PROCESS DESIGN AND CONTROL

Branch-and-Cut Algorithmic Framework for 0−1 Mixed-Integer Convex Nonlinear Programs

Hao Wu,[†,‡] Hao Wen,[‡] and Yushan Zhu*[,†]

Department of Chemical Engineering, Tsinghua University, Beijing 100084, China, and Institute of Process Engineering, Chinese Academy of Sciences, Beijing 100190, China

With widespread use of mixed-integer modeling for continuous and discontinuous nonlinear chemical processes, a robust mixed-integer nonlinear optimization algorithm is strongly demanded for the design and operation of chemical processes that achieve greater profits and also satisfy environmental and safety constraints. In this article, an extensive evaluation of a branch-and-cut algorithmic framework for 0−1 mixed-integer convex nonlinear programs, called MINO for Mixed-Integer Nonlinear Optimizer, is presented. The numerical performances of MINO are tested against four sets of medium-sized practical application problems from the fields of operations research and chemical engineering. The effects of the nonlinear programming (NLP) solvers, the cut generating and maintaining strategy, the node selection method, and the frequency of cut generation are examined, in order to handle MINLP problem with different mathematical structures. Preliminary computational results show that MINO exhibits a capability for handling practical MINLP problems that is comparable to that of the commercial solvers SBB and DICOPT, and it shows better performance for some problems for which either commercial solver encounters convergence problem within a CPU time limit of 6 h.

1. Introduction

Mixed-integer optimization provides a powerful framework for mathematically modeling many process optimization problems involving both discrete and continuous variables. It has achieved wide and successful applications in process systems engineering, such as in process or product design problems and in batch-process scheduling problems.[1–6] Whereas mixed-integer linear programming (MILP) methods and codes have been available for more than 30 years,[7] the number of methods and computer codes for solving mixed-integer nonlinear programming (MINLP) problems is still rather limited. SBB[8] solves MINLP problems by implementing a branch-and-bound framework in which a sequential quadratic programming (SQP) algorithm[9] is used to obtain the solution to the continuous relaxation NLP subproblem. BARON[10] is a computational system that is also based on the branch-and-bound approach, but its striking feature is the ability to solve nonconvex MINLP problems to global optimality. DICOPT[11] is an algorithmic implementation of the well-known out-approximation (OA) algorithm.[12,13] α-ECP is a code that executes the extended cutting plane method[14,15] for MINLP problems. MINOPT[16] implements the OA and GBD[2] (generalized Benders decomposition) methods and has been applied into solving the mixed-integer dynamic optimization problems. SBB and DICOPT are commercially available solvers in the modeling system GAMS.[17] Grossmann[3] reviewed these codes in 2002 and summarized the cases in which each code is likely to perform best. The recent creation of COIN-OR[18] provides reusable open-source software for LP, MILP, NLP, and MINLP. It contains three MINLP packages: B-BB based on the branch-and-bound algorithm; B-OA based on the OA algorithm; and B-Hyb, which is a hybrid code that implements a branch-and-cut-based OA algorithm.[19] An algorithmic framework for these open-source codes is provided by Bonami et al.[20]

Although branch-and-bound-type algorithms for MILP have achieved great success during the past 40 years, they encounter difficulties for MINLP problems because the integral gap between an MINLP problem and its continuous relaxation problem is much larger than that for MILP.[10] After observing that a valid cutting plane can improve the bounding property of a branch-and-bound algorithm,[38] Zhu and Kuno[21] proposed a branch-and-cut algorithm for 0−1 mixed-integer nonlinear programs, where the disjunctive cut is produced by solving a cut generation linear program (CGLP) based on the polyhedral approximation of the mixed-integer nonlinear set of the MINLP problem. Recently, Zhu et al.[5] improved this algorithm further by replacing the cut generation approach with the CGLP proposed by Balas et al. for MILP problems.[22–24] However, most of the test problems used have been very small, and the optimal settings of the algorithmic parameters have not yet been exploited. In this article, the effectiveness of this improved branch-and-cut algorithm is evaluated systematically by larger operations research and chemical engineering problems. The effects of the NLP solvers, the cut generating and maintaining strategy, the node selection method, and the frequency of cut generation are taken into account to check their impact on the performance of the algorithm for different structured MINLP problems. The results are compared with those obtained by the commercial MINLP solvers SBB and DICOPT, as reported by Bonami et al.[20]

2. Theoretical Background

The 0−1 mixed-integer nonlinear programming problem considered in this work can be formulated as

* To whom correspondence should be addressed. E-mail: yszhu@tsinghua.edu.cn.
† Tsinghua University.
‡ Chinese Academy of Sciences.

10.1021/ie9001074 CCC: $40.75
Published on Web 09/18/2009

MINLP

$$\begin{aligned} \min\ & Z = c^{\mathrm{T}}x \\ \text{s.t.}\ & g_i(x,y) \le 0, \quad i = 1, ..., l \\ & Ax + Gy \le b \\ & x \in \mathscr{R}^n, \quad y \in \{0,1\}^q \end{aligned}$$

where the constant vectors and matrices for the linear constraint set are defined as $c \in \mathscr{R}^n$, $A \in \mathscr{R}^{m\times n}$, $G \in \mathscr{R}^{m\times q}$, and $b \in \mathscr{R}^m$. A general 0−1 MINLP problem with a nonlinear objective function can be transformed into the above form by minimizing an additional continuous variable and transforming the mixed-integer objective function into an additional mixed-integer inequality.[21]

Let (F_0,F_1) denote the sets of binary variables fixed at 0 and 1, respectively, and let F denote the remaining ones. The subproblem of MINLP at node (F_0,F_1) on the enumeration tree can be formulated as

MINLP(F_0,F_1)

$$\begin{aligned} \min\ & Z_{\mathrm{BB}} = c^{\mathrm{T}}x \\ \text{s.t.}\ & g_i(x,y) \le 0 \quad i = 1, ..., l \\ & Ax + Gy \le b \\ & y_j = 0, \quad j \in F_0 \\ & y_j = 1, \quad j \in F_1 \\ & y_j \in \{0,1\}, \quad j \in F \\ & x \in \mathscr{R}^n, \quad y \in \mathscr{R}^q \end{aligned}$$

The base of the branch-and-cut algorithm is a branch-and-bound (BB) type of enumeration approach where the lower bound for the subproblem is computed by its continuous relaxations

NLP(F_0,F_1)

$$\begin{aligned} \min\ & Z_{\mathrm{LBB}} = c^{\mathrm{T}}x \\ \text{s.t.}\ & g_i(x,y) \le 0 \quad i = 1, ..., l \\ & Ax + Gy \le b \\ & y_j = 0, \quad j \in F_0 \\ & y_j = 1, \quad j \in F_1 \\ & 0 \le y_j \le 1, \quad j \in F \\ & x \in \mathscr{R}^n, \quad y \in \mathscr{R}^q \end{aligned}$$

The branch-and-bound procedure is described in pseudocode in Figure 1.

The lower bound in the branch-and-bound approach can be strengthened with valid cutting planes. This leads to the combination of the cutting-plane approach with the branch-and-bound or the branch-and-cut algorithm for MINLP. Let $(\bar{x},\bar{y})$ be a solution to the problem NLP(F_0,F_1), called the relaxation solution. If any of the components of the binary variables y are fractional, a valid inequality, described by $\alpha^{\mathrm{T}}x + \beta^{\mathrm{T}}y \le \gamma$, can be added to the current feasible set such that this inequality is violated by $(\bar{x},\bar{y})$. Denote by NLP(C,F_0,F_1) the continuous relaxation problem of the branch-and-cut algorithm, where C represents the set of the cutting planes added at the current node.

A brief review of our previous work on the branch-and-cut procedure is given below. The lift-and-project cuts, which were originally developed for MILP problems by Balas et al.,[24] are generated by a polyhedral approximation of MINLP(C,F_0,F_1), denoted as LP(C,F_0,F_1), as

LP(C,F_0,F_1)

$$\begin{aligned} \min\ & Z_{\mathrm{LP}} = c^{\mathrm{T}}x \\ \text{s.t.}\ & g_i(\bar{x},\bar{y}) + \nabla g_i(\bar{x},\bar{y})^{\mathrm{T}}\begin{pmatrix} x-\bar{x} \\ y-\bar{y}\end{pmatrix} \le 0 \quad i = 1, ..., l \\ & (\alpha^k)^T x + (\beta^k)^T y \\ & Ax + Gy \le b \\ & y_j = 0, \quad j \in F_0 \\ & y_j = 1, \quad j \in F_1 \\ & 0 \le y_j \le 1, \quad j \in F \\ & x \in \mathscr{R}^n, \quad y \in \mathscr{R}^q \end{aligned}$$

where $\alpha^{\mathrm{T}}x + \beta^{\mathrm{T}}y \le \gamma$ is the cut added to the NLP relaxation problem. To simplify the notation, let $\tilde{A}x + \tilde{G}y \le \tilde{b}$ represent the set of constraints on LP(C,F_0,F_1). If there is a variable y_k such that $0 < y_k < 1$ in $(\bar{x},\bar{y})$, a lift-and-project cut $\alpha^{\mathrm{T}}x + \beta^{\mathrm{T}}y \le \gamma$ is obtained by solving the following cut generating linear program

CGLP

$$\begin{aligned} \max\ & \alpha^{\mathrm{T}}\bar{x} + \beta^{\mathrm{T}}\bar{y} - \gamma \\ \text{s.t.}\ & \alpha - \tilde{A}^{\mathrm{T}}u \le 0 \\ & \alpha - \tilde{A}^{\mathrm{T}}v \le 0 \\ & \beta - \tilde{G}^{\mathrm{T}}u - u_0 e_k \le 0 \\ & \beta - \tilde{G}^{\mathrm{T}}v + v_0 e_k \le 0 \\ & \gamma - u^{\mathrm{T}}\tilde{b} = 0 \\ & \gamma - v^{\mathrm{T}}\tilde{b} + v_0 = 0 \\ & \sum_i u_i + u_0 + \sum_i v_i + v_0 = 1 \\ & u, u_0, v, v_0 \ge 0 \end{aligned}$$

The cut vectors can be expressed by the solutions to the CGLP, as

$$\begin{aligned} &\alpha_j = \min\{u^{\mathrm{T}}\tilde{A}_j, v^{\mathrm{T}}\tilde{A}_j\} && \text{for } j = 1, ..., n \\ &\beta_j = \min\{u^{\mathrm{T}}\tilde{G}_j, v^{\mathrm{T}}\tilde{G}_j\} && \text{for } j \in \{1, ..., q\}, j \ne k \\ &\beta_j = \min\{u^{\mathrm{T}}\tilde{G}_j + u_0, v^{\mathrm{T}}\tilde{G}_j - v_0\} && \text{for } j = k \\ &\gamma = \min\{u^{\mathrm{T}}\tilde{b}, v^{\mathrm{T}}\tilde{b} + v_0\} \end{aligned} \tag{1}$$

The cut can be produced in a reduced space because some binary variables are fixed at either their lower or upper bounds and can be eliminated before the CGLP is constructed; the detailed procedure is presented in our previous work.[5] Figure 2 describes the procedure of the branch-and-cut algorithm in pseudocode.

3. Algorithmic Framework

The branch-and-cut procedure for 0−1 MINLP described in this work was implemented in an algorithmic package entitled

```
Initialize S := {F0 = φ, F1 = φ}, UBD := +∞
While S ≠ φ do
    select node (F0, F1) ∈ S
    S := S \ (F0, F1)
    solve relaxation NLP (F0, F1)
    if NLP (F0, F1) is feasible
        let (x̄, ȳ) be optimal solution of NLP (F0, F1)
        if cx̄ < UBD
            if ȳ is integer
                (x*, y*) := (x̄, ȳ)
                UBD := cx̄
            else (branch)
                select a j with ȳj fractional
                S := S ∪ (F0 ∪ {j}, F1)
                S := S ∪ (F0, F1 ∪ {j})
            endif
        else
            fathom the node by bound
        endif
    else
        fathom the node
    endif
enddo
```

Figure 1. Pseudocode of branch-and-bound procedure.

MINO (i.e., Mixed-Integer Nonlinear Optimizer) written in the C programming language. Two versions of MINO packages were coded with differences in the NLP and LP solvers. The first one uses the interior-point NLP solver IPOPT[18,25] and the LP solver ILOG CPLEX 10.0.[26] The other one solves the NLP subproblem using the active-set solver LSGRG[27,28] and the CGLP using LINDOAPI 2.0.[29] The tests were run on a workstation with a 2.0-GHz central processing unit (CPU) and 4G of random-access memory (RAM).

3.1. Cut Generating Strategy and Frequency. For use of lift-and-project cutting planes in a branch-and-cut procedure, Balas et al.[24] concluded that it is better to generate cuts in large rounds rather than one at a time or in small rounds. Consequently, our implementation generates rounds of cuts for all 0−1 variables that are fractional in the current solution or some upper bound, whichever is smaller. A fixed-skip-factor (SK) strategy is used to control the frequency of cut generation, which means that one round of cuts is generated every SK nodes of the enumeration tree for a fixed integer value of SK. The skip factor SK was chosen between 4 and 16 for the test problems.

3.2. Cut Management. For large-scale 0−1 MINLP problems, if all cuts generated in the course of the algorithm are kept active in the problem formulation, the NLP and CGLP would become too large and be too much of a computational burden for NLP and LP solvers. The concept of a cut pool, which was first proposed by Padberg and Rinaldi[30] and used by Balas et al.,[31] was used to manage cuts. Figure 3 gives the description of the cut pool. There are two lists in the pool, the active list and the inactive list. The active-set solver LSGRG does not perform well for large-scale problems because of its exponential complexity, so the number of cuts added to the NLP problems should be limited. Therefore, the total number of cuts allowed in the cut pool has an upper limit of 1000 in this work. All cuts are added to the cut pool and set to be active until the limit of the cut pool is half-fulfilled. When there are more than half of the limit number of cuts in the cut pool (500 in this work), all cuts are evaluated by the solution $(\bar{x},\bar{y})$ retrieved at the new node; the violated inequalities are added to the active

```
Initialize S := {F0 = φ, F1 = φ}, UBD := +∞  C = empty
While S ≠ φ do
    select node (F0, F1) ∈ S
    S := S \ (F0, F1)
    solve relaxation NLP (C, F0, F1)
    if NLP (C, F0, F1) is feasible
        let (x̄, ȳ) be optimal solution of NLP (C, F0, F1)
        if cx̄ < UBD
            if ȳ is integer
                (x*, y*) := (x̄, ȳ)
                UBD := cx̄
            else (branch)
                select a j with ȳj fractional
                S := S ∪ (F0 ∪ {j}, F1)
                S := S ∪ (F0, F1 ∪ {j})
            endif
            if mod(nodes, skip factor) = 0 (generate cuts)
                create CGLP
                solve CGLP and generate cut: α^T x + β^T y ≤ γ
                add cut: α^T x + β^T y ≤ γ   to the constraint set
            endif
        else
            fathom the node by bound
        endif
    else
        fathom the node
    endif
enddo
```

Figure 2. Pseudocode of branch-and-cut procedure.

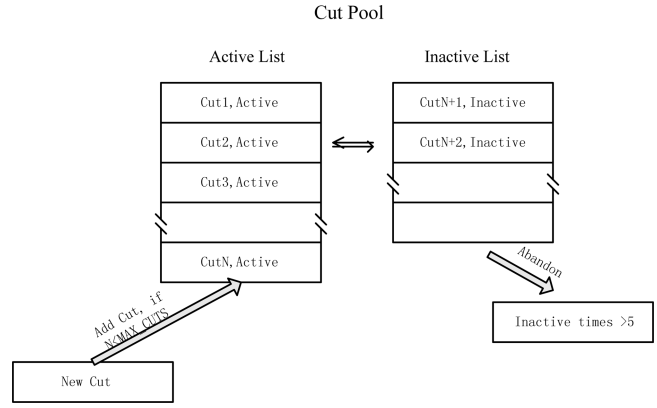

Figure 3. Cut pool.

list, and the rest to the inactive list. If a cut is inactive for more than five times in total, it is abandoned and removed from the cut pool. After the number of cuts reaches the upper limit, cut generation stops until more inactive cuts are abandoned and the cut pool has room for new cuts.

During a round of cut generation, it is very likely that the same cuts will be generated for different values of y_k. To avoid adding the same cuts, a new cut is added to the cut pool only if the cosine value of the angle between its normal vector and that of each previously added cut is less than θ, where θ is a parameter that was taken to be between 0.9 and 0.999 for the test problems.

3.3. Branching Rules and Node Selection Strategy. If the current node needs to be branched, a 0−1 variable whose current solution is fractional is selected for branching. In our implementation, we simply chose the binary variable y_k, $k \in F$, whose current value is the closest to 0.5

$$k = \arg\min_k \{|y_k - 0.5|\}$$

There are three popular node selection strategies for the enumeration tree for the standard branch-and-bound algorithm, namely, best-bound, depth-first, and breadth-first. The best-bound strategy was chosen here and was implemented by listing the nodes in increasing order of objective function value and always picking the first one. For comparison, some tests for the depth-first strategy are also presented.

4. Numerical Experiments

The problems used in the numerical experiments are typical applications from operations research and chemical engineering. All of these problems are available on the web in AMPL and GAMS formats.[32] The BatchS[33] problems are multiproduct batch plant design problems with multiple units in parallel and intermediate storage tanks. These problems consist of determining the volume of the equipment, the number of units in parallel, and the volumes and locations of the intermediate storage tanks. The nonlinearities in this set of problems stem from exactly one constraint that contains an exponential term. The SLay[34] problems are safety layout problems, where one is interested in placing a set of units with fixed width and length such that the Euclidean distance between their center point and a predefined "safety point" is minimized. This type of problem is a mixed-integer quadratic program, and thus, the nonlinearities in this set of problems are contained solely in the objective function (as quadratic terms). The Syn[12,35] problems are synthesis[34] problems and correspond to the synthesis portion of retrofit−synthesis problems. The cutting stock problems, called Trimloss,[36] are problems in which one is interested in cutting out a set of product paper rolls from raw paper rolls such that the cost function, including the trim loss as well as the overproduction, is minimized. The nonlinearities in this set of problems arise from square-root transformations that were used to remove the nonconvexity of a set of bilinear constraints. Table 1 lists the characteristics of the test problems. The Slay04M problem is used only for comparison of the relaxation solutions obtained by different NLP solvers and is not included in other numerical tests. Tables 2 and 3 list the computational results for the branch-and-bound algorithm and branch-and-cut algorithms, respectively, with the default settings (SK = 4, θ = 0.99, max cut =1000).

4.1. Impact of NLP Solving Method. The stability and robustness of IPOPT are better than those of LSGRG, so we intended to use IPOPT as the NLP solver first. However, both branch-and-bound and branch-and-cut methods encounter difficulties for the Slay and Trimloss problems. Table 4 gives the relaxation solution of the Slay04M problem on the root node obtained by different NLP solvers. Note that, even though the optimal values are the same, the optimal solutions are quite different from each other. For the solution obtained by LSGRG, most binary variables have values fixed at 0 or 1. In contrast, none of the binary variables is 0 or 1 in the solution obtained by IPOPT. This can be explained by that the solution obtained by an interior-point solver will tend to be on the center of the optimal "face" when an NLP has more than one optimal solution, whereas the solution obtained by an active-set solver will lie at an "extreme point" of the face. The integral solutions are always obtained at the extreme points, so the number of enumeration nodes is larger when using an interior-point method to solve relaxation problems. It should be noted that, to have a clearer understanding of the effects of the NLP algorithm, a more thorough analysis might be required.

No reliable open-source active-set NLP solver is currently available. LSGRG was used for comparison with IPOPT. As the problem scale increases, the branch-and-bound method with IPOPT as its NLP solver encounters a bottleneck. The number of enumeration nodes solved is over 100000 for Slay07M, whereas only 211 nodes are needed when using LSGRG. However, the stability of LSGRG is not reliable, as numerical difficulties occur when solving large-scale problems. Therefore, LSGRG was used only on those problems that could not be solved when using IPOPT as the NLP solver, including the Slay problems and Trimloss5 and Trimloss6. Because the IPOPT version of the MINO program is more reliable and has much more flexibility than the LSGRG version, most of the remaining numerical tests were run with the IPOPT version.

4.2. Node Selection Strategy. Table 5 lists the results obtained using the depth-first node selection strategy. Compared with the best-bound approach (Table 3, with the results of the Slay problems as obtained by the LSGRG version in Tables 6 and 7), the depth-first strategy shows great advantage on BatchS151208M, BatchS201210M, and Syn20M04M and also shows great disadvantage on BatchS101006M, Syn20M02M, and the Slay problems. It is hard to determine which strategy is likely to perform best on a given set of test problems. The best-bound strategy, which exhibited better performance, on average, was chosen for other numerical experiments.

4.3. Impact of Skip Factor. A small skip factor means that cuts are generated more often, which leads to an increase in the number of CGLPs and the computational cost. However, a small skip factor also results in the generation of more cuts after the solution of the same number of enumeration nodes on the branch-and-bound tree. If those cuts are tight enough, because of the greatly strengthened lower bound, the number of nodes that need to be branched could be greatly reduced, thereby accelerating the solution process. Because there is no guarantee that the decrease in the number of enumeration nodes exceeds the computational time of the CGLP, the efficiency of the branch-and-cut algorithm is related to the balance between the computational cost of the CGLP and the cut updating frequency.

When a low value for the skip factor is used, too many cuts are generated in each round, but the cuts are almost parallel to each other, which tends to cause numerical problems for NLP solvers. Based on this observation, the range of skip factors chosen in this work was 4−16. The performances of fixed skip factors with values of 4−16 were compared for different problems; Tables 6 and 7 report the results. Generally, the number of enumerated nodes (equal to the number of NLPs solved) is smaller when a small skip factor is used. This indicates that the cuts presented in this work are strongly effective in most cases and that the computing time saved by cuts can exceed the computational cost of the CGLP. This is translated into an advantage in total solution time. For only a few problems, such as BatchS201210M, the impact of the skip factor was ambiguous. Figure 4 shows a performance plot[37] of branch-and-cut algorithm with different skip factors and the typical branch-and-bound algorithm corresponding to Table 6. The curve plotted for a given algorithm A gives the proportion of problems that are solved within a factor of p of the time required by the best algorithm [i.e., p = (solution time of algorithm A)/(solution time of the best algorithm)]. For $p = 1$, the plotted point for algorithm A represents the proportion of problems for which algorithm A is fastest. Figure 4 clearly shows that the smallest skip factor, 4, provides the best performance. Skip factors 8 and 16 exhibit similar levels of performance, which means that

Table 1. Characteristics of Test Problems

problem name	continuous variables	binary variables	constraints			optimal solution value[20]
			total	linear	nonlinear	
Trimloss2	7	31	22	20	2	5.3
Trimloss4	21	85	60	56	4	8.3
Trimloss5	31	131	90	85	5	<11.2[a]
Trimloss6	43	173	114	108	6	*b*
Slay04M	21	24	55	54	1	26142.77
Slay07M	57	84	190	189	1	64748.82
Slay08M	73	112	253	252	1	84960.21
Slay09M	91	144	325	324	1	107805.75
Slay10M	111	180	406	405	1	129579.88
BatchS101006M	151	130	1020	1009	11	769440
BatchS151208M	244	204	1782	1766	16	1543472
BatchS201210M	309	252	2424	2403	21	2296535
Syn20M02M	210	80	406	378	28	1752.13
Syn20M04M	260	160	1052	996	56	3532.74

[a] Best known solution. [b] No solution available.

Table 2. Results of the Branch-and-Bound Algorithm

problem name	NLPs	time	NLP solver
BatchS101006M	57470	6 h	IPOPT
BatchS151208M	>100000	>6 h	IPOPT
BatchS201210M	>100000	>6 h	IPOPT
Slay07M	>100000	>6 h	IPOPT
Slay07M	211	45.46 s	LSGRG
Slay08M	1600	95.44 s	LSGRG
Slay09M	1427	146.92 s	LSGRG
Slay10M	*a*	*a*	LSGRG
Syn20M02M	>100000	>6 h	IPOPT
Syn20M04M	>100000	>6 h	IPOPT
Trimloss2	283	10.46 s	IPOPT
Trimloss4	>100000	>6 h	IPOPT
Trimloss5	>100000	>6 h	IPOPT
Trimloss6	>100000	>6 h	IPOPT

[a] LSGRG encounters numerical difficulties within 6 h.

Table 3. Results of the Branch-and-Cut Algorithm under the Default Settings[a]

problem name	NLPs	LPs	cuts	NLP time	LP time (s)
BatchS101006M	1870	537	365	320.51 s	18.53
BatchS151208M	754	1785	154	304.37 s	201.48
BatchS201210M	8970	1560	1131	5683.57 s	233.47
Slay07M	*b*	*b*	*b*	>6 h	*b*
Slay08M	*b*	*b*	*b*	>6 h	*b*
Slay09M	*b*	*b*	*b*	>6 h	*b*
Slay10M	*b*	*b*	*b*	>6 h	*b*
Syn20M02M	2994	480	480	700.58 s	8.10
Syn20M04M	12158	3062	2710	9040.73 s	262.07
Trimloss2	254	44	44	4.34 s	0.04
Trimloss4	46910	2123	1675	17815.98 s	29.70
Trimloss5	*b*	*b*	*b*	>6 h	*b*
Trimloss6	*b*	*b*	*b*	>6 h	*b*

[a] SK = 4, θ = 0.99, best-bound node selection, max cut = 1000, NLP solver is IPOPT. [b] These problems cannot be solved within 6 h.

the effect of the cuts is counterbalanced by the computational cost of the CGLP under these two circumstances.

The computation time of MINO is proportional to the numbers of CGLPs and NLPs solved. The effects of the cuts on these two numbers are contrary to each other. On one side, more cuts mean more CGLPs solved and more CPU time expended on CGLP solving, but on the other side, more cuts could enhance the bounding property of the NLP in the enumeration process, thus greatly reducing the number of nodes for enumeration, which means fewer NLPs that need to be solved and, therefore, less computation time. Generally, it is much less computationally expensive to solve a CGLP than an NLP, so the best performance is observed for SK = 4.

Table 4. Relaxation Solutions of Slay04M Generated by Different NLP Solvers

	Z(1,1,2)[a]	Z(1,1,3)	Z(1,1,4)	Z(1,2,3)	Z(1,2,4)	Z(1,3,4)
LSGRG[b]	0.0	0.0	0.0	0.0	0.0	0.0
IPOPT[c]	0.2735	0.2557	0.2518	0.2318	0.2287	0.2458
	Z(2,1,2)	Z(2,1,3)	Z(2,1,4)	Z(2,2,3)	Z(2,2,4)	Z(2,3,4)
LSGRG	0.0	0.0	0.0	0.0	0.0	0.0
IPOPT	0.2246	0.2453	0.2518	0.2636	0.2717	0.2554
	Z(3,1,2)	Z(3,1,3)	Z(3,1,4)	Z(3,2,3)	Z(3,2,4)	Z(3,3,4)
LSGRG	0.3425	0.1478	0.0478	0.0	0.0	0.0
IPOPT	0.2804	0.2490	0.2315	0.2258	0.2054	0.2354
	Z(4,1,2)	Z(4,1,3)	Z(4,1,4)	Z(4,2,3)	Z(4,2,4)	Z(4,3,4)
LSGRG	0.6574	0.8521	0.9521	1.0	1.0	1.0
IPOPT	0.2212	0.2498	0.2647	0.2786	0.2940	0.2632

[a] Z(*D*,*i*,*j*) represents the *D*th disjunction for nonoverlapping constraints between rectangles *i* and *j*. [b] LSGRG is the active-set type NLP solver used in MINO. [c] IPOPT is the interior point type NLP solver used in MINO.

Table 5. Results of the Branch-and-Cut Algorithm with the Depth-First Node Selection Strategy[a]

problem name	NLPs	LPs	cuts	NLP time (s)	LP time (s)
BatchS101006M	5280	708	666	721.78	19.26
BatchS151208M	384	320	113	93.11	36.05
BatchS201210M	612	51	20	238.41	7.94
Slay07M*	295	108	50	11.29	1.58
Slay08M*	†	†	†	†	†
Slay09M*	17326	1219	1008	1282.34	36.50
Slay10M*	†	†	†	†	†
Syn20M02M	19596	2203	2203	1922.04	158.29
Syn20M04M	3258	413	413	2004.91	35.23
Trimloss2	626	56	56	8.98	0.07
Trimloss4	76592	6064	6046	15585.23	52.66

* The NLP solver used for Slay problems is LSGRG,LP solver is Lindoapi. † LSGRG encounters numerical difficulties within 6 h. ‡These problems cannot be solved within 6 h [a] SK = 4, θ = 0.99, max cut = 1000, NLP solver is IPOPT.

It should be noted that the test results for Trimloss5 and Trimloss6 problems are not listed, because the computing times of these two problems were too long. For the same reason, the results for these two problems are not listed for other tests. We just give the optimal solution that we found in the last section.

4.4. Cut Management. In a branch-and-cut framework, the pivotal role of the added cuts is to reduce the number of enumeration nodes. However, this might not translate into a large advantage in running time. This can be explained by an

Table 6. Numbers of NLPs and LPs Solved and Cuts Generated Using Different Skip Factors, for $\theta = 0.99$

	SK = 4			SK = 8			SK = 16		
problem name	NLPs	LPs	cuts	NLPs	LPs	cuts	NLPs	LPs	cuts
BatchS101006M	1870	537	365	5198	525	455	6362	407	288
BatchS151208M	754	1785	154	21152	1880	839	68608	5052	3408
BatchS201210M	8970	1560	1131	*a*	*a*	*a*	8386	2212	355
Slay07M[b]	192	121	30	147	44	8	147	44	8
Slay08M[b]	73	15	8	321	55	17	615	90	30
Slay09M[b]	502	350	96	718	241	75	614	165	35
Slay10M[b]	*c*	*c*	*c*	*c*	*c*	*c*	7277	952	236
Syn20M02M	2994	480	480	4986	502	502	7996	370	370
Syn20M04M	12158	3062	2710	*a*	*a*	*a*	*a*	*a*	*a*
Trimloss2	254	44	44	330	31	31	370	22	22
Trimloss4	46910	2123	1675	*a*	*a*	*a*	*a*	*a*	*a*

[a] These problems cannot be solved within 6 h. [b] The NLP solver used for Slay problems is LSGRG, and the LP solver is Lindoapi. [c] LSGRG encounters numerical difficulties.

Table 7. Solution Times under Different Skip Factors, for $\theta = 0.99$

	SK = 4			SK = 8			SK = 16		
problem name	NLP (s)	LP (s)	total (s)	NLP (s)	LP (s)	total (s)	NLP (s)	LP (s)	total (s)
BatchS101006M	320.51	18.53	339.04	804.08	15.38	819.46	1068.62	14.79	1083.41
BatchS151208M	304.37	201.48	505.85	7506.59	617.25	8123.84	20817.14	3293.31	24110.45
BatchS201210M	5683.57	233.47	5917.04	*a*	*a*	*a*	3569.75	1484.97	5054.72
Slay07M[b]	7.68	1.68	9.36	9.04	0.76	9.80	9.04	0.76	9.80
Slay08M[b]	4.02	0.52	4.54	14.84	1.54	16.38	43.16	2.81	45.97
Slay09M[b]	72.3	12.06	84.36	136.84	8.08	144.92	74.78	6.78	81.56
Slay10M[b]	*c*	*c*	*c*	*c*	*c*	*c*	1599.82	57.89	1657.71
Syn20M02M	700.58	8.10	708.68	685.32	15.51	700.83	1957.41	3.77	1961.18
Syn20M04M	9040.73	262.07	9302.80	*a*	*a*	*a*	*a*	*a*	*a*
Trimloss2	4.34	0.04	4.38	5.38	0.04	5.42	4.94	0.03	4.97
Trimloss4	17815.98	29.70	17845.68	*a*	*a*	*a*	*a*	*a*	*a*

[a] These problems cannot be solved within 6 h. [b] The NLP solver used for Slay problems is LSGRG, and the LP solver is Lindoapi. [c] LSGRG encounters numerical difficulties.

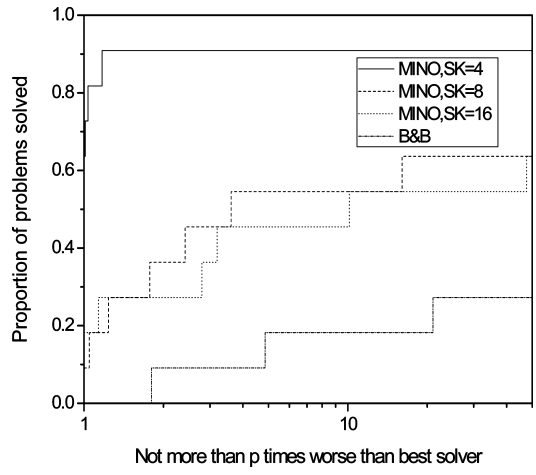

Figure 4. Relative performances of branch-and-cut algorithm with different skip factors and branch-and-bound algorithm.

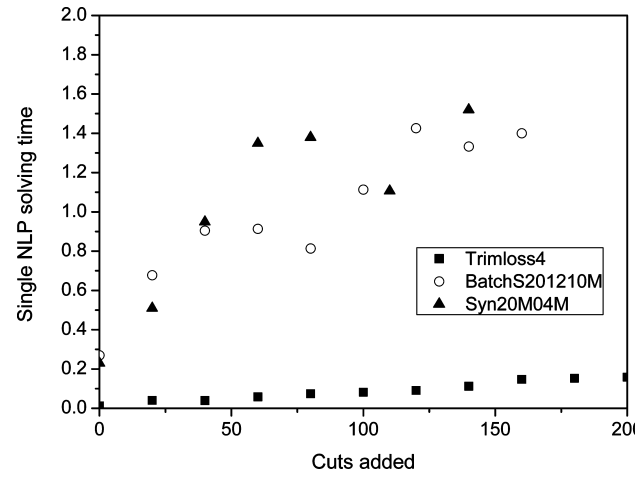

Figure 5. Effect of number of cuts on NLP solution time.

increase in the scale of the NLP problem and the average solution time for a single NLP. For the Trimloss4 problem, which includes 60 constraints, the solution time for a relaxation problem (NLP) is approximately 0.01 s. With 200 cuts added, the number of constraints in a single relaxation problem is 260, and the solution time is approximately 0.16 s. Figure 5 shows the relationship between the number of cuts added and the solution time for a relaxation problem. It shows a linear increase in solution time. For large problems, the increase in computational cost caused by the number of cuts could kill the profit saved by reducing the number of enumeration nodes. Moreover, a large number of cuts also necessitates a large amount of storage. Thus, the maximum number of cuts allowed in the cut pool should be limited. The maximum number of cuts allowed in this work was 1000. Cuts were validated by current relaxation solution $(\bar{x},\bar{y})$, and only the active cuts were added to the NLP and CGLP.

Different values of θ mean different levels of tolerance for cuts added to the cut pool. Tables 8 and 9 report the computational results for different θ values. The largest θ value of 0.999 has an advantage on BatchS201210M and Syn20M02M, whereas the θ value of 0.9 performs better for Slay09M. For the rest of the test problems, different θ values show comparative performances. Note that the θ value produces a very small effect on the Slay problems and, in particular, no effect on Slay08M. Because of the weak nonlinearity of the Slay problems, cuts generated by a linear approximation from different relaxation solutions are usually very different from each other. Therefore, different tolerances for the cuts (θ values) have little effect on

Table 8. Numbers of NLPs and LPs Solved and Cuts Generated Using Different Values of θ, for SK = 4

	$\theta = 0.9$			$\theta = 0.99$			$\theta = 0.999$		
problem name	NLPs	LPs	cuts	NLPs	LPs	cuts	NLPs	LPs	cuts
BatchS101006M	950	482	381	1870	537	365	854	490	448
BatchS151208M	‡	‡	‡	754	1785	154	454	1506	123
BatchS201210M	17566	2680	2401	8970	1560	1131	768	572	51
Slay07M*	325	151	58	192	121	30	192	121	30
Slay08M*	73	15	8	73	15	8	73	15	8
Slay09M*	400	249	95	502	350	96	661	337	101
Slay10M*	†	†	†	†	†	†	†	†	†
Syn20M02M	79350	16055	16055	2994	480	480	2832	565	565
Syn20M04M	11634	2804	2585	12158	3062	2710	8714	2411	2180
Trimloss2	388	96	95	254	44	44	248	59	59
Trimloss4	‡	‡	‡	46910	2123	1675	‡	‡	‡

* The NLP solver used for Slay problems is LSGRG, LP solver is Lindoapi. † LSGRG encounters numerical difficulties. ‡ These problems cannot be solved within 6 h.

Table 9. Solution Times under Different Values of θ, for SK = 4

	$\theta = 0.9$			$\theta = 0.99$			$\theta = 0.999$		
problem name	NLP (s)	LP (s)	total (s)	NLP (s)	LP (s)	total (s)	NLP (s)	LP (s)	total (s)
BatchS101006M	146.22	17.18	163.4	320.51	18.53	339.04	147.33	16.29	163.62
BatchS151208M	*a*	*a*	*a*	304.37	201.48	505.85	137.61	175.25	312.86
BatchS201210M	9942.13	197.01	10139.14	5683.57	233.47	5917.04	425.37	109.31	534.68
Slay07M[b]	17.24	1.78	19.02	7.68	1.68	9.36	7.68	1.68	9.36
Slay08M[b]	4.02	0.52	4.54	4.02	0.52	4.54	4.02	0.52	4.54
Slay09M[b]	42.3	6.96	49.26	72.3	12.06	84.36	85.59	9.15	94.74
Slay10M[b]	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*	*c*
Syn20M02M	8569.35	141.26	8710.61	700.58	8.1	708.68	574.22	5.07	579.29
Syn20M04M	7282.69	29.59	7312.28	9040.73	262.07	9302.8	4158.75	25.59	4184.34
Trimloss2	6.21	0.15	6.36	4.34	0.04	4.38	4.02	0.1	4.12
Trimloss4	*a*	*a*	*a*	17815.98	29.70	17845.68	*a*	*a*	*a*

[a] These problems cannot be solved within 6 h. [b] The NLP solver used for Slay problems is LSGRG, and the LP solver is Lindoapi. [c] LSGRG encounters numerical difficulties.

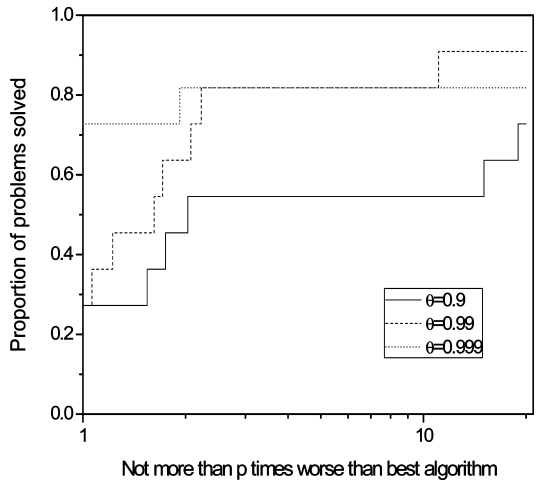

Figure 6. Relative performance of branch-and-cut algorithm with different values of θ.

these problems. Figure 6 is the performance plot for different θ values. It shows that different values of θ are close in average performance, whereas the smallest value of 0.9 has a slight disadvantage.

4.5. Comparison with Commercial Solvers. Table 10 gives the best results obtained by the branch-and-cut algorithm presented in this work and compared with the commercial MINLP solvers SBB and DICOPT. The results for SBB and DICOPT were reported by Bonami et al.[20] Note that the computing time for the results obtained by SBB is the CPU time reported by GAMS. This time is much smaller than the clock time on large problems. A computing time of >3 h means that the problem could not be solved within 3 h or did not converge at all. DICOPT was found to be particularly efficient

Table 10. Comparison with Commercial Solvers, Where the Data for the Commercial Solvers are from Reference20

problem name	SBB time[20]	DICOPT time[20]	branch-and-cut time
BatchS101006M	728.46 s	14.69 s	339.04 s
BatchS151208M	3871.95 s	22.53 s	505.85 s
BatchS201210M	>3 h	28.44 s	5054.72 s
Slay09M[a]	21.51 s	>3 h	84.36 s
Slay10M[a]	19.45 s	>3 h	1657.71 s
Syn20M04M	>3 h	0.27 s	9302.8 s
Trimloss4	>3 h	>3 h	4.9 h
Trimloss5[a]	>3 h	>3 h	3 h
Trimloss6[a]	not tried	not tried	6 h

[a] The NLP solver for these problems is LSGRG, and the LP solver is Lindoapi.

on the BatchS and Syn problems, with solution times of less than 30 s, whereas SBB dominated on the Slay problems. Neither commercial solver was efficient on Trimloss problems, and both failed to converge on Trimloss5 and Trimloss6. The computational results for the branch-and-cut algorithm show an efficiency comparable to that of the commercial solvers. For those problems on which the commercial solvers encountered difficulties, the branch-and-cut algorithm showed better performance because the lower bound was strengthened by lift-and-project cuts and the enumeration strategy was improved.

Figure 7 is the performance plot corresponding to Table 10. DICOPT shows a flat curve, because it is quite efficient on the problems it can solve and inefficient on the rest. The branch-and-cut algorithm shows similar performances for different sets of test problems, which indicates that it is well-suited for different kinds of MINLP problems. By choosing the optimal parameters and cut pool maintenance strategy, the branch-and-cut algorithm can be efficient for different kinds of MINLP

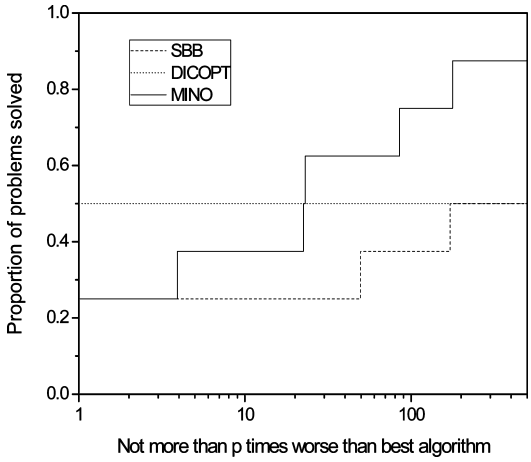

Figure 7. Relative performances of MINO and commercial solvers SBB and DICOPT, where the data of commercial solvers are from ref 20.

Table 11. Comparison with the Commercial Solver DICOPT under the Same Computing Environment with a 2.0-GHz CPU and 4G of RAM

problem name	DICOPT time	branch-and-cut time
BatchS101006M	4.86 s	339.04 s
BatchS151208M	17.83 s	505.85 s
BatchS201210M	33.93 s	5054.72 s
Syn20M02M	3.83 s	579.29 s
Syn20M04M	498.48 s	9302.8 s
Trimloss4	failure	4.9 h
Trimloss5[a]	failure	3 h
Trimloss6[a]	failure	6 h

[a] The NLP solver for these problems is LSGRG, and the LP solver is Lindoapi.

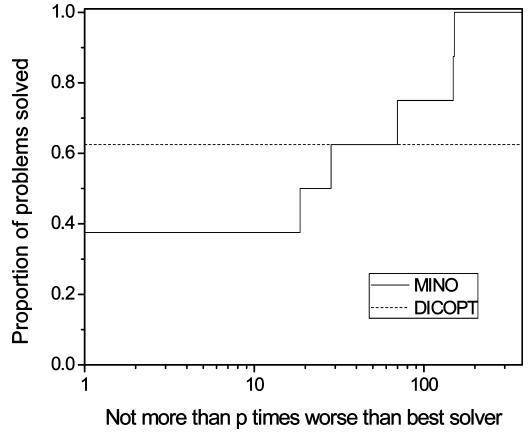

Figure 8. Relative performances of MINO and commercial solver DICOPT under the same computing environment with a 2.0-GHz CPU and 4G of RAM.

problems. For Trimloss5 and Trimloss6, both SBB and DICOPT failed to converge. In contrast, the branch-and-cut algorithm was able to find the optimal solution within acceptable time. The optimal solutions of Trimloss5 and Trimloss6 were found to be 10.8 and 20.7, respectively.

To strengthen the analysis, we checked the performance of MINO against the commercial solver DICOPT under the same computing environment for eight MINLP problems including the tough problems Trimloss 5 and Trimloss 6, and the results are presented in Table 11 and Figure 8. According to these results, a similar conclusion can be drawn, although there are some differences for the results shown in Tables 10 and 11 because of the different computing environments. The results suggest that MINO is more suitable for hard MINLP problems with strong nonlinearities. It should be noted, however, that, based on the examples, the improvement observed when using MINO might be due to the problem structure, given that all of the Trimloss problems, which are the ones for which MINO exhibits better performance, have the same structure.

5. Conclusion

An algorithmic framework for an improved branch-and-cut approach for 0−1 mixed-integer convex nonlinear programs is presented. The effectiveness of this algorithm was evaluated systematically against four sets of operation research and chemical engineering problems. The computational results show that solving the relaxation problem NLP(C,F_0,F_1) by the active-set method is better than solving it by the interior-point method in a enumeration tree. The branch-and-cut algorithm exhibits the best performance with the smallest skip factor of 4. Different node selection strategies and θ values provide peak performances on different test problems. Generally, the algorithmic parameters that were stable and best in average performance (SK = 4,θ = 0.99, best-bound node selection) were used. The efficiency of the branch-and-cut algorithm was found to be comparable to that of commercial solvers. For those problems for which the commercial solvers encountered difficulties, the branch-and-cut algorithm performed better because the lower bound was strengthened by lift-and-project cuts and the enumeration strategy was improved.

Acknowledgment

Y.Z. gratefully appreciates the financial support from the National Science Foundation of China (Nos. 20506013 and 20776075) and the National High Technology Research and Development (863) Program of China (Nos. 2006AA02Z337 and 2008AA02Z208).

Literature Cited

(1) Biegler, L. T.; Grossmann, I. E. Retrospective on optimization. *Comput. Chem. Eng.* **2004**, *28* (8), 1169–1192.

(2) Floudas, C. A. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*; Oxford University Press: New York, 1995.

(3) Grossmann, I. E. Review of Nonlinear Mixed-Integer and Disjunctive Programming Techniques. *Optim. Eng.* **2002**, *3* (3), 227–252.

(4) Grossmann, I. E.; John, L. A. Mixed-Integer Optimization Techniques for Algorithmic Process Synthesis. In *Advances in Chemical Engineering*; Academic Press: New York, 1996; Vol. 23, pp 171−246.

(5) Zhu, Y.; Hu, Y.; Wu, H.; Nakaiwa, M. An improved branch-and-cut algorithm for mixed-integer nonlinear systems optimization problem. *AIChE J.* **2008**, *54* (12), 3239–3247.

(6) Zhu, Y. S.; Kuno, T. Global optimization of nonconvex MINLP by a hybrid branch-and-bound and revised general benders decomposition approach. *Ind. Eng. Chem. Res.* **2003**, *42* (3), 528–539.

(7) Nemhauser, G.; Wolsey, L. *Integer and Combinatorial Optimization*; Wiley-Interscience: New York, 1988.

(8) Borchers, B.; Mitchell, J. E. An Improved Branch-and-Bound Algorithm for Mixed-Integer Nonlinear Programs. *Comput. Oper. Res.* **1994**, *21* (4), 359–367.

(9) Leyffer, S. Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Comput. Optim. Appl.* **2001**, *18* (3), 295–309.

(10) Tawarmalani, M.; Sahinidis, N. V. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Math. Program.* **2004**, *99* (3), 563–591.

(11) Viswanathan, J.; Grossmann, I. E. A Combined Penalty-Function and Outer-Approximation Method for MINLP Optimization. *Comput. Chem. Eng.* **1990**, *14* (7), 769–782.

(12) Duran, M. A.; Grossmann, I. E. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Math. Program.* **1986**, *36* (3), 307–339.

(13) Fletcher, R.; Leyffer, S. Solving Mixed-Integer Nonlinear Programs by Outer Approximation. *Math. Program.* **1994**, *66* (3), 327–349.

(14) Porn, R.; Westerlund, T. A cutting plane method for minimizing pseudo-convex functions in the mixed integer case. *Comput. Chem. Eng.* **2000**, *24* (12), 2655–2665.

(15) Westerlund, T.; Pettersson, F. An Extended Cutting Plane Method for Solving Convex MINLP Problems. *Comput. Chem. Eng.* **1995**, *19*, S131–S136.

(16) Schweiger, C. A.; Floudas, C. A. Process Synthesis, Design and Control: A Mixed Integer Optimal Control Framework. In *Proceedings of DYCOPS-5 on Dynamics and Control of Process Systems*; Elsevier Science: New York, 1998; pp 189−194.

(17) General Algebraic Modeling System (GAMS); GAMS Development Corporation: Washington, DC; see www.gams.com.

(18) Computational Infrastructure for Operations Research (COIN-OR); COIN-OR Foundation, Inc.; see www.coin-or.org.

(19) Quesada, I.; Grossmann, I. E. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Comput. Chem. Eng.* **1992**, *16*, 937–947.

(20) Bonami, P.; Biegler, L. T.; Conna, A. R.; Cornuejols, G.; Grossmann, I. E.; Laird, C. D.; Lee, J.; Lodi, A.; Margot, F.; Sawaya, N.; Wachter, A. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optim.* **2008**, *5* (2), 186–204.

(21) Zhu, Y.; Kuno, T. A disjunctive cutting-plane-based branch-and-cut algorithm for 0−1 mixed-integer convex nonlinear programs. *Ind. Eng. Chem. Res.* **2006**, *45* (1), 187–196.

(22) Balas, E.; Ceria, S.; Cornuejols, G. A Lift-and-Project Cutting Plane Algorithm for Mixed 0−1 Programs. *Math. Program.* **1993**, *58* (3), 295–324.

(23) Balas, E.; Ceria, S.; Cornuejols, G.; Natraj, N. Gomory cuts revisited. *Oper. Res. Lett.* **1996**, *19* (1), 1–9.

(24) Balas, E.; Perregaard, M. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer gomory cuts for 0−1 programming. *Math. Program.* **2003**, *94* (2−3), 221–245.

(25) Wächter, A.; Biegler, L. T. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* **2006**, *106* (1), 25–57.

(26) *ILOG CPLEX 10.0*; ILOG Company: Sunnyvale, CA, 2006.

(27) Lasdon, L. *LSGRG Version 3.0 Release Notes*; MSIS Department, College of Business Administration, University of Texas: Austin, TX, 2000.

(28) Smith, S.; L., L. Solving Large Sparse Nonlinear Programs Using GRG. *ORSA J. Comput.* **1992**, *4* (1), 2–15.

(29) LINDO Systems, Chicago, IL; see http://www.lindo.com/.

(30) Padberg, M.; Rinaldi, G. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetrical Traveling Salesman Problems. *SIAM Rev.* **1991**, *33* (1), 60–100.

(31) Balas, E.; Ceria, S.; Cornuejols, G. Mixed 0−1 programming by lift-and-project in a branch-and-cut framework. *Manage. Sci.* **1996**, *42* (9), 1229–1246.

(32) CMU-IBM Open Source MINLP Project; see http://egon.cheme.cmu.edu/ibm/page.htm.

(33) Vecchietti, A.; Grossmann, I. E. LOGMIP: A disjunctive 0−1 nonlinear optimizer for process systems models. *Comput. Chem. Eng.* **1997**, *21*, S427–S432.

(34) Sawaya, N. W. Reformulations, relaxations and cutting planes for generalized disjunctive programming. *Ph.D. Thesis*, Carnegie Mellon University, Pittsburgh, PA, 2006.

(35) Turkay, M.; Grossmann, I. E. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Comput. Chem. Eng.* **1996**, *20* (8), 959–978.

(36) Harjunkoski, I.; Westerlund, T.; Porn, R.; Skrifvars, H. Different transformations for solving non-convex trim-loss problems by MINLP. *Eur. J. Oper. Res.* **1998**, *105* (3), 594–603.

(37) Dolan, E. D.; More, J. J. Benchmarking optimization software with performance profiles. *Math. Program.* **2002**, *91* (2), 201–213.

(38) Stubbs, R. A.; Mehrotra, S. A branch-and-cut method for 0−1 mixed convex programming. *Math. Program.* **1999**, *86*, 515–532.

Received for review January 20, 2009
Revised manuscript received August 10, 2009
Accepted August 28, 2009

IE9001074