versus SSBC. Experiment is described in Section 4.7

Problem	SBC		SSBC	
	Node	**Time**	**Node**	**Time**
BatchS151208M	2853	51.9	2853	53.0
BatchS201210M	2521	61.1	2521	63.1
Batch_Storage10_BM_10_10_6_4	12675	162.8	11681	151.8
Batch_Storage_BM_10_10_6_4	11015	141.5	12225	177.2
CLay0305H	10039	42.1	10039	43.0
FLay05H	99693	2515.0	99693	2506.3
FLay05M	83409	1095.1	83409	1081.7
fo7_2	75711	167.8	75711	169.5
fo7	381401	690.5	381401	692.0
m7	77263	123.2	105467	189.0
nd-12	10765	875.8	8821	969.5
nd-13	10007	1577.2	8187	881.7
nd-14	97573	9666.7	97573	9646.8
o7_2	1109651	2719.0	1109651	2762.5
o7	2743271	6772.7	2743271	6761.0
RSyn0810M02M	113667	403.0	111167	422.7
RSyn0810M03M	106807	724.8	98249	706.2
RSyn0810M04M	101741	1042.4	134163	1506.6
RSyn0815M02M	458867	1671.0	454679	1674.2
RSyn0820M02M	2341249	10553.1	2051391	9520.4
RSyn0830M02M	881683	5102.7	874439	5065.1
Safety3	1317425	5540.1	1383869	5542.7
safety_no_rotation_CH	5239	25.9	5239	25.5
SLay07H	1145	16.2	1145	16.0
SLay08H	2517	53.4	2517	52.5
SLay09H	8839	215.9	8839	211.9
SLay09M	4063	33.5	4063	33.4
sssd-16-8-3	2885523	2156.5	1572157	1530.6
sssd-17-7-3	518221	397.4	288299	224.8
sssd-18-7-3	924233	659.0	397461	298.5
sssd-20-8-3	807023	621.9	577703	679.5
Syn20M04M	89253	277.5	91351	296.2
Syn30M03M	53087	281.1	47999	267.3
Syn30M04M	251043	1762.2	254651	1872.0
Syn40M02M	147023	498.1	147475	534.6
trimloss4	26849	71.0	26705	61.8
uflquad-15-60	2275	586.2	2275	601.7
uflquad-15-80	3413	2003.5	3413	2002.4
uflquad-20-40	3351	651.3	3451	663.3
uflquad-25-40	5177	1413.7	5171	1459.6

Continued on next page

Table 12 – continued from previous page

Problem	SBC		SSBC	
	Node	**Time**	**Node**	**Time**
Arithmetic mean	394689.0	1585.6	332509.4	1535.4
Geometric mean	54788.4	504.9	51591.5	490.9

Table 13: Computational Results Comparing Normalization Conditions αNORM, SNC and EN. Experiment is described in Section 4.8

Problem	SBCGLP-EN		SBCGLP-SNC		SBCGLP-INF	
	Node	**Time**	**Node**	**Time**	**Node**	**Time**
BatchS151208M	2089	58.9	2221	63.2	4397	102.4
BatchS201210M	2269	78.5	2067	81.1	2755	83.9
BatchS10BM101064	10457	133.9	8403	108.7	19507	189.8
BatchSBM101064	8581	114.5	7993	102.7	12131	114.8
CLay0305H	11465	67.3	10683	54.6	17883	75.4
FLay05H	116703	3395.2	110929	3182.4	107597	2517.0
FLay05M	84107	1123.8	79487	1062.5	79811	822.3
fo7_2	58463	180.2	50597	138.2	195049	448.1
fo7	521569	1301.2	538915	1410.5	296871	589.2
m7	2953	6.0	1719	3.7	9157	11.8
nd-12	14913	1268.4	7649	651.5	8863	638.9
nd-13	9083	1109.0	8293	1032.2	8337	1029.0
nd-14	153795	16315.1	83049	9269.5	86379	2466.9
o7_2	1966133	5572.3	2663961	8401.1	1596783	3439.8
o7	6595207	18856.6	8634197	24504.7	12127783	29773.0
RSyn0810M02M	11343	87.8	10295	83.2	84545	381.1
RSyn0810M03M	9827	200.4	10221	225.2	73641	764.9
RSyn0810M04M	3509	239.0	6617	337.2	115439	1808.4
RSyn0815M02M	7497	85.4	9223	94.0	230969	1060.2
RSyn0820M02M	37439	279.3	29831	242.8	1113583	5570.5
RSyn0830M02M	8573	144.2	10339	168.2	186921	1326.4
Safety3	3971759	20254.3	4753685	23666.1	1806949	6405.5
safety_no_rotation_CH	2799	21.1	5623	36.9	5013	26.8
SLay07H	311	11.9	977	18.6	705	14.8
SLay08H	2417	63.2	2855	69.1	2587	57.1
SLay09H	11679	315.2	10083	326.5	12493	300.6
SLay09M	5149	50.5	1669	20.5	4401	37.7
sssd-16-8-3	1788795	1510.2	701601	811.8	2062375	1337.7
sssd-17-7-3	994571	600.5	252627	204.0	107961	71.7
sssd-18-7-3	256585	173.5	162115	150.4	381639	263.8
sssd-20-8-3	553445	445.8	15112159	20816.1	407137	289.4
Syn20M04M	571	47.5	563	46.5	9411	134.5
Syn30M03M	117	34.8	77	34.7	3039	71.7
Syn30M04M	239	90.2	281	87.6	32767	436.9
Syn40M02M	324	26.4	267	27.3	4783	66.0
trimloss4	22591	72.6	16963	57.6	23389	52.7
uflquad-15-60	2029	797.5	2103	754.4	2547	649.0
uflquad-15-80	3347	2673.1	3493	2865.4	3459	1730.7
uflquad-20-40	2933	761.9	2763	671.2	4351	787.2
uflquad-25-40	6227	2246.3	6039	1939.2	6915	1899.9
Arithmetic mean	431546.6	2020.3	833065.8	2595.5	531508.1	1696.2
Geometric mean	16129.3	290.4	15922.6	294.3	37453.7	381.6

Table 14: Computational Results Comparing SBCGLP-SNC versus NOSB-CGLP-SNC. Experiment is described in Section 4.9

Problem	SBCGLP-SNC			NOSB-CGLP-SNC	
	Node	Time(noSB)	Time	Node	Time
BatchS151208M	2221	53.4	63.2	3527	62.1
BatchS201210M	2067	63.1	81.1	4697	86.2
Batch_Storage10_BM_10_10_6_4	8403	106.2	108.7	22257	210.6
Batch_Storage_BM_10_10_6_4	7993	99.9	102.7	23367	232.2
CLay0305H	10683	52.2	54.6	14067	74.8
FLay05H	110929	3179.5	3182.4	98809	2104.8
FLay05M	79487	1062.4	1062.5	81819	974.9
fo7_2	50597	138.0	138.2	99441	273.4
fo7	538915	1410.3	1410.5	265121	566.9
m7	1719	3.4	3.7	3423	4.4
nd-12	7649	639.3	651.5	18787	1657.6
nd-13	8293	1015.2	1032.2	13701	468.4
nd-14	83049	9243.4	9269.5	158775	17835.1
o7_2	2663961	8400.8	8401.1	2509627	6244.6
o7	8634197	24504.5	24504.7	7310483	17817.5
RSyn0810M02M	10295	52.0	83.2	10311	38.1
RSyn0810M03M	10221	108.4	225.2	25319	169.2
RSyn0810M04M	6617	132.8	337.2	4481	76.4
RSyn0815M02M	9223	50.0	94.0	13175	48.7
RSyn0820M02M	29831	183.2	242.8	7805	43.9
RSyn0830M02M	10339	91.4	168.2	12131	67.2
Safety3	4753685	23664.1	23666.1	3858601	15682.2
safety_no_rotation_CH	5623	33.2	36.9	28773	135.9
SLay07H	977	11.3	18.6	3521	42.7
SLay08H	2855	48.4	69.1	10329	172.1
SLay09H	10083	284.3	326.5	65571	1590.3
SLay09M	1669	16.5	20.5	30783	251.6
sssd-16-8-3	701601	808.9	811.8	2860827	2156.1
sssd-17-7-3	252627	201.7	204.0	415133	248.3
sssd-18-7-3	162115	147.8	150.4	3829325	5172.7
sssd-20-8-3	15112159	20812.1	20816.1	1951923	1569.4
Syn20M04M	563	6.7	46.5	849	5.9
Syn30M03M	77	4.0	34.7	177	4.4
Syn30M04M	281	9.8	87.6	375	9.6
Syn40M02M	267	4.8	27.3	160	3.1
trimloss4	16963	57.2	57.6	30761	68.5
uflquad-15-60	2103	745.1	754.4	2261	562.8
uflquad-15-80	3493	2844.8	2865.4	3635	1603.9
uflquad-20-40	2763	661.2	671.2	4397	861.0
uflquad-25-40	6039	1918.8	1939.2	7481	1780.8
Arithmetic mean	833065.8	2571.8	2595.5	595150.1	2024.4
Geometric mean	15922.6	209.0	294.3	26181.9	262.7

Table 15: Number of Nodes for FILMINT, BESTLIN, SSBC, and SBCGLP-SNC. Experiment is described in Section 4.10

Problem	FILMINT	BESTLIN	SSBC	SBCGLP-SNC
BatchS151208M	4097	2193	2853	2221
BatchS201210M	4001	1111	2521	2067
Batch_Storage10_BM_10_10_6_4	18647	10307	11681	8403
Batch_Storage_BM_10_10_6_4	21433	10335	12225	7993
CLay0305H	10595	11077	10039	10683
FLay05H	99681	101823	99693	110929

Continued on next page

Table 15 – continued from previous page

Problem	FILMINT	BESTLIN	SSBC	SBCGLP-SNC
FLay05M	93765	85797	83409	79487
fo7_2	84269	133599	75711	50597
fo7	265115	278535	381401	538915
m7	587399	187299	105467	1719
nd-12	25139	12611	8821	7649
nd-13	22953	8377	8187	8293
nd-14	98507	102169	97573	83049
o7_2	1653741	1891701	1109651	2663961
o7	4025281	3511795	2743271	8634197
RSyn0810M02M	122077	102967	111167	10295
RSyn0810M03M	93807	78845	98249	10221
RSyn0810M04M	131657	98627	134163	6617
RSyn0815M02M	441187	435263	454679	9223
RSyn0820M02M	2126481	1984789	2051391	29831
RSyn0830M02M	798717	843825	874439	10339
Safety3	4211361	2446389	1383869	4753685
safety_no_rotation_CH	23267	2173	5239	5623
SLay07H	3615	143	1145	977
SLay08H	9989	741	2517	2855
SLay09H	117219	1501	8839	10083
SLay09M	15729	1097	4063	1669
sssd-16-8-3	2026245	1312505	1572157	701601
sssd-17-7-3	1314097	444949	288299	252627
sssd-18-7-3	602883	206693	397461	162115
sssd-20-8-3	10870313	200725	577703	15112159
Syn20M04M	90763	89111	91351	563
Syn30M03M	49627	50557	47999	77
Syn30M04M	257063	251895	254651	281
Syn40M02M	153035	138493	147475	267
trimloss4	19159	25919	26705	16963
uflquad-15-60	2415	1809	2275	2103
uflquad-15-80	3577	3041	3413	3493
uflquad-20-40	4517	2679	3451	2763
uflquad-25-40	7001	4169	5171	6039
Arithmetic mean	762760.6	376940.9	332509.4	833065.8
Geometric mean	87019.1	41245.3	51591.5	15922.6

Table 16: Solution Time for FILMINT, BESTLIN, SSBC, and SBCGLP-SNC. Experiment is described in Section 4.10

Problem	FILMINT	BESTLIN	SSBC	SBCGLP-SNC
BatchS151208M	55.7	52.7	53.0	63.18
BatchS201210M	65.3	45.5	63.1	81.05
Batch_Storage10_BM_10_10_6_4	189.7	108.4	151.8	108.69
Batch_Storage_BM_10_10_6_4	208.5	109.3	177.2	102.68
CLay0305H	59.6	58.2	43.0	54.55
FLay05H	2501.6	2083.9	2506.3	3182.42
FLay05M	1161.1	981.5	1081.7	1062.47
fo7_2	151.4	344.5	169.5	138.2
fo7	454.1	391.4	692.0	1410.54
m7	1325.7	265.6	189.0	3.65
nd-12	2423.8	1604.4	969.5	651.49
nd-13	529.1	1412.3	881.7	1032.16
nd-14	4948.2	6723.5	9646.8	9269.54
o7_2	3714.9	5564.0	2762.5	8401.09
o7	8929.3	6184.1	6761.0	24504.69

Continued on next page

Table 16 – continued from previous page

Problem	FILMINT	BESTLIN	SSBC	SBCGLP-SNC
RSyn0810M02M	364.4	470.0	422.7	83.19
RSyn0810M03M	477.4	874.1	706.2	225.15
RSyn0810M04M	1008.4	1479.6	1506.6	337.2
RSyn0815M02M	1531.4	2192.2	1674.2	94.04
RSyn0820M02M	9170.8	10858.7	9520.4	242.81
RSyn0830M02M	4142.6	6789.1	5065.1	168.21
Safety3	18232.8	9328.3	5542.7	23666.13
safety_no_rotation_CH	122.0	14.4	25.5	36.9
SLay07H	47.4	9.1	16.0	18.59
SLay08H	181.5	32.4	52.5	69.1
SLay09H	3031.1	73.8	211.9	326.46
SLay09M	123.4	16.0	33.4	20.5
sssd-16-8-3	1517.9	2716.1	1530.6	811.78
sssd-17-7-3	854.2	775.9	224.8	203.98
sssd-18-7-3	430.1	373.3	298.5	150.35
sssd-20-8-3	19030.5	504.7	679.5	20816.12
Syn20M04M	220.6	410.3	296.2	46.47
Syn30M03M	204.9	402.6	267.3	34.67
Syn30M04M	1623.7	2204.6	1872.0	87.59
Syn40M02M	465.1	984.8	534.6	27.26
trimloss4	47.2	50.4	61.8	57.64
uflquad-15-60	607.2	474.2	601.7	754.38
uflquad-15-80	2017.3	1337.0	2002.4	2865.41
uflquad-20-40	831.9	498.6	663.3	671.22
uflquad-25-40	2146.9	967.0	1459.6	1939.24
Arithmetic mean	2378.7	1744.2	1535.4	2595.5
Geometric mean	712.6	495.5	490.9	294.3

Table 17: Extra gap closed for LIN, PSEUDO, SBC, GUB, MIXING, BESTLIN and SSBC.

Problem	LIN	PSEUDO	SBC	GUB	MIXING	BESTLIN	SSBC
BatchS151208M	0.0	0.0	7.0	15.3	18.7	0.0	7.0
BatchS201210M	0.0	0.0	8.2	18.7	22.9	0.0	8.2
Batch_Storage10_BM_10_10_6_4	3.7	3.1	15.5	18.6	18.1	3.7	15.5
Batch_Storage_BM_10_10_6_4	4.1	3.0	15.6	19.1	18.8	4.1	15.6
CLay0305H	0.0	0.0	10.3	39.4	39.3	0.0	10.3
FLay05H	0.0	0.0	9.9	39.6	37.6	0.0	9.9
FLay05M	0.0	0.0	9.9	39.6	34.4	0.0	9.9
fo7_2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
fo7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
m7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
nd-12	10.1	10.1	10.1	10.1	10.4	10.1	10.1
nd-13	19.2	19.2	19.2	19.2	19.2	19.2	19.2
nd-14	0.0	0.0	5.1	5.1	7.0	0.0	5.1
o7_2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
o7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RSyn0810M02M	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RSyn0810M03M	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RSyn0810M04M	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RSyn0815M02M	0.0	0.0	0.0	0.1	6.5	0.0	0.0
RSyn0820M02M	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RSyn0830M02M	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Safety3	0.0	0.0	5.0	15.1	15.1	0.0	5.0
safety_no_rotation_CH	0.0	0.0	10.1	22.0	20.0	0.0	10.1
SLay07H	0.0	0.0	38.8	69.6	69.6	0.0	38.8

Continued on next page

Table 17 – continued from previous page

Problem	LIN	PSEUDO	SBC	GUB	MIXING	BESTLIN	SSBC
SLay08H	0.0	0.0	26.2	67.3	67.3	0.0	26.2
SLay09H	0.0	0.0	23.6	62.3	62.3	0.0	23.6
SLay09M	0.0	0.0	23.6	62.3	62.3	0.0	23.6
sssd-16-8-3	0.0	0.0	3.5	3.5	3.5	0.0	3.5
sssd-17-7-3	0.0	0.0	4.6	4.6	4.6	0.0	4.6
sssd-18-7-3	0.0	0.0	4.3	4.3	4.3	0.0	4.3
sssd-20-8-3	0.0	0.0	4.1	4.1	4.1	0.0	4.1
Syn20M04M	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Syn30M03M	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Syn30M04M	0.1	0.0	0.0	0.0	0.0	0.1	0.0
Syn40M02M	0.0	0.0	0.0	0.3	5.2	0.0	0.0
trimloss4	0.0	0.0	6.3	10.5	30.4	0.0	9.6
uflquad-15-60	0.0	0.0	4.0	4.0	30.5	0.0	4.0
uflquad-15-80	0.0	0.0	3.7	3.7	27.0	0.0	3.7
uflquad-20-40	0.0	0.0	3.0	3.0	32.9	0.0	3.0
uflquad-25-40	0.0	0.0	2.7	2.7	34.8	0.0	2.7
Average	0.9	0.9	6.9	14.1	17.7	0.9	6.9

Table 18: Extra gap closed for SBCGLP-EN, SBCGLP-INF, SBCGLP-SNC and NOSB-CGLP-SNC.

Problem	SBCGLP-EN	SBCGLP-INF	SBCGLP-SNC	NOSB-CGLP-SNC
BatchS151208M	0.0	6.3	0.0	0.0
BatchS201210M	0.0	5.2	0.0	0.0
Batch_Storage10_BM_10_10_6_4	9.0	5.7	6.9	0.5
Batch_Storage_BM_10_10_6_4	11.1	3.0	8.3	3.6
CLay0305H	0.0	0.0	0.0	0.0
FLay05H	0.0	0.0	0.0	0.0
FLay05M	0.0	2.7	0.0	0.0
fo7_2	0.0	0.0	0.0	0.0
fo7	0.0	0.0	0.0	0.0
m7	0.0	0.0	0.0	0.0
nd-12	16.3	10.1	14.8	7.0
nd-13	19.2	19.5	19.2	12.5
nd-14	9.6	0.0	9.5	10.2
o7_2	0.0	0.0	0.0	0.0
o7	0.0	0.0	0.0	0.0
RSyn0810M02M	37.1	9.6	37.2	34.1
RSyn0810M03M	44.4	12.0	44.4	40.5
RSyn0810M04M	50.9	10.2	51.8	43.7
RSyn0815M02M	29.6	16.2	29.6	31.9
RSyn0820M02M	37.3	11.5	37.1	35.9
RSyn0830M02M	38.5	16.8	37.6	37.5
Safety3	0.0	5.0	0.0	0.0
safety_no_rotation_CH	0.0	5.2	0.0	0.0
SLay07H	0.0	6.7	0.0	0.0
SLay08H	0.0	4.3	0.0	0.0
SLay09H	0.0	3.9	0.0	0.0
SLay09M	0.0	15.4	0.0	0.0
sssd-16-8-3	12.2	52.4	8.7	8.5
sssd-17-7-3	4.9	47.6	3.6	3.7
sssd-18-7-3	4.2	70.2	3.2	3.2
sssd-20-8-3	9.1	65.1	6.7	6.7
Syn20M04M	79.6	25.9	61.1	78.7
Syn30M03M	82.6	37.2	82.3	86.4
Syn30M04M	88.6	37.7	86.2	85.9

Continued on next page

Table 18 – continued from previous page

Problem	SBCGLP-EN	SBCGLP-INF	SBCGLP-SNC	NOSB-CGLP-SNC
Syn40M02M	93.3	43.2	93.3	92.9
trimloss4	0.4	0.3	0.4	5.2
uflquad-15-60	1.4	2.7	1.6	0.0
uflquad-15-80	1.2	2.2	1.0	0.0
uflquad-20-40	1.3	1.9	1.2	0.0
uflquad-25-40	0.0	1.7	0.0	0.0
Average	17.0	13.9	16.1	15.7