Strong Branching Inequalities for Convex Mixed Integer Nonlinear Programs

Mustafa Kılınç

Department of Industrial Engineering
University of Pittsburgh, Pittsburgh, PA 15213
mrk46@pitt.edu

Jeff Linderoth, James Luedtke

Department of Industrial and Systems Engineering
University of Wisconsin-Madison, Madison, WI 53706
linderoth@wisc.edu, jrluedt1@wisc.edu

Andrew Miller

Institut de Mathématiques de Bordeaux
Université Bordeaux 1
RealOpt, INRIA Bordeaux Sud-Ouest
Andrew.Miller@math.u-bordeaux1.fr

September 3, 2011

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.

1 Introduction

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

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

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 $B \subseteq I$ is the 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 [16] or the LP/NLP branch-and-bound algorithm [30], the lower bound comes from solving a linear program (LP), often called *the master problem* that is based on a polyhedral outer-approximation of $\mathcal{P}$:

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

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, 20].

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 [27, 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

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

and

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

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, 27].

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. Leyffer [26] also has 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. 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 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 Lemma 1 can be found in the Ph.D. thesis of Kılınç [24].

Lemma 1 *Let* $i \in I$ *and* $k \in \mathbb{Z}$*. 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 [29] and were systematically studied by Günlük and Pochet [22]. 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 3 is a straightforward application of the *mixing inequalities* of [22] 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 3 ([5, 22]) *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 [22].

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 [24]. 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 outerapproximate 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 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* $\lambda^0, \lambda^1 \in \mathbb{R}_+^{|J|}, \mu^0, \mu^1 \in \mathbb{R}_+^m, \theta^0, \theta^1 \in \mathbb{R}_+$ *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 4 *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

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

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 [24].

$\Diamond$

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)*

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

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 [24].

$\Diamond$

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)[18]:

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

The SNC normalization is criticized by Fischetti, Lodi, and Tramontani [18] for its dependence on the relative scaling of the constraints. To overcome this drawback, they proposed the *Euclidean Normalization*

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

instead of (SNC). We refer reader to [18] 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

$$\begin{aligned} \alpha_k^0 &= \sigma c_k^0 + \lambda^0 D_k^0 + \mu^0 A_k, \\ \alpha_k^1 &= \sigma c_k^1 + \lambda^1 D_k^1 + \mu^1 A_k, \\ \hat{m}_k &= \frac{\alpha_k^0 - \alpha_k^1}{\theta^0 + \theta^1}, \end{aligned}$$

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_k, A_k represents the kth 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 of loss 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

$$\gamma_j = \min\{\sigma c_j^0 + \lambda^0 D_j^0 + \mu^0 A_j, \sigma c_j^1 + \lambda^1 D_j^1 + \mu^1 A_j\}.$$

Thus, the inequality

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

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 [30], which uses the outerapproximation master problem (4). In our experiments, all strong branching inequalities are added directly to (4). FilMINT uses MINTO [28] to enforce integrality of the master problem via branching and filterSQP [19] 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 [27].

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 significant 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 [25], the GAMS MINLP World [13], the collection on the website of the IBM-CMU research group [33],

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 [31, 35], layout design problems [32, 14], synthesis design problems [16, 34], retrofit planning [32], stochastic service system design problems [17], cutting stock problems [23], uncapacitated facility location problems [21] and network design problems [12]. Characteristics of the instances are given in Table 1, 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 have been run on a cluster of identical 64-bit Intel Core2 Duo microprocessors clocked at 3.00 GHz, each with 2 GB RAM. In order to concisely display the relative performance of different solution techniques, we make use of performance profiles (see [15]). 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

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

where z_{CUTS} is the value of LP relaxation after adding inequalities, $z_{\mathrm{MP}(\mathcal{K})}$ 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. The instances for which at least one solver failed to terminate because of a memory limite are taken out of consideration when reporting aggregate numbers.

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 subproblem (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

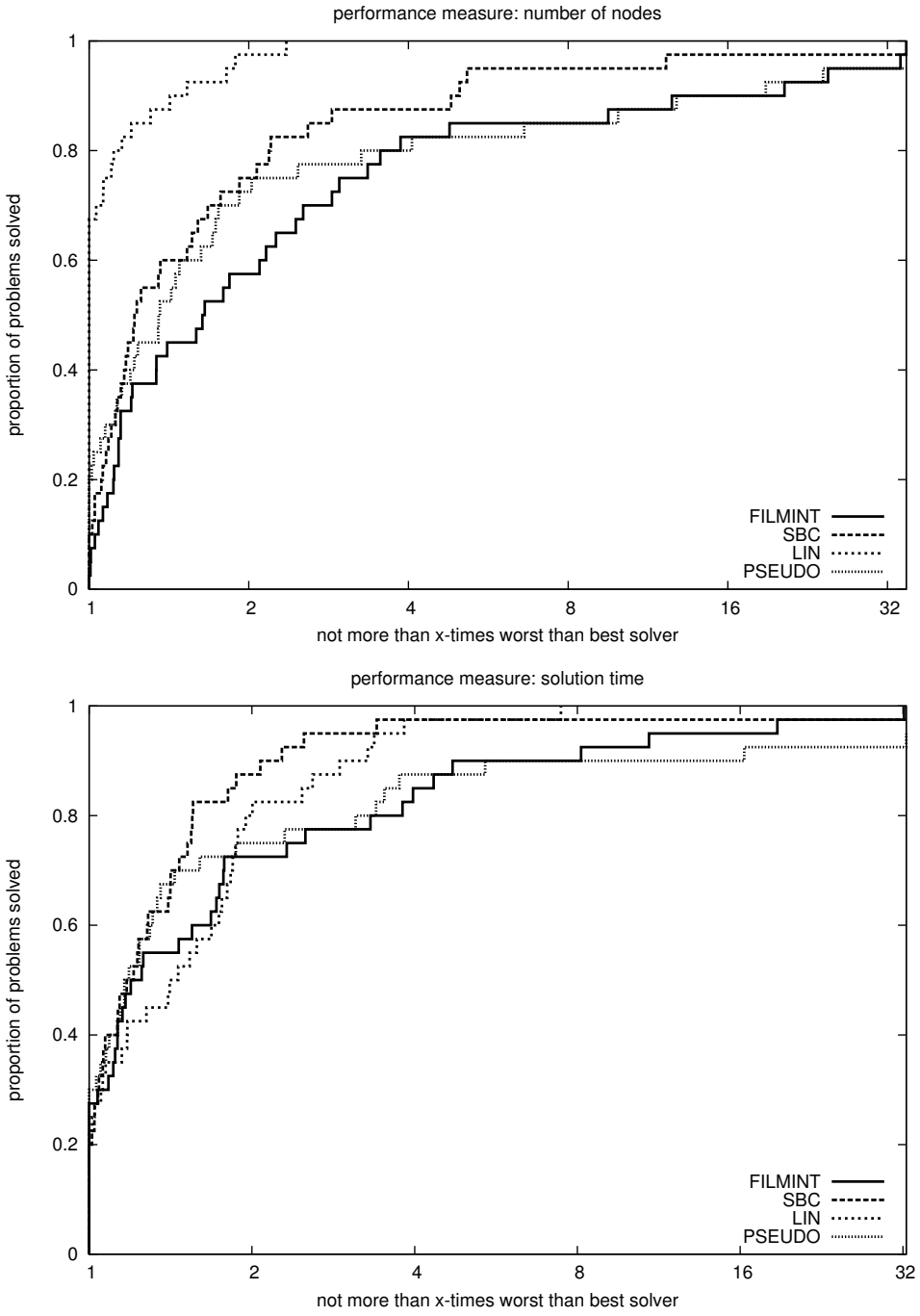
- `FILMINT`: The default version of FilMINT.
- `LIN`: FilMINT, but with the master problem augmented with linearizations from NLP-based branching subproblems, as described in Section 2.1.
- `SBC`: FilMINT, but with the master problem augmented with the simple strong branching cuts (5).
- `PSEUDO`: FilMINT, 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. In PSEUDO, as in the versions that add strong branching inequalities, we perform NLP-based strong branching at most once for each variable. However, rather than adding any strong branching inequalities, we instead simply initialize FiLMINT's pseudocosts based on the optimal values of (NLP_i^0) and (NLP_i^1). These pseudocosts are then updated based on FiLMINT's default update strategy. Thus, although the initial pseudocost information is based directly on the nonlinear programming relaxation, 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, and all methods are at least as good as `FILMINT`.

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

Figure 1: Performance Profile of Elementary NLP-based Strong Branching Inequalities



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.

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.

In Table 2, 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 30 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 19.20% of the gap at root node, and adding only SBC inequalities closed 9.66%. It is then somewhat surprising that the number of nodes required for the two methods is approximately equal. One explanation of this phenomenon is that FilMINT chooses worse 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 our 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 3, 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 19.06% of the optimality gap at root node, and SBC closed only 7.77% of the 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, 92 seconds and 13,651 nodes are required to prove optimality when using MIXSBC, compared to only 29 seconds and 6,035 nodes for SBC alone.

4.6 Linearization Strategies

In our computational experiments we were not able to significantly improve 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 4, 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 for 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 1,568.3 seconds to 1,078.9 seconds.

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 5. 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.

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 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 (SSBC) (16).
- `SBCGLP-SNC`: FilMINT, adding disjunctive inequalities based on solving the CGLP (17) using the standard normalization condition (SNC).

Figure 2: Performance Profile of CGLP with Different Normalization Constraints

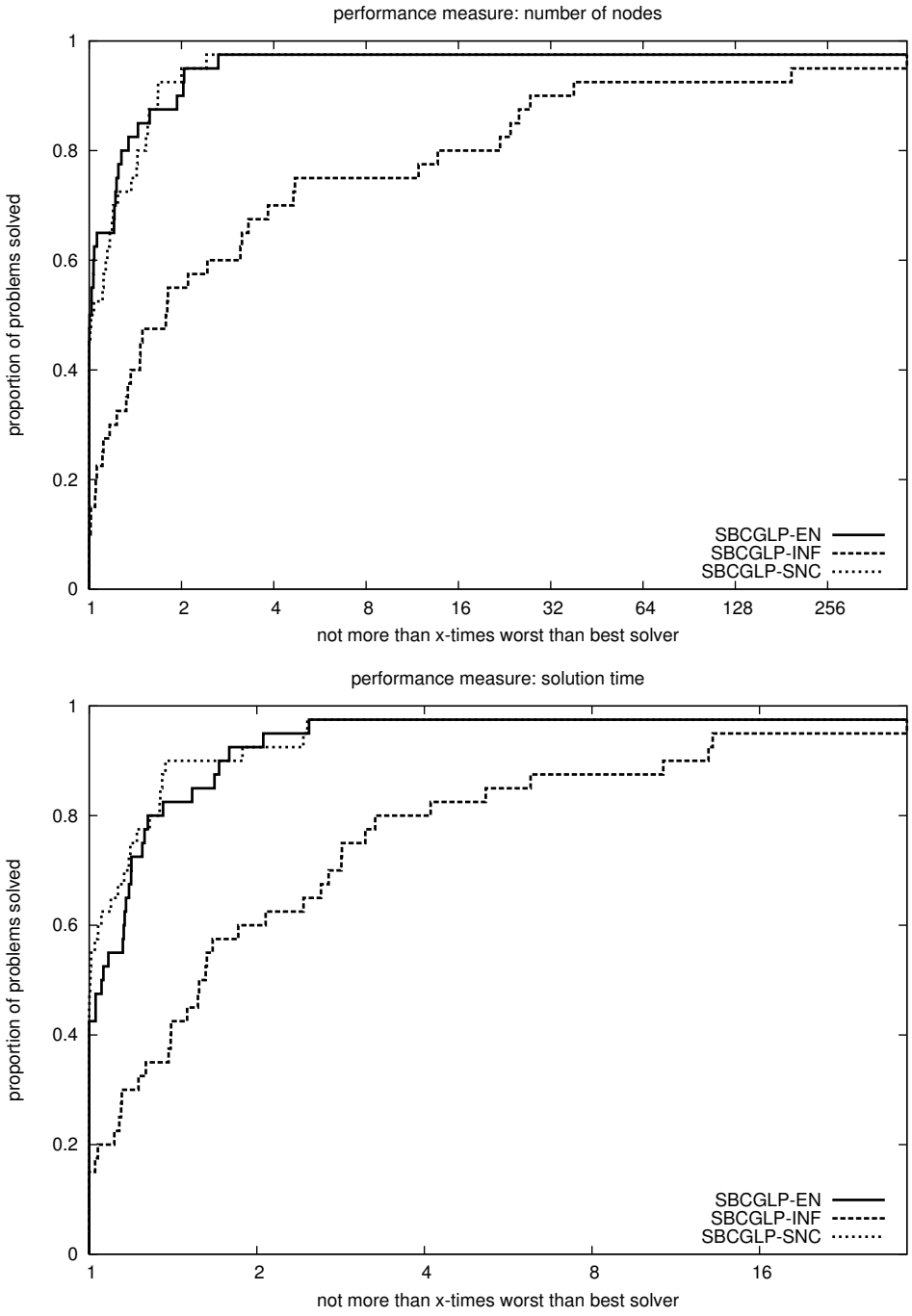
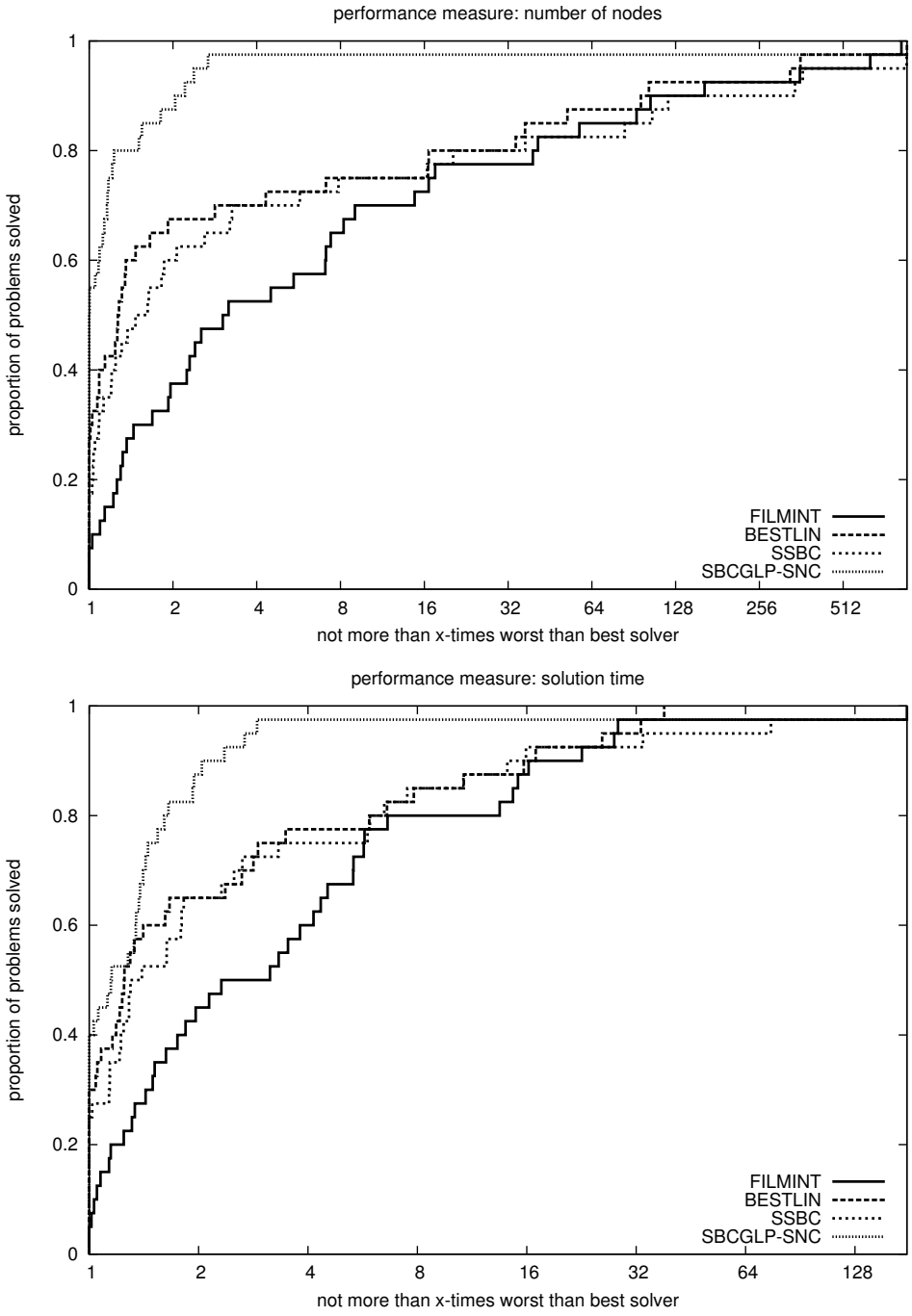


Figure 3: Performance Profile of Best Methods and FILMINT



The monoidal strengthening step described in Section 3.4 was applied to the inequalities generated by methods `SSBC` and `SBCGLP-SNC`. In Table 6, we list the average number of nodes and solution time for 38 instances in our test set. (The instances *RSyn0820M02M* and *Safety3* are taken out of consideration when reporting aggregate numbers in Table 6 since SSBC and SBCGLP-SNC failed to terminate for these instances due to a memory limit). The performance profiles in Figure 3 show that 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. Additionally, both linearizations and SSBC inequalities improve the performance of default FilMINT substantially. The detailed performance of methods on each instance is given in Table 14 and Table 15.

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 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.

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. Bonami, M. Kılınç, and J. Linderoth. Algorithms and software for solving convex mixed integer nonlinear programs. In *IMA Volumes on MINLP*, 2011. to appear.

[12] R R Boorstyn and H Frank. Large-scale network topological optimization. *IEEE Trans. Commun*, pages 29–47, 1997.

[13] 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.

[14] 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.

[15] E. Dolan and J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[16] M. A. Duran and I. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[17] S. Elhedhli. Service System Design with Immobile Servers, Stochastic Demand, and Congestion. *Manufacturing & Service Operations Management*, 8(1):92–97, 2006.

[18] M. Fischetti, A. Lodi, and A. Tramontani. On the separation of disjunctive cuts. *Mathematical Programming*, pages 1–26, 2009.

[19] R. Fletcher and S. Leyffer. User manual for filterSQP, 1998. University of Dundee Numerical Analysis Report NA-181.

[20] I. E. Grossmann. Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, 3:227–252, 2002.

[21] 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.

[22] O. Günlük and Y. Pochet. Mixing mixed-integer inequalities. *Mathematical Programming*, 90(3):429 – 457, 2001.

[23] 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.

[24] M. Kılınç. *Disjunctive Cutting Planes and Algorithms for Convex Mixed Integer Nonlinear Programming,*. PhD thesis, University of Wisconsin-Madison, 2011.

[25] S. Leyffer. MacMINLP: Test problems for mixed integer nonlinear programming, 2003. `http://www.mcs.anl.gov/~leyffer/macminlp`.

[26] S. Leyffer. Experiments with MINLP branching techniques. Preprint ANL/MCS-P1734-0310, Argonne National Laboratory, Mathematics and Computer Science Division, March 2010.

[27] 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.

[28] G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi. MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.

[29] Y. Pochet and L. Wolsey. Lot sizing with constant batches: Formulation and valid inequalities. *Mathematics of Operations Research*, 18:767–785, 1993.

[30] 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.

[31] 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.

[32] N. Sawaya. *Reformulations, relaxations and cutting planes for generalized disjunctive programming*. PhD thesis, Chemical Engineering Department, Carnegie Mellon University, 2006.

[33] N. W. Sawaya, C. D. Laird, and P. Bonami. A novel library of non-linear mixed-integer and generalized disjunctive programming problems. In preparation, 2006.

[34] 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.

[35] 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

For Tables 7-15, if the method could not provide the optimal solution because the solver hit a memory limit we state the reason with the letter m.

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

Problem	`FILMINT`	`SBC`	`LIN`	`PSEUDO`
BatchS151208M	5391	3071	1395	1609
Continued on next page				

Table 7 – continued from previous page

Problem	FILMINT	SBC	LIN	PSEUDO
BatchS201210M	3447	2909	1405	1911
Batch_Storage10_BM_10_10_6_4	15917	10351	7597	11059
Batch_Storage_BM_10_10_6_4	21849	11817	9707	15781
CLay0305H	26449	11017	15979	10445
FLay05H	102207	99691	102791	101697
FLay05M	91579	83289	82173	83129
fo7_2	109811	105729	78265	111777
fo7	276547	331499	293427	206665
m7	621027	320671	395459	209605
nd-12	20873	12179	17125	7275
nd-13	24391	8023	7587	6891
nd-14	101813	105107	95321	89597
o7_2	2028675	1567755	1417959	1274499
o7	2675019	2741915	3213077	4634451
RSyn0810M02M	117015	114369	102065	126109
RSyn0810M03M	88593	93341	87963	98585
RSyn0810M04M	156835	136943	87607	92047
RSyn0815M02M	493283	486495	457321	430125
RSyn0820M02M	2157455	m	2032033	m
RSyn0830M02M	825939	903237	763039	1129401
Safety3	3087205	1431767	1869475	1936385
safety_no_rotation_CH	18327	5525	1925	4769
SLay07H	4505	635	359	689
SLay08H	15203	1921	743	1505
SLay09H	82525	12257	2377	15709
SLay09M	16689	6035	493	4899
sssd-16-8-3	2092313	2793721	1279401	m
sssd-17-7-3	1287961	1297181	269459	3452263
sssd-18-7-3	265695	396017	79281	1918463
sssd-20-8-3	3573363	144505	262273	2725971
Syn20M04M	90393	89875	103651	153573
Syn30M03M	49731	50247	43761	177719
Syn30M04M	237847	245963	228451	744635
Syn40M02M	145331	85947	79033	89121
trimloss4	25535	40863	21279	25885
uflquad-15-60	2523	2303	2091	2501
uflquad-15-80	3689	3401	3319	3559
uflquad-20-40	3591	3359	2681	3619
uflquad-25-40	6973	5115	4221	7399

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

Problem	FILMINT	SBC	LIN	PSEUDO
BatchS151208M	52.0	32.0	30.6	29.9
BatchS201210M	43.3	53.3	43.9	34.4
Batch_Storage10_BM_10_10_6_4	112.1	92.6	63.0	81.7
Batch_Storage_BM_10_10_6_4	150.8	99.9	85.1	111.6
CLay0305H	108.7	28.6	96.4	32.3
FLay05H	1702.0	1486.6	1995.6	1425.5
FLay05M	696.6	695.9	726.3	617.1
fo7_2	177.0	163.0	174.9	189.4
fo7	355.6	442.4	535.6	284.2
m7	796.8	336.8	787.8	240.4
nd-12	1499.0	681.8	2814.8	377.4
nd-13	497.8	509.1	1233.2	542.4
nd-14	6222.7	7502.8	10516.8	3618.7
o7_2	3384.4	2580.0	3548.3	2015.1
o7	3797.3	4292.8	6936.7	8728.7
RSyn0810M02M	272.7	279.0	431.5	281.0
RSyn0810M03M	326.5	461.8	654.4	436.3
RSyn0810M04M	746.3	821.7	900.7	639.2
RSyn0815M02M	1126.5	1136.7	1944.3	998.1
RSyn0820M02M	5770.4	m	8433.4	m
RSyn0830M02M	3017.0	3204.5	5623.0	3738.6
Safety3	9063.5	3607.8	6271.4	5779.5
safety_no_rotation_CH	63.4	20.5	13.5	18.3
SLay07H	38.4	8.9	9.4	9.3
SLay08H	197.0	29.8	24.3	28.7
SLay09H	1458.6	194.2	77.8	264.1
SLay09M	93.1	29.2	8.6	32.2
sssd-16-8-3	1106.5	1563.5	2039.2	m
sssd-17-7-3	614.7	602.1	265.0	1431.0
sssd-18-7-3	110.5	185.1	98.9	1609.8
sssd-20-8-3	3317.5	103.2	394.9	3348.6
Syn20M04M	153.1	188.9	396.5	286.0
Syn30M03M	147.0	177.7	264.6	516.7
Syn30M04M	1127.7	1141.5	1730.2	3507.4
Syn40M02M	317.8	213.1	345.3	205.1
trimloss4	36.9	57.0	38.8	42.2
uflquad-15-60	437.4	379.5	446.0	470.0
uflquad-15-80	1220.9	1177.7	1354.1	1265.2

Continued on next page

Table 8 – continued from previous page

Problem	FILMINT	SBC	LIN	PSEUDO
uflquad-20-40	463.4	443.3	418.1	485.8
uflquad-25-40	1389.4	948.9	960.5	1365.7

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

Problem	SBC		GUBSBC	
	Node	**Time**	**Node**	**Time**
BatchS151208M	3071	32.0	2869	43.0
BatchS201210M	2909	53.3	2591	50.9
Batch_Storage10_BM_10_10_6_4	10351	92.6	8497	66.6
Batch_Storage_BM_10_10_6_4	11817	99.9	12643	91.5
CLay0305H	11017	28.6	19173	44.1
FLay05H	99691	1486.6	114785	1791.7
FLay05M	83289	695.9	85747	757.0
nd-12	12179	681.8	11675	700.8
nd-13	8023	509.1	7683	736.8
nd-14	105107	7502.8	105997	4642.1
RSyn0810M02M	114369	279.0	116249	312.0
RSyn0810M03M	93341	461.8	95047	632.4
RSyn0810M04M	136943	821.7	91385	826.4
RSyn0815M02M	486495	1136.7	441117	1162.7
RSyn0820M02M	m	m	1983993	5824.7
RSyn0830M02M	903237	3204.5	1013379	3507.7
Safety3	1431767	3607.8	1621317	4299.8
safety_no_rotation_CH	5525	20.5	5515	20.0
SLay07H	635	8.9	1167	11.0
SLay08H	1921	29.8	3649	45.1
SLay09H	12257	194.2	16239	289.7
SLay09M	6035	29.2	7409	50.0
sssd-16-8-3	2793721	1563.5	2134397	1175.7
sssd-17-7-3	1297181	602.1	318079	158.4
sssd-18-7-3	396017	185.1	358449	178.0
sssd-20-8-3	144505	103.2	696655	375.0
Syn20M04M	89875	188.9	89227	261.3
Syn30M03M	50247	177.7	51149	226.9
Syn30M04M	245963	1141.5	235895	1309.7
Syn40M02M	85947	213.1	141867	396.9

Continued on next page

Table 9 – continued from previous page

Problem	SBC		GUBSBC	
	Node	**Time**	**Node**	**Time**
trimloss4	40863	57.0	29579	48.7

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

Problem	SBC		MIXSBC	
	Node	**Time**	**Node**	**Time**
BatchS151208M	3071	32.0	3243	74.0
BatchS201210M	2909	53.3	3091	71.5
Batch_Storage10_BM_10_10_6_4	10351	92.6	13375	108.5
Batch_Storage_BM_10_10_6_4	11817	99.9	10833	90.7
CLay0305H	11017	28.6	15937	54.7
FLay05H	99691	1486.6	104233	1569.8
FLay05M	83289	695.9	82225	657.5
fo7_2	105729	163.0	105729	163.0
fo7	331499	442.4	331499	442.0
m7	320671	336.8	320671	336.6
nd-12	12179	681.8	9715	606.5
nd-13	8023	509.1	7403	541.6
nd-14	105107	7502.8	94813	6208.2
o7_2	1567755	2580.0	1567755	2586.1
o7	2741915	4292.8	2741915	4297.7
RSyn0810M02M	114369	279.0	115679	329.8
RSyn0810M03M	93341	461.8	93097	474.6
RSyn0810M04M	136943	821.7	91211	639.1
RSyn0815M02M	486495	1136.7	464247	1197.6
RSyn0820M02M	m	m	2001271	5427.3
RSyn0830M02M	903237	3204.5	903237	3200.8
Safety3	1431767	3607.8	3267119	10082.1
safety_no_rotation_CH	5525	20.5	5227	21.3
SLay07H	635	8.9	1351	13.3
SLay08H	1921	29.8	2559	42.5
SLay09H	12257	194.2	10699	174.2
SLay09M	6035	29.2	13651	92.4
sssd-16-8-3	2793721	1563.5	3966469	1835.8
sssd-17-7-3	1297181	602.1	846487	381.1
sssd-18-7-3	396017	185.1	389081	181.8

Continued on next page

Table 10 – continued from previous page

Problem	SBC		MIXSBC	
	Node	**Time**	**Node**	**Time**
sssd-20-8-3	144505	103.2	1461383	761.0
Syn20M04M	89875	188.9	90757	184.3
Syn30M03M	50247	177.7	50247	177.5
Syn30M04M	245963	1141.5	245963	1139.9
Syn40M02M	85947	213.1	83179	223.6
trimloss4	40863	57.0	27273	56.3
uflquad-15-60	2303	379.5	2431	475.8
uflquad-15-80	3401	1177.7	3595	1428.9
uflquad-20-40	3359	443.3	4063	624.9
uflquad-25-40	5115	948.9	6697	1651.8

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

Problem	LIN		BESTLIN	
	Node	**Time**	**Node**	**Time**
BatchS151208M	1395	30.6	2347	37.91
BatchS201210M	1405	43.87	1791	41.99
Batch_Storage10_BM_10_10_6_4	7597	63.04	7101	57.09
Batch_Storage_BM_10_10_6_4	9707	85.12	8643	65.3
CLay0305H	15979	96.38	21131	83.32
FLay05H	102791	1995.57	106617	1600.88
FLay05M	82173	726.34	87447	647.02
fo7_2	78265	174.92	85429	151.63
fo7	293427	535.6	300365	370.69
m7	395459	787.84	130141	171.17
nd-12	17125	2814.81	9673	771.29
nd-13	7587	1233.23	8283	691.18
nd-14	95321	10516.79	115987	6062.51
o7_2	1417959	3548.33	1564357	3438.74
o7	3213077	6936.68	3376079	4737.24
RSyn0810M02M	102065	431.46	116957	301.3
RSyn0810M03M	87963	654.44	85217	466.33
RSyn0810M04M	87607	900.72	63177	510.04
RSyn0815M02M	457321	1944.27	462711	1206.16
RSyn0820M02M	2032033	8433.37	1954165	5218.89
RSyn0830M02M	763039	5622.96	856223	3572.13

Continued on next page

Table 11 – continued from previous page

Problem	LIN		BESTLIN	
	Node	**Time**	**Node**	**Time**
Safety3	1869475	6271.37	2604659	7053.02
safety_no_rotation_CH	1925	13.48	4573	17.98
SLay07H	359	9.38	813	11.06
SLay08H	743	24.25	2349	36.69
SLay09H	2377	77.8	6345	125.43
SLay09M	493	8.58	2295	16.35
sssd-16-8-3	1279401	2039.23	1317759	789.18
sssd-17-7-3	269459	264.96	787457	372.9
sssd-18-7-3	79281	98.92	201431	105.19
sssd-20-8-3	262273	394.88	242025	146.23
Syn20M04M	103651	396.46	89153	264.4
Syn30M03M	43761	264.62	45875	182
Syn30M04M	228451	1730.23	249021	1261.79
Syn40M02M	79033	345.33	81585	210.75
trimloss4	21279	38.83	23583	34.36
uflquad-15-60	2091	445.99	1851	292.52
uflquad-15-80	3319	1354.05	3019	914.67
uflquad-20-40	2681	418.12	2853	324.03
uflquad-25-40	4221	960.54	4831	794.65

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

Problem	SBC		SSBC	
	Node	**Time**	**Node**	**Time**
BatchS151208M	3071	31.97	3071	31.95
BatchS201210M	2909	53.33	3259	51.32
Batch_Storage10_BM_10_10_6_4	10351	92.55	14651	131.98
Batch_Storage_BM_10_10_6_4	11817	99.87	11879	106.69
CLay0305H	11017	28.62	11017	28.6
FLay05H	99691	1486.59	99691	1484.55
FLay05M	83289	695.89	83289	696.52
fo7_2	105729	163.01	105729	163.26
fo7	331499	442.44	331499	442.27
m7	320671	336.79	320671	336.7
nd-12	12179	681.76	6587	476.7
nd-13	8023	509.07	8471	490.8

Continued on next page

Table 12 – continued from previous page

Problem	SBC		SSBC	
	Node	**Time**	**Node**	**Time**
nd-14	105107	7502.78	110687	4997.81
o7_2	1567755	2579.99	1567755	2581.51
o7	2741915	4292.82	2741915	4327.49
RSyn0810M02M	114369	279.01	115459	302.48
RSyn0810M03M	93341	461.84	94485	467.8
RSyn0810M04M	136943	821.72	77739	597.46
RSyn0815M02M	486495	1136.73	463365	1086.75
RSyn0820M02M	m	m	m	m
RSyn0830M02M	903237	3204.45	1071975	3619.43
Safety3	1431767	3607.77	1576035	4239.98
safety_no_rotation_CH	5525	20.5	5525	20.47
SLay07H	635	8.85	635	8.85
SLay08H	1921	29.78	1921	29.77
SLay09H	12257	194.23	12257	194.42
SLay09M	6035	29.21	6035	29.22
sssd-16-8-3	2793721	1563.47	1491441	834.42
sssd-17-7-3	1297181	602.06	1043811	805.61
sssd-18-7-3	396017	185.12	374615	191.52
sssd-20-8-3	144505	103.19	354987	238.97
Syn20M04M	89875	188.87	91683	197.25
Syn30M03M	50247	177.72	47779	178.85
Syn30M04M	245963	1141.51	248575	1186.51
Syn40M02M	85947	213.07	83171	210.24
trimloss4	40863	57.03	23353	35.02
uflquad-15-60	2303	379.45	2303	379.84
uflquad-15-80	3401	1177.67	3401	1177.76
uflquad-20-40	3359	443.29	3435	417.04
uflquad-25-40	5115	948.88	5233	963.24

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	4101	67.31	2843	43.77	3499	65.92
BatchS201210M	3113	70.78	3649	81.59	2555	80.17

Continued on next page

Table 13 – continued from previous page

Problem	SBCGLP-EN		SBCGLP-SNC		SBCGLP-INF	
	Node	**Time**	**Node**	**Time**	**Node**	**Time**
BatchS10BM101064	10601	99.1	8323	72.97	14965	118.21
BatchSBM101064	8943	93.01	10615	93.32	16149	151.33
CLay0305H	14595	56.67	17037	67.33	21429	64.85
FLay05H	106611	1767.95	104777	1667.28	106173	1902.9
FLay05M	83085	781.69	80525	614.33	84149	680.5
fo7_2	81119	139.27	65135	116.93	88959	148.01
fo7	240737	467.54	277537	515.96	231859	375.35
m7	2461	3.67	3817	4.48	11401	11.5
nd-12	10221	662.61	9937	658.01	6487	266.88
nd-13	8561	649.7	8079	566.05	8491	563.07
nd-14	97085	6645.54	109993	7035.41	107957	5285.1
o7_2	1412825	2643.59	2826683	4971.34	1428113	2709.14
o7	5710371	10136.23	6357337	10162.08	m	m
RSyn0810M02M	7173	51.15	7051	51.1	83641	317.42
RSyn0810M03M	12047	176.39	12007	177.23	56303	500.6
RSyn0810M04M	2443	173.21	3827	180.11	67095	893.06
RSyn0815M02M	15143	88.17	12565	76.86	274925	1011.29
RSyn0820M02M	33773	185.04	37359	202.34	1285209	5446.63
RSyn0830M02M	10773	126.15	8909	108.35	210943	1411.48
Safety3	m	m	m	m	2025337	5561.19
safety_no_rotation_CH	2461	14.11	3375	19.04	8139	34.24
SLay07H	495	8.98	711	11.9	1559	16.64
SLay08H	1617	29.2	1853	34.15	1787	35.81
SLay09H	9121	185.26	4717	108.22	8387	170.49
SLay09M	3777	28.4	1851	16.83	5761	35.08
sssd-16-8-3	1228207	686.96	465701	333.11	1131861	557.1
sssd-17-7-3	370313	192.37	182611	107.42	699239	306.61
sssd-18-7-3	546293	251.77	444957	214.8	594687	338.96
sssd-20-8-3	268233	175.43	647019	423.73	563567	457.56
Syn20M04M	1167	35.77	869	33.81	11903	110.92
Syn30M03M	83	27.24	139	27.59	16197	111.86
Syn30M04M	171	66.76	287	74.52	79373	717.48
Syn40M02M	219	20.38	227	19.64	5523	53.42
trimloss4	21925	41.8	27185	53.02	32695	58.56
uflquad-15-60	2131	483.34	2147	470.82	2485	487.71
uflquad-15-80	3375	1708.49	3421	1510.98	3575	1435.12
uflquad-20-40	2957	471.5	2861	436.28	4187	611.01
uflquad-25-40	5267	1242.44	5263	1054.66	6951	1463.38

Table 14: Number of Nodes for FILMINT, BESTLIN, SSBC, and SBCGLP-SNC. Experiment is described in Section 4.9

Problem	FILMINT	BESTLIN	SSBC	SBCGLP-SNC
BatchS151208M	5391	2347	3071	2843
BatchS201210M	3447	1791	3259	3649
Batch_Storage10_BM_10_10_6_4	15917	7101	14651	8323
Batch_Storage_BM_10_10_6_4	21849	8643	11879	10615
CLay0305H	26449	21131	11017	17037
FLay05H	102207	106617	99691	104777
FLay05M	91579	87447	83289	80525
fo7_2	109811	85429	105729	65135
fo7	276547	300365	331499	277537
m7	621027	130141	320671	3817
nd-12	20873	9673	6587	9937
nd-13	24391	8283	8471	8079
nd-14	101813	115987	110687	109993
o7_2	2028675	1564357	1567755	2826683
o7	2675019	3376079	2741915	6357337
RSyn0810M02M	117015	116957	115459	7051
RSyn0810M03M	88593	85217	94485	12007
RSyn0810M04M	156835	63177	77739	3827
RSyn0815M02M	493283	462711	463365	12565
RSyn0820M02M	2157455	1954165	m	37359
RSyn0830M02M	825939	856223	1071975	8909
Safety3	3087205	2604659	1576035	m
safety_no_rotation_CH	18327	4573	5525	3375
SLay07H	4505	813	635	711
SLay08H	15203	2349	1921	1853
SLay09H	82525	6345	12257	4717
SLay09M	16689	2295	6035	1851
sssd-16-8-3	2092313	1317759	1491441	465701
sssd-17-7-3	1287961	787457	1043811	182611
sssd-18-7-3	265695	201431	374615	444957
sssd-20-8-3	3573363	242025	354987	647019
Syn20M04M	90393	89153	91683	869
Syn30M03M	49731	45875	47779	139
Syn30M04M	237847	249021	248575	287
Syn40M02M	145331	81585	83171	227
trimloss4	25535	23583	23353	27185
uflquad-15-60	2523	1851	2303	2147
uflquad-15-80	3689	3019	3401	3421

Continued on next page

Table 14 – continued from previous page

Problem	FILMINT	BESTLIN	SSBC	SBCGLP-SNC
uflquad-20-40	3591	2853	3435	2861
uflquad-25-40	6973	4831	5233	5263

Table 15: Solution Time for FILMINT, BESTLIN, SSBC, and SBCGLP-SNC. Experiment is described in Section 4.9

Problem	FILMINT	BESTLIN	SSBC	SBCGLP-SNC
BatchS151208M	52.0	37.9	32.0	43.8
BatchS201210M	43.3	42.0	51.3	81.6
Batch_Storage10_BM_10_10_6_4	112.1	57.1	132.0	73.0
Batch_Storage_BM_10_10_6_4	150.8	65.3	106.7	93.3
CLay0305H	108.7	83.3	28.6	67.3
FLay05H	1702.0	1600.9	1484.6	1667.3
FLay05M	696.6	647.0	696.5	614.3
fo7_2	177.0	151.6	163.3	116.9
fo7	355.6	370.7	442.3	516.0
m7	796.8	171.2	336.7	4.5
nd-12	1499.0	771.3	476.7	658.0
nd-13	497.8	691.2	490.8	566.1
nd-14	6222.7	6062.5	4997.8	7035.4
o7_2	3384.4	3438.7	2581.5	4971.3
o7	3797.3	4737.2	4327.5	10162.1
RSyn0810M02M	272.7	301.3	302.5	51.1
RSyn0810M03M	326.5	466.3	467.8	177.2
RSyn0810M04M	746.3	510.0	597.5	180.1
RSyn0815M02M	1126.5	1206.2	1086.8	76.9
RSyn0820M02M	5770.4	5218.9	m	202.3
RSyn0830M02M	3017.0	3572.1	3619.4	108.4
Safety3	9063.5	7053.0	4240.0	m
safety_no_rotation_CH	63.4	18.0	20.5	19.0
SLay07H	38.4	11.1	8.9	11.9
SLay08H	197.0	36.7	29.8	34.2
SLay09H	1458.6	125.4	194.4	108.2
SLay09M	93.1	16.4	29.2	16.8
sssd-16-8-3	1106.5	789.2	834.4	333.1
sssd-17-7-3	614.7	372.9	805.6	107.4
sssd-18-7-3	110.5	105.2	191.5	214.8
sssd-20-8-3	3317.5	146.2	239.0	423.7

Continued on next page

Table 15 – continued from previous page

Problem	FILMINT	BESTLIN	SSBC	SBCGLP-SNC
Syn20M04M	153.1	264.4	197.3	33.8
Syn30M03M	147.0	182.0	178.9	27.6
Syn30M04M	1127.7	1261.8	1186.5	74.5
Syn40M02M	317.8	210.8	210.2	19.6
trimloss4	36.9	34.4	35.0	53.0
uflquad-15-60	437.4	292.5	379.8	470.8
uflquad-15-80	1220.9	914.7	1177.8	1511.0
uflquad-20-40	463.4	324.0	417.0	436.3
uflquad-25-40	1389.4	794.7	963.2	1054.7

Table 1: 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 2: Solution Statistics Comparing SBC versus GUBSBC inequalities

	Extra gap closed (%)	**Average # of nodes**		**Average time**	
		Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
SBC	9.66	289476.6	48531.6	840.3	244.3
GUBSBC	19.20	261314.3	51506.5	807.1	274.7

Table 3: Solution Statistics Comparing Mixing SBC versus SBC

	Extra gap closed (%)	**Average # of nodes**		**Average time**	
		Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
SBC	7.77	352,975.5	50,396.9	922.4	313.3
MIXSBC	19.06	450,208.7	56,829.5	1107.7	379.4

Table 4: Solution Statistics Comparing LIN versus BESTLIN

	Average # of nodes		**Average time**	
	Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
LIN	338,089.2	39,330.3	1,568.3	407.3
BESTLIN	375,883.0	47,346.7	1,078.9	312.3

Table 5: Solution Statistics Comparing Strengthened SBC versus SBC

	Extra gap closed (%)	**Average # of nodes**		**Average time**	
		Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
SBC	7.77	352,975.5	50,396.9	922.4	313.3
SSBC	7.86	323,574.1	49,200.5	865.7	310.7

Table 6: Solution Statistics Comparing Best Methods and FILMINT

	Average # of nodes		**Average time**	
	Arithmetic	**Geometric**	**Arithmetic**	**Geometric**
FILMINT	413,917.2	71,059.5	983.6	485.8
BESTLIN	275,697.2	38,634.1	812.7	267.1
SSBC	290,614.6	44,910.6	776.9	290.1
SBCGLP-SNC	308,574.7	12,797.4	847.8	161.9