Comput Optim Appl (2014) 59:639–665
DOI 10.1007/s10589-014-9690-8

Strong-branching inequalities for convex mixed integer nonlinear programs

Mustafa Kılınç · Jeff Linderoth · James Luedtke · Andrew Miller

Received: 19 September 2013 / Published online: 7 October 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.

Electronic supplementary material The online version of this article (doi:10.1007/s10589-014-9690-8) contains supplementary material, which is available to authorized users.

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

J. Linderoth (✉) · J. Luedtke
Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI 53706, USA
e-mail: linderoth@wisc.edu

J. Luedtke
e-mail: jrluedt1@wisc.edu

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

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

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

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 [3,28]. 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 [2,28].

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 Sect. 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 [35,36]. 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 Sect. 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 Sect. 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 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 SBC*

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

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 et al. [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 Sect. 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 SBC (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

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

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:

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

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*

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

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*

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

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,*

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

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 $\hat{\lambda}^h, \hat{\mu}^h, \hat{\phi}^h \ge 0$ and a Lagrange multiplier $\hat{\theta}^h \ge 0$, 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:

$$\sum_{i=1}^{n} |\alpha_i| + \sigma = 1. \qquad (\alpha\text{NORM})$$

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

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

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

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

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 Sect. 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

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

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 $D_{\bullet k}, A_{\bullet k}$ 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 Sects. 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 [16,33], 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 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 s to 3 h.

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

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

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.

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 Sect. 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.

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	1,435	18	435
RSyn0810M04M		821	336	2,117	24	772
RSyn0815M02M		471	188	960	22	236
RSyn0820M02M		511	208	1,047	28	266
RSyn0830M02M		621	248	1,233	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	1,044	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	1,489	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	√	1,216	15	1,280	0	0
uflquad-20-40	√	821	20	840	0	0
uflquad-25-40	√	1,026	25	1,040	0	0

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 Sect. 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 Electronic supplementary material give the performance of each of these methods on each of the instances in our test suite. The tables are summarized in Fig. 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 Sect. 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

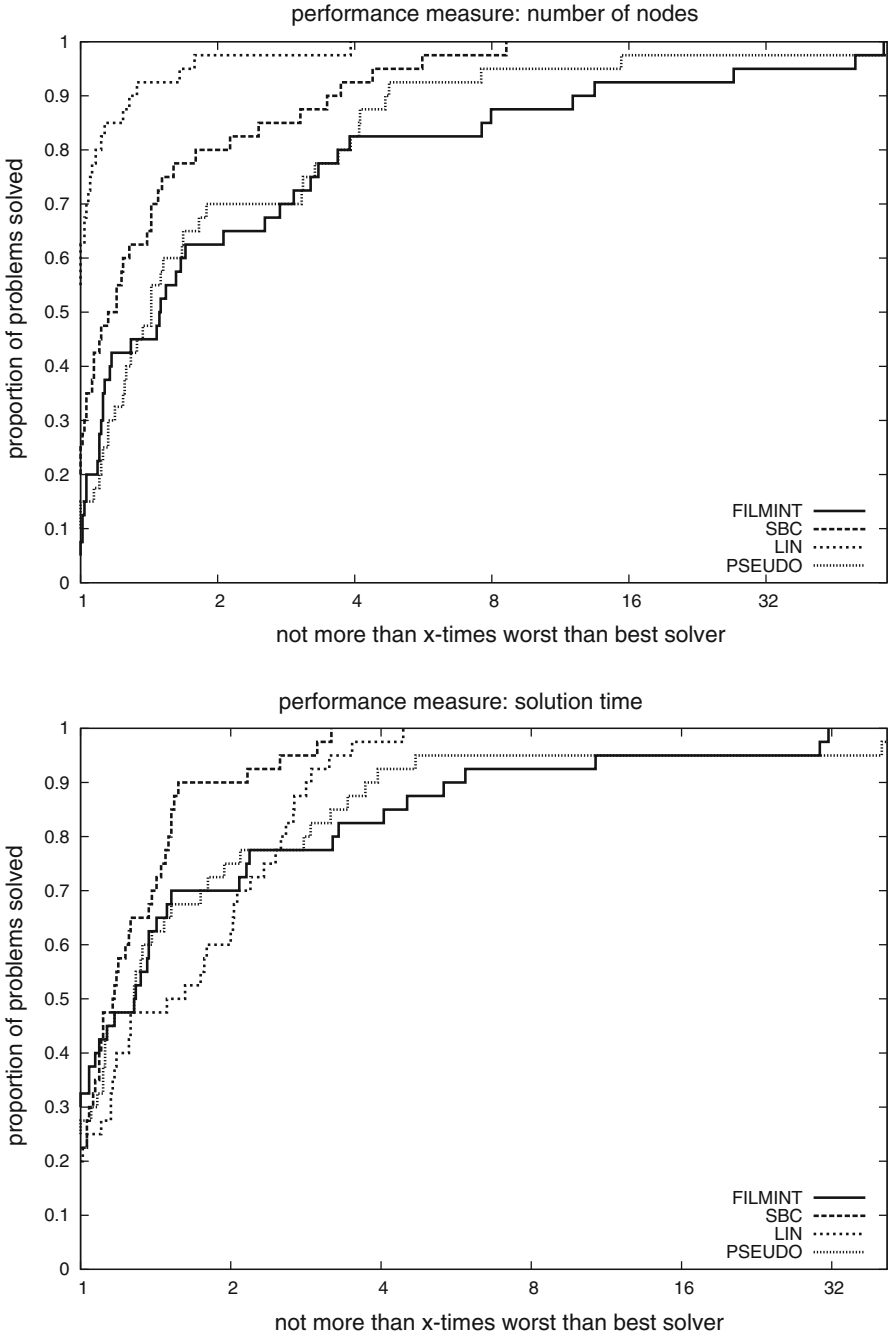

Fig. 1 Performance profile of elementary NLP-based strong-branching inequalities

Table 2 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

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 (see Electronic supplementary material). 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:

Table 3 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

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

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 (see Electronic supplementary material). 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 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 s and 8545 nodes are required to prove optimality when using MIXSBC, compared to only 33 s 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 Fig. 1 indicate that linearizations are very effective in reducing the number of nodes, but often lead to unacceptable solution times due

Table 4 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

Table 5 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

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 from the master problem. Full results of the performance of the two methods on each instance can be found in Table 11 (see Electronic supplementary material). 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 to 1744.2 s.

4.7 Performance of multiplier-strengthened SBC inequalities

Initial computational experience with the multiplier-strengthened cuts SSBC (16), introduced in Sect. 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 Sect. 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 perfor-

mance of the two methods on each instance can be found in Table 12 (see Electronic supplementary material).

4.8 Normalizations for CGLP

In Sect. 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 Fig. 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 SNC and the 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 (see Electronic supplementary material).

4.9 CGLP without strong branching

The CGLP described in Sect. 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.

Table 14 (see Electronic supplementary material). 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 Fig. 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

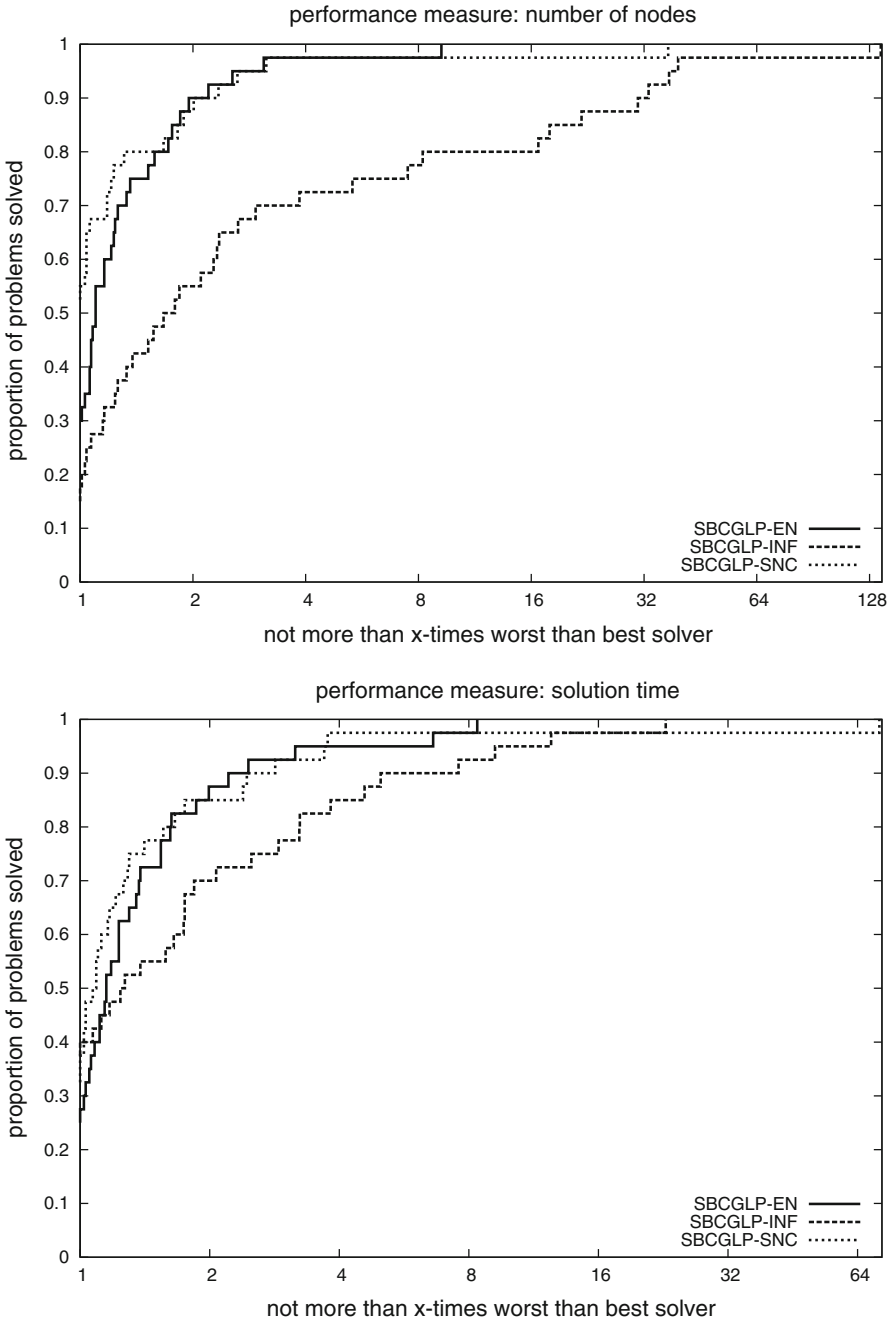

Fig. 2 Performance profile of CGLP with different normalization constraints

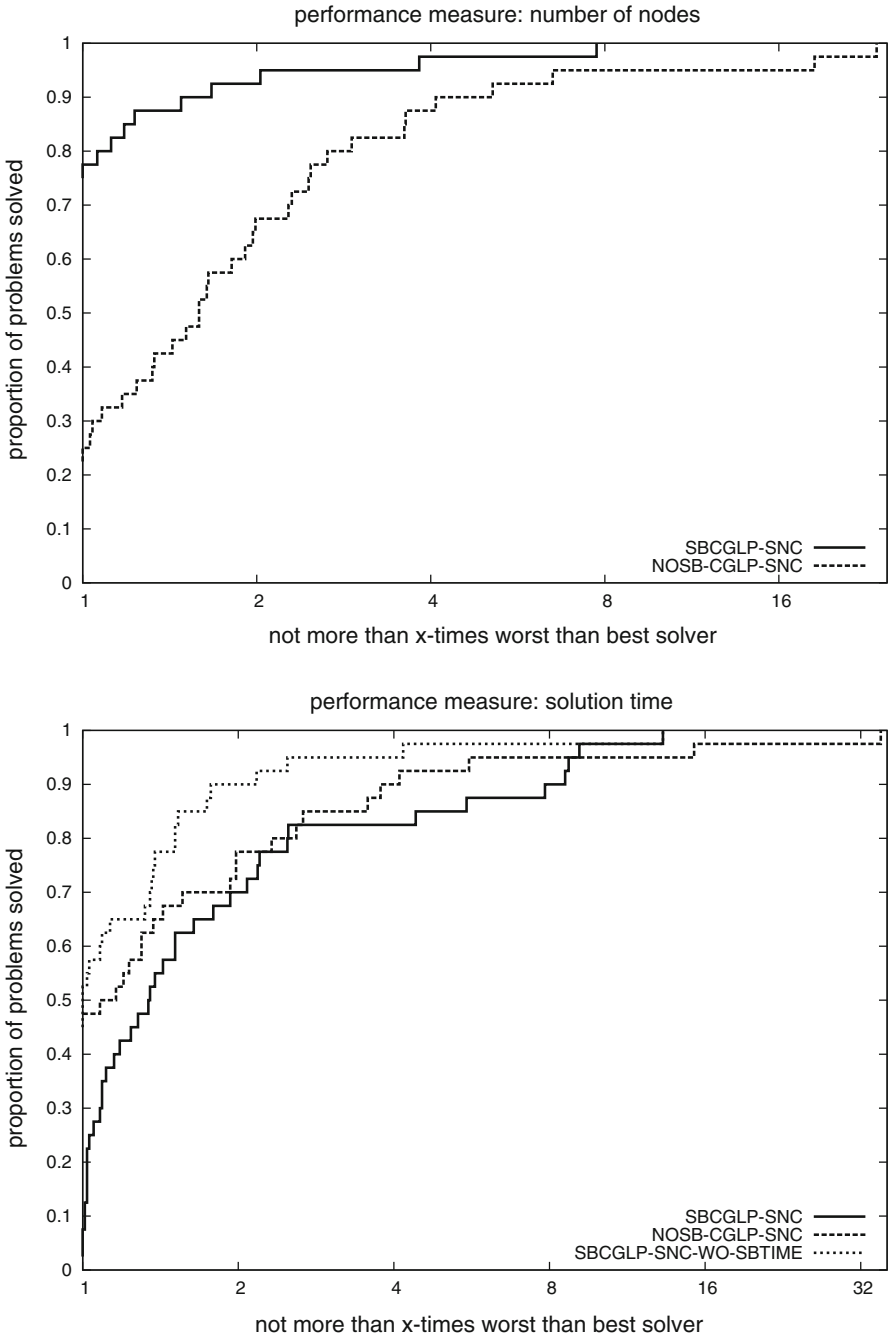

Fig. 3 Performance profile of CGLP with and without strong branching

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 Fig. 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 Sect. 2.1. The linearization management strategy introduced in Sect. 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 SNC.

The monoidal strengthening step described in Sect. 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 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

Table 6 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

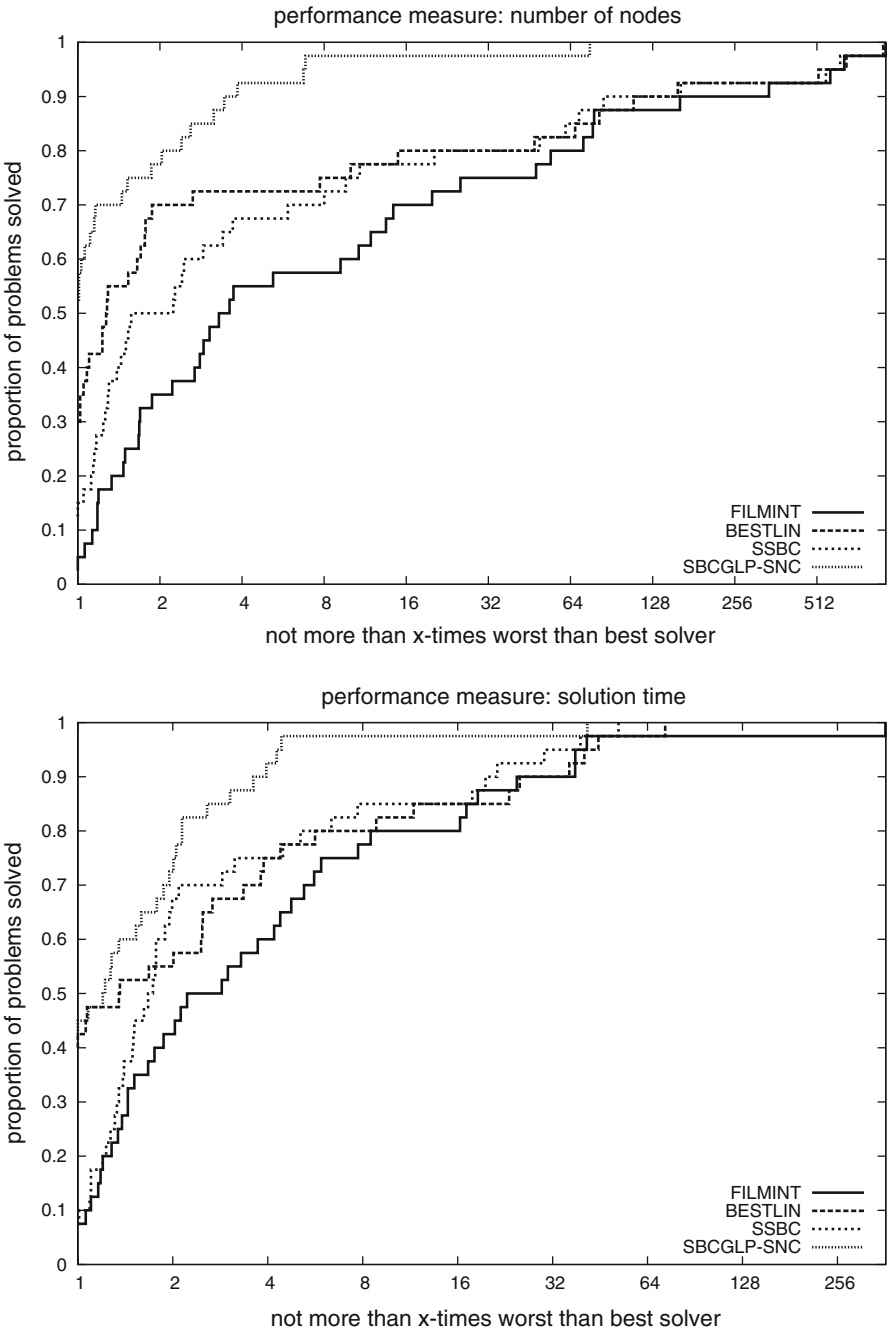

Fig. 4 Performance profile of best methods and FILMINT

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 Fig. 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 Sect. 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 16 (see Electronic supplementary material).

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 SBCs, and we demonstrated the relation of SBCs 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 SBCs 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.

Acknowledgments 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. Abhishek, K., Leyffer, S., Linderoth, J.T.: FilMINT: an outer-approximation-based solver for nonlinear mixed integer programs. INFORMS J. Comput. **22**, 555–567 (2010)
2. Achterberg, T.: Constraint Integer Programming. Ph.D. Thesis, Technischen Universtät Berlin (2007)
3. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Oper. Res. Lett. **33**, 42–54 (2004)
4. Applegate, D., Bixby, R., Cook, W., Chvátal, V.: The Traveling Salesman Problem, A Computational Study. Princeton University Press, Princeton (2006)
5. Atamtürk, A., Nemhauser, G., Savelsbergh, M.W.P.: Conflict graphs in solving integer programming problems. Eur. J. Oper. Res. **121**, 40–55 (2000)
6. Balas, E.: A modified lift-and-project procedure. Math. Program. **79**, 19–31 (1997)
7. Balas, E.: Disjunctive programming: properties of the convex hull of feasible points. Discr. Appl. Math. **89**(1–3), 3–44 (1998)
8. Balas, E., Ceria, S., Cornuéjols, G.: A lift-and-project cutting plane algorithm for mixed 0–1 programs. Math. Program. **58**, 295–324 (1993)

9. Balas, E., Ceria, S., Cornuéjols, G.: Mixed 0–1 programming by lift-and-project in a branch-and-cut framework. Manag. Sci. **42**, 1229–1246 (1996)
10. Balas, E., Jeroslow, R.G.: Strengthening cuts for mixed integer programs. Eur. J. Oper. Res. **4**, 224–234 (1980)
11. Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A.: Mixed-integer nonlinear optimization. Acta Numer. **22**, 1–131 (2013)
12. Bonami, P., Kılınç, M., Linderoth, J.: Algorithms and software for solving convex mixed integer nonlinear programs. IMA Vol. Math. Appl. **54**, 1–40 (2012)
13. Bonami, P., Lee, J., Leyffer, S., Wächter, A.: 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. Boorstyn, R.R., Frank, H.: Large-scale network topological optimization. IEEE Trans. Commun. **25**, 29–47 (1997)
15. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib - a collection of test models for mixed-integer nonlinear programming. INFORMS J. Comput. **15**(1), 114–119 (2003)
16. Castillo, I., Westerlund, J., Emet, S., Westerlund, T.: Optimization of block layout deisgn problems with unequal areas: a comparison of MILP and MINLP optimization methods. Comput. Chem. Eng. **30**, 54–69 (2005)
17. Dolan, E., Moré, J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
18. Duran, M.A., Grossmann, I.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Math. Program. **36**, 307–339 (1986)
19. Elhedhli, S.: Service system design with immobile servers, stochastic demand, and congestion. Manuf. Serv. Oper. Manag. **8**(1), 92–97 (2006)
20. Fischetti, M., Lodi, A., Tramontani, A.: On the separation of disjunctive cuts. Math. Programm. **128**, 205–230 (2011)
21. Fletcher, R., Leyffer, S.: User manual for filterSQP. University of Dundee Numerical Analysis Report NA-181 (1998)
22. Grossmann, I.E.: Review of nonlinear mixed-integer and disjunctive programming techniques. Optim. Eng. **3**, 227–252 (2002)
23. Günlük, O., Lee, J., Weismantel, R.: MINLP strengthening for separaable convex quadratic transportation-cost ufl. Technical Report RC24213 (W0703–042), IBM Research Division, March (2007)
24. Günlük, O., Pochet, Y.: Mixing mixed-integer inequalities. Math. Program. **90**(3), 429–457 (2001)
25. Harjunkoski, I., Pörn, R., Westerlund, T.: MINLP: Trim-loss problem. In: Floudas, Christodoulos A., Pardalos, Panos M. (eds.) Encyclopedia of Optimization, pp. 2190–2198. Springer, New York (2009)
26. Kılınç, M.: Disjunctive Cutting Planes and Algorithms for Convex Mixed Integer Nonlinear Programming. Ph.D. Thesis, University of Wisconsin-Madison (2011)
27. Leyffer, S.: MacMINLP: Test Problems for Mixed Integer Nonlinear Programming, (2003). http://www.mcs.anl.gov/~leyffer/macminlp
28. Linderoth, J.T., Savelsbergh, M.W.P.: A computational study of search strategies in mixed integer programming. INFORMS J. Comput. **11**, 173–187 (1999)
29. Nemhauser, G.L., Savelsbergh, M.W.P., Sigismondi, G.C.: MINTO, a Mixed INTeger Optimizer. Oper. Res. Lett. **15**, 47–58 (1994)
30. Pochet, Y., Wolsey, L.: Lot sizing with constant batches: formulation and valid inequalities. Math. Oper. Res. **18**, 767–785 (1993)
31. Quesada, I., Grossmann, I.E.: An LP/NLP based branch-and-bound algorithm for convex MINLP optimization problems. Comput. Chem. Eng. **16**, 937–947 (1992)
32. Ravemark, D.E., Rippin, D.W.T.: Optimal design of a multi-product batch plant. Comput. Chem. Eng. **22**(1–2), 177–183 (1998)
33. Sawaya, N.: Reformulations, relaxations and cutting planes for generalized disjunctive programming. Ph.D. Thesis, Chemical Engineering Department, Carnegie Mellon University (2006)
34. Sawaya, N., Laird, C.D., Biegler, L.T., Bonami, P., Conn, A.R., Cornuéjols, G., Grossmann, I.E., Lee, J., Lodi, A., Margot, F., Wächter, A.: CMU-IBM open source MINLP project test set, (2006). http://egon.cheme.cmu.edu/ibm/page.htm
35. Saxena, A., Bonami, P., Lee, J.: Convex relaxations of non-convex mixed integer quadratically constrained programs: extended formulations. Math. Program. Ser. B **124**, 383–411 (2010)

36. Stubbs, R., Mehrotra, S.: A branch-and-cut method for 0–1 mixed convex programming. Math. Program. **86**, 515–532 (1999)
37. Türkay, M., Grossmann, I.E.: Logic-based MINLP algorithms for the optimal synthesis of process networks. Comput. Chem. Eng. **20**(8), 959–978 (1996)
38. Vecchietti, A., Grossmann, I.E.: LOGMIP: a disjunctive 0–1 non-linear optimizer for process system models. Comput. Chem. Eng. **23**(4–5), 555–565 (1999)