RC23771 (W0511-023) November 4, 2005
Mathematics

IBM Research Report

An Algorithmic Framework for Convex Mixed Integer Nonlinear Programs

Pierre Bonami[1], Lorenz T. Biegler[1], Andrew R. Conn[2], Gérard Cornuéjols[1,3], Ignacio E. Grossmann[1], Carl D. Laird[1], Jon Lee[2], Andrea Lodi[2,4], François Margot[1], Nicolas Sawaya[1], Andreas Wächter[2]

[1]Carnegie Mellon University
Pittsburgh, PA 15213

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

[3]Faculté des Sciences de Luminy
13288 Marseille, France

[4]University of Bologna
viale Risorgimento 2
40136 Bologna, Italy

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. Ithas been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distributionoutside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g, payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home

An algorithmic framework for convex mixed integer nonlinear programs

Pierre Bonami[1,5], Lorenz T. Biegler[2], Andrew R. Conn[3],
Gérard Cornuéjols[1,4], Ignacio E. Grossmann[2], Carl D. Laird[2,5], Jon Lee[3],
Andrea Lodi[3,6], François Margot[1,7], Nicolas Sawaya[2], Andreas Wächter[3]

October 31, 2005

Abstract

This paper is motivated by the fact that mixed integer nonlinear programming is an important and difficult area for which there is a need for developing new methods and software for solving large-scale problems. Moreover, both fundamental building blocks, namely mixed integer linear programming and nonlinear programming, have seen considerable and steady progress in recent years. Wishing to exploit expertise in these areas as well as on previous work in mixed integer nonlinear programming, this work represents the first step in an ongoing and ambitious project within an open-source environment. COIN-OR is our chosen environment for the development of the optimization software. A class of hybrid algorithms, of which branch and bound and polyhedral outer approximation are the two extreme cases, is proposed and implemented. Computational results that demonstrate the effectiveness of this framework are reported, and a library of mixed integer nonlinear problems that exhibit convex continuous relaxations is made publicly available.

1 Introduction

Solution algorithms for mixed integer nonlinear programs (MINLPs) have become an active area of research [9, 10, 19, 31, 36]. Owing to the steady progress over the years in the development and successful implementation of algorithms for mixed integer linear programs (MILPs) and nonlinear programs (NLPs), it is natural to expect that combining expertise from both fields might yield significant advances. The recent creation of COIN-OR [6] provides a useful vehicle for facilitating the development and dissemination of open-source software for problems in operations research. In particular, COIN-OR contains reusable software components for MILP (e.g., `Cbc`) and NLP (e.g., `Ipopt`). In 2004, researchers at IBM and CMU joined forces to study algorithms for MINLPs and develop associated open-source software, leveraging components already available from COIN-OR. This paper introduces an algorithmic framework for MINLP resulting from this

[1]Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA.
[2]Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA.
[3]IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA.
[4]LIF, Faculté des Sciences de Luminy, 13288 Marseille, France.
Supported in part by NSF grant DMI-0352885 and ONR grant N00014-03-1-0188.
[5]Supported in part by a grant from IBM.
[6]DEIS, University of Bologna, viale Risorgimento 2, 40136 Bologna, Italy.
[7]Supported in part by a grant from IBM and ONR grant N00014-03-1-0188.

collaboration. For MINLPs with convex relaxation, this framework is an exact algorithm, but it can also be applied to nonconvex MINLPs as a heuristic.

Thus far, the main contributions of this effort are:

(i) a new publicly available library of test instances of convex MINLPs;

(ii) a new family of hybrid algorithms of which branch-and-bound (BB) and polyhedral outer approximation (OA) are two extreme (classical) cases;

(iii) a new open-source framework which uses existing software in COIN-OR. In particular `Cbc`, `Cgl`, `Clp` and `Ipopt` are used as building blocks. We expect the package to be available on the COIN-OR site in 2006;

(iv) computational results on publicly available test problems.

We consider the mixed integer nonlinear program

$$P \begin{cases} \min f(x,y) \\ \text{s.t.} \\ g(x,y) \leq 0, \\ x \in X \cap \mathbb{Z}^n, y \in Y, \end{cases}$$

where X and Y are polyhedral subsets of $\mathbb{R}^n$ and $\mathbb{R}^p$ respectively, and X is bounded. The functions $f : X \times Y \to \mathbb{R}$ and $g : X \times Y \to \mathbb{R}^m$ are continuously twice differentiable. When f and g are convex functions, P is said to be *convex*.

Two main ideas have been proposed for solving convex MINLPs. The first one is a branch and bound (BB) approach [14, 19], where lower bounds for subproblems

$$P_{\bar{X}} \begin{cases} \min f(x,y) \\ \text{s.t.} \\ g(x,y) \leq 0, \\ x \in \bar{X} \cap \mathbb{Z}^n, y \in Y, \end{cases}$$

with polyhedral subsets $\bar{X}$ of X, are computed by solving their continuous relaxations $\tilde{P}_{\bar{X}}$; here and elsewhere $\tilde{P}$ denotes the continuous relaxation of the associated problem P, e.g.,

$$\tilde{P}_{\bar{X}} \begin{cases} \min f(x,y) \\ \text{s.t.} \\ g(x,y) \leq 0, \\ x \in \bar{X}, y \in Y. \end{cases}$$

We note that the lower bound in the BB approach can be strengthened with cutting planes like the ones described in [30], which represent a generalization to the nonlinear case of the lift-and-project cuts of [2].

The second approach alternates between solving an MILP and a convex NLP. We know of two different methods that follow this approach: generalized Benders decomposition [11] and Outer Approximation (OA) [9]. The MILP solved in both approaches is obtained from P by replacing the nonlinear functions by polyhedral outer approximations. Calling $(\bar{x}, \bar{y})$ the optimal solution of this MILP, the convex NLP is P with x fixed to $\bar{x}$ (i.e., $P_{\bar{x}}$). We note that a related approach is the extended cutting-plane method [36], which relies on successive solutions of the MILP problem.

Our computational framework uses outer approximations and subproblem relaxations $\tilde{P}_{\bar{X}}$ to compute lower bounds, and $P_{\bar{x}}$ to compute upper bounds in a flexible branch-and-cut scheme. When only subproblem relaxations $\tilde{P}_{\bar{X}}$ are used to compute lower bounds, we have the classical BB algorithm. When outer approximations and $P_{\bar{x}}$ are used alternatingly at the root node, we get the classical OA algorithm.

In Section 2, the OA algorithm is presented in more detail. In Section 3, three related algorithms are presented. Subsection 3.1 presents the NLP Branch-and-Bound framework and details of our implementation (named `I-BB`) based on the interior-point NLP solver `Ipopt`. Subsection 3.2 presents the OA framework and details of our implementation (named `I-OA`) based on `Ipopt` and the LP solver `Clp`. Finally, Subsection 3.3 presents the hybrid algorithm (named `I-Hyb`). Depending on the parameter setting of `I-Hyb`, one obtains either `I-BB` or `I-OA` or anything in between. In Section 4, we present computational results comparing `I-BB`, `I-OA`, and one incarnation of `I-Hyb` with the commercial solvers `SBB` and `Dicopt`.

2 Outer approximation of the MINLP

In the following, we assume that P is convex. The OA algorithm consists of using linearizations of the objective function and the constraints at different points to build a MILP relaxation of the problem.

First note that P can be reformulated as an MINLP, $\hat{P}$, with a linear objective by introducing an extra variable, α, which is minimized subject to the additional constraint $f(x, y) \leq \alpha$. Now consider any point $(\bar{x}, \bar{y}) \in X \times Y$, not necessarily feasible to P. By convexity of f and g, the constraints

$$\nabla f(\bar{x}, \bar{y})^T \begin{pmatrix} x - \bar{x} \\ y - \bar{y} \end{pmatrix} + f(\bar{x}, \bar{y}) \leq \alpha, \tag{1}$$

$$\nabla g(\bar{x}, \bar{y})^T \begin{pmatrix} x - \bar{x} \\ y - \bar{y} \end{pmatrix} + g(\bar{x}, \bar{y}) \leq 0 \tag{2}$$

are then valid for $\hat{P}$.

Therefore, given any set of points $T = \left\{ \left(x^1, y^1\right), \ldots, \left(x^K, y^K\right) \right\}$, we can build a relaxation of P:

$$P^{\mathrm{OA}}(T) \begin{cases} \min \alpha \\ \text{s.t.} \\ \nabla f(x^k, y^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} + f(x^k, y^k) \leq \alpha, \\ \nabla g(x^k, y^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} + g(x^k, y^k) \leq 0, \end{cases} \quad \forall (x^k, y^k) \in T$$
$$x \in X \cap \mathbb{Z}^n,\ y \in Y,\ \alpha \in \mathbb{R}.$$

As is well known, this is equivalent to the problem

$$\begin{cases} \min_{x \in X \cap \mathbb{Z}^n\ y \in Y} \quad \max_k \quad f(x^k, y^k) + \nabla f(x^k, y^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \\ \qquad\qquad\qquad\quad \text{s.t.} \quad g(x^k, y^k) + \nabla g(x^k, y^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \leq 0, \\ \qquad\qquad\qquad\qquad (x^k, y^k) \in T. \end{cases}$$

Theorem 1 below states that if T contains suitable points and if a constraint qualification holds for P, then $P^{\mathrm{OA}}(T)$ and P are equivalent, in the sense that they have the same optimal

value, and that the optimal solution $(\bar{x}, \bar{y})$ of P corresponds to an optimal solution $(\bar{\alpha}, \bar{x}, \bar{y})$ of $P^{\mathrm{OA}}(T)$ with $\bar{\alpha} = f(\bar{x}, \bar{y})$.

Theorem 1. *For any $\bar{x} \in X \cap \mathbb{Z}^n$, if the problem*

$$P_{\bar{x}} \begin{cases} \min f(\bar{x}, y) \\ \text{s.t.} \\ g(\bar{x}, y) \leq 0, \\ y \in Y \end{cases}$$

is feasible, then let $\bar{y}$ be one optimal solution. On the other hand, if $P_{\bar{x}}$ is infeasible, then $\bar{y}$ is defined as one optimal solution to the following feasibility problem:

$$P^{\mathrm{F}}_{\bar{x}} \begin{cases} \min \sum_{i=1}^{m} u \\ \text{s.t.} \\ g(\bar{x}, y) - u \leq 0, \\ y \in Y,\ u \in \mathbb{R}^m_+. \end{cases}$$

Let $\hat{T}$ be the set of all such pairs $(\bar{x}, \bar{y})$.

Assuming that f and g are convex, continuously twice differentiable, and that a constraint qualification holds at every optimum of $P_{\bar{x}}$ and $P^{\mathrm{F}}_{\bar{x}}$, then P and $P^{\mathrm{OA}}(\hat{T})$ have the same optimal value.

Theorem 1 is weaker than Theorem 1 in [10], where it is incorrectly stated that (under the conditions of Theorem 1) (x^*, y^*) is optimal for P if and only if it is optimal for $P^{\mathrm{OA}}(T)$.

Indeed, although all optimal solutions of P are optimal solutions of $P^{\mathrm{OA}}(T)$, the converse is not true as the following example shows.

Example 1 : Consider the following MINLP where we optimize over a ball in $\mathbb{R}^3$

$$\begin{cases} \min z \\ \text{s.t.} \\ (x - \frac{1}{2})^2 + y^2 + z^2 \leq 1, \\ x \in \mathbb{Z} \cap [-1, 2],\ y \in \mathbb{R},\ z \in \mathbb{R}. \end{cases}$$

This problem has an optimal value of $-\frac{\sqrt{3}}{2}$. The MILP obtained by applying Theorem 1 is the

following

$$\begin{cases} \min \alpha \\ \text{s.t.} \\ z - \alpha \leq 0, & \text{(constraint (1))} \\ x - \sqrt{3}z \leq 1 + 3/2, & \text{((2) taken for the optimal solution of } P_{(1)}\ (x,y,z) = (1,0,-\frac{\sqrt{3}}{2})) \\ -x - \sqrt{3}z \leq 3/2, & \text{((2) taken for the optimal solution of } P_{(0)}\ (x,y,z) = (0,0,-\frac{\sqrt{3}}{2})) \\ x \geq \dfrac{-7}{12}, & \text{((2) taken for the optimal solution of } P^F_{(-1)}\ (x,y,z) = (-1,0,0)) \\ x \leq \dfrac{19}{12}, & \text{((2) taken for the optimal solution of } P^F_{(2)}\ (x,y,z) = (2,0,0)) \\ x \in \mathbb{Z},\ y \in \mathbb{R},\ z \in \mathbb{R}. \end{cases}$$

It can be seen that in the linearized problem variable y is unconstrained. Therefore for any $\lambda \in \mathbb{R}$, the point $(1, \lambda, -\frac{\sqrt{3}}{2})$ is feasible and also optimal for the MILP $P^{OA}(\hat{T})$, but it is infeasible for the MINLP whenever $\lambda \neq 0$. ■

2.1 Outer approximation algorithm

Originally proposed in [9], the relaxation $P^{\mathrm{OA}}(T)$ induces a natural iterative algorithm for solving P. We present here briefly a slightly modified version introduced in [10]. The algorithm starts with $T = \{(x^0, y^0)\}$, where (x^0, y^0) can be either a feasible solution to P or to its continuous relaxation P_X. Then each iteration starts by solving $P^{\mathrm{OA}}(T)$ to find a point $(\alpha^k, x^k, \hat{y})$ and a lower bound α^k on the optimal value of P. The problem P_{x^k} defined above is then solved. If P_{x^k} is feasible, then its optimal solution y^k associated with x^k gives a feasible solution and an upper bound for P and (x^k, y^k) is added to T to strengthen the mixed-integer linear relaxation. If P_{x^k} is infeasible, let y^k be the solution of the feasibility problem $P^{\mathrm{F}}_{x^k}$, and add (x^k, y^k) to T. As shown by Theorem 2 in [10], the algorithm converges to an optimal solution of P in a finite number of iterations provided that assumptions on convexity, differentiability and constraint qualifications of Theorem 1 hold.

The algorithm is described in Figure 1.

2.2 Branch-and-cut based outer approximation

Quesada and Grossmann [24] proposed an algorithmic scheme that combines the use of linear and nonlinear programming in an original branch-and-cut scheme. The motivation of the method proposed by them is to improve the outer-approximation scheme presented in Section 2.1 by integrating the construction of the outer approximation of P into a single tree search. In this way the sequential solution of several MILPs is avoided. Instead one single tree search is performed, during which nonlinear programs are solved and used to progressively tighten the MILP relaxation.

As in the OA algorithm, a mixed integer linear relaxation $P^{\mathrm{OA}}(T)$ is used. But, instead of solving to optimality the successive approximations given by the $P^{\mathrm{OA}}(T)$ relaxations, we perform a branch-and-cut procedure, where the linear outer approximation is updated at selected nodes of the search tree.

$z^U := +\infty$;
$z^L := -\infty$;
$(x^0, y^0) :=$ optimal solution of $\tilde{P}$;
$T := \{(x^0, y^0)\}$;
$k := 1$; Choose a convergence tolerance ϵ
while $z^U - z^L > \epsilon$ and $P^{\mathrm{OA}}(T)$ is feasible **do**
    Let $(\hat{\alpha}, \hat{x}, \hat{y})$ be the optimal solution of the updated$P^{\mathrm{OA}}(T)$;
    $z^L := \hat{\alpha}$;
    if $P_{\hat{x}}$ is feasible;
      then Let $x^k := \hat{x}$ and y^k be the optimal solution to $P_{\hat{x}}$;
          $z^U := \min(z^U, f(x^k, y^k))$;
       else Let $x^k := \hat{x}$ and y^k be the optimal solution to $P^{\mathrm{F}}_{\hat{x}}$;
    fi
    $T := T \cup \{(x^k, y^k)\}$;
    $k := k + 1$;
od

Figure 1: Outer approximation algorithm.

The branch-and-cut algorithm of Quesada and Grossmann is based on two problems, related to $P^{\mathrm{OA}}(T)$ and P_x presented in the previous section. The outer approximation of a subproblem corresponding to $\bar{X} \subseteq X$

$$P^{\mathrm{OA}}_{\bar{X}}(T) \begin{cases} \min \alpha \\ \text{s.t.} \\ \nabla f(x^k, y^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} + f(x^k, y^k) \leq \alpha \\ \nabla g(x^k, y^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} + g(x^k, y^k) \leq 0 \\ x \in \bar{X} \cap \mathbb{Z}^n,\ y \in Y,\ \alpha \in \mathbb{R} \end{cases} \quad \forall (x^k, y^k) \in T$$

is used as a relaxation which gives lower bounds, while $\tilde{P}_{\bar{X}}$ gives feasible solutions, upper bounds and new outer approximation constraints.

Let i be a node of the search tree, and let $X^i \subseteq X$ be the modified polyhedral feasibility set for x at that node (with modified bounds). The continuous relaxation $\tilde{P}^{\mathrm{OA}}_{X^i}(T)$ is solved, obtaining a solution (x^*, y^*) and a lower bound on the optimal value of the subtree rooted at i. If x^* satisfies the integrality requirements, a feasible solution for P may exist at that node, and it can be found by solving P_{x^*}. Let $\bar{y}$ be an optimal solution to P_{x^*}. If $(x^*, \bar{y})$ is feasible for P, the upper bound is updated. Moreover, we update T to $T \cup \{(x^*, \bar{y})\}$.

As shown in [24] the algorithm converges to an optimal solution in finite time provided that assumptions on convexity, continuity, differentiability and constraint qualification of Theorem 1 hold.

The complete algorithm is described in Figure 2.

In the algorithm of Quesada and Grossmann as described above, the NLP subproblem P_{x^*} is solved only when the optimal solution (x^*, y^*) of $\tilde{P}^{\mathrm{OA}}_{X^i}(T)$ is integer feasible. This is the least we can do in terms of solving NLPs, to guarantee the convergence of the algorithm by application of

$z^U := +\infty$;
$(x^0, y^0) :=$ optimal solution of $\tilde{P}$;
$T := \{(x^0, y^0)\}$;
$X^0 := X$;
Initialize list of active nodes: $\mathcal{L} := \{0\}$;
$i := 0$;
while $\mathcal{L} \neq \emptyset$ **do**
    Select a node k in $\mathcal{L}$;
    Let (α^k, x^k, y^k) be the optimal solution of $\tilde{P}^{\mathrm{OA}}_{X^k}(T)$;
    while x^k is integer and $\alpha^k < z^U$ **do**
        if P_{x^k} is feasible
          then let $\bar{y}$ be its optimal solution;
           else let $\bar{y}$ be the optimal solution of $P^{\mathrm{F}}_{x^k}$
        fi
        if $(x^k, \bar{y})$ is feasible for P
          then $z^U := \min(z^U, f(x^k, \bar{y}))$;
        fi
        $T := T \cup \{(x^k, \bar{y})\}$;
        Let (α^k, x^k, y^k) be the optimal solution of updated $\tilde{P}^{\mathrm{OA}}_{X^k}(T)$;
    od
    if $\alpha^k < z^U$ *(branch)*
      then Select a variable x_j with a fractional x^k_j for branching;
        $X^{i+1} := X^k \cap \{x \in \mathbb{R}^n : x_j \leq \lfloor x^k_j \rfloor\}$;
        $X^{i+2} := X^k \cap \{x \in \mathbb{R}^n : x_j \geq \lceil x^k_j \rceil\}$;
        $\mathcal{L} := \mathcal{L} \cup \{i+1, i+2\} \setminus \{k\}$;
        $i := i + 2$;
       else *(fathom the node by bounds)*
        $\mathcal{L} := \mathcal{L} \setminus \{k\}$;
    fi
od

Figure 2: Outer approximation based branch-and-cut.

Theorem 1. However, it is entirely possible to solve P_{X^i} at additional nodes in order to reduce the size of the branch-and-bound tree. In Section 3.3, we present two ways of enhancing this branch-and-bound scheme which we implemented in our hybrid algorithm.

3 Our algorithmic framework

We implemented three different algorithms:

- `I-BB` : the simple nonlinear programming based BB as presented in Section 1;
- `I-OA` : the OA algorithm as presented in Section 2.1;
- `I-Hyb`: an enhanced version of the hybrid procedure presented in Section 2.2.

The three algorithms are implemented using existing software components from the COIN-OR open source library as building blocks. To solve the continuous NLPs we use the interior point solver `Ipopt 3.0` [35]. The various branch-and-bound techniques are based on the branch-and-cut module `Cbc` and the cut-generation library `Cgl`, and the LP subproblems are solved using `Clp`.

3.1 Algorithm `I-BB`

The BB algorithm was implemented by modifying `Cbc` in order to replace the solution of LPs at each node of the tree by the solution of NLPs. BB is an efficient technique for solving MILPs, as the reoptimization of an LP after small modifications can be done efficiently using hot-start techniques in simplex-based algorithms. (The reader is refereed to [17, 23, 37] for terminology and background information related to branch-and-cut for solving MILPs.) The same applies to NLPs, to some extent, when they are solved using active set algorithms [19]. Nevertheless, we chose the interior point NLP solver `Ipopt` as our solver, as a suitable open source active-set NLP solver was unavailable. Note that using a different NLP solver is possible with minor modifications to the code.

The BB algorithm follows a simple scheme; no cutting-plane method is used, no strong branching is performed, and no heuristic methods are implemented. The branching variable selection strategy uses pseudo-costs initialized with the *average known* pseudo-cost as defined in [20].

To try to reduce the time spent solving NLPs, at each node `Ipopt` is warm-started by using the optimal primal and dual solution of the direct parent node as a starting point. In addition, the initial modification of the starting point, aiming to move it sufficiently away from bounds, is done less strongly for a warm start than for a cold start. This leads to a reduction in the average run time for solving the NLPs.

`Ipopt` was used with the adaptive barrier parameter strategy [22]. We also added a heuristic to switch sooner to the feasibility restoration phase if little progress was made at the beginning of the optimization, in order to improve infeasibility detection.

3.2 Algorithm `I-OA`

The OA algorithm is implemented using `Ipopt` to solve the NLPs and `Cbc` to solve the MILPs. It follows exactly the algorithm described in Figure 1. The default strategies of `Cbc` are used, namely:

- for branching variable selection at a node strong branching is performed on the five most integer infeasible variables (in case of ties variables are chosen in lexicographic order) in the current optimal solution of the relaxed problem,
- various cutting planes procedures (mixed integer Gomory cuts, probing cuts, mixed integer rounding cuts, clique cuts, cover cuts and flow cover cuts [6]) are applied but only at the root node.

3.3 Algorithm `I-Hyb`

This algorithm is an enhanced version of the branch-and-cut procedure presented in Section 2.2. Its implementation is based on `Cbc`. One enhancement is that more NLPs are solved in order to reduce the size of the tree. This is done in two ways: by solving the NLP relaxation $\tilde{P}_{X_k}$ at additional nodes of the tree and by performing local enumerations at nodes of the tree. In the limiting cases, the first way reduces the algorithm to a classical NLP BB, while the second one reduces the algorithm to the classical OA algorithm. Of course, both alternatives can be combined within a hybrid algorithm, an approach investigated in this paper, apparently for the first time. A further enhancement is that our hybrid scheme uses a number of MILP techniques available in `Cbc`. We describe the enhancements in more detail in the following subsections.

3.3.1 Solving NLP relaxations at some nodes

Consider a node k. If $\tilde{P}_{X^k}$, the relaxation of the corresponding subproblem, is infeasible, then node k can be fathomed. Otherwise, let $(\bar{x}, \bar{y})$ be an optimal solution of $\tilde{P}_{X^k}$. If $\bar{x}$ is integer, then we obtain a new feasible solution $(\bar{x}, \bar{y})$ for P, which gives an upper bound for the problem, and the node can be fathomed. Otherwise, we add $(\bar{x}, \bar{y})$ to T.

Solving $\tilde{P}_{X^k}$ allows us to strengthen the bounds and therefore typically leads to a smaller enumeration tree. Note also that this scheme improves the outer approximation higher in the tree, in contrast to the basic algorithm where the outer approximation is strengthened only at nodes k where the optimal solution $(\bar{x}, \bar{y})$ to $\tilde{P}^{\mathrm{OA}}_{X^k}$ has $\bar{x}$ integer.

If the subproblem relaxation $\tilde{P}_{X^k}$ is solved at each node of the tree, then the algorithm reduces to a NLP BB algorithm. This is unlikely to be the optimal strategy for all problem classes. In the computational experiments presented in Section 4, the subproblem relaxations are solved every $L := 10$ nodes.

3.3.2 Performing local searches at nodes

The enhancement described above makes the algorithm closer to a pure BB algorithm. An alternative approach is to perform some outer-approximation iterations at selected nodes of the tree.

At any node k of the tree, we may perform a truncated MILP BB procedure to try to find a feasible solution (x^*, y^*) for $P^{\mathrm{OA}}_{X^k}(T)$. If such a solution is found, then we solve P_{x^*} (or $P^{\mathrm{F}}_{x^*}$, if P_{x^*} is infeasible), obtaining $(x^*, \bar{y})$. As before, the solution $(x^*, \bar{y})$ is added to T, and the upper bound and best incumbent solution are possibly updated. We can then repeat this process until the MINLP problem corresponding to node k is solved.

In our experiments, we perform such a local search only at the root node with an overall limit on the time spent in solving such MILP's of $\kappa := 30$ seconds. The local searches are performed by using `Cbc` with the settings used by `I-OA` described in Section 3.3.

Of course, if the value for the time limit κ is set to infinity the algorithm then reduces to the usual OA algorithm.

3.3.3 Integration of MILP techniques

`Cbc` is an elaborate code for MILP. Using it as the underlying framework for our hybrid procedure allows us to take advantage of the advanced MILP techniques already implemented in or utilized by `Cbc`. `Cbc` uses several families of cuts from `Cgl` to tighten the OA relaxations. To produce the results in Section 4 we use mixed integer Gomory cuts, probing cuts, mixed integer rounding cuts, and cover cuts. The different cut generators are called at the root node and in the tree search according to the dynamic strategy implemented in `Cbc`.

For the branching variable selection strategy, we use the implementation of reliability branching [1] available in `Cbc` with a reliability parameter of 8 and a limit of 20 variables on which strong branching is performed and a limit of 100 simplex iterations in the strong branching. If a feasible solution for $P^{\mathrm{OA}}(T)$ is found during strong branching, an NLP is solved to verify if it corresponds to a feasible solution for P and additional outer-approximation constraints are added in the same way as it is done when nodes of the tree are processed.

The implementation of the hybrid algorithm follows the branch-and-cut procedure described in Figure 3.

4 Computational results

The problems used in the computational experiments were gathered from different sources, and feature applications from operations research and chemical engineering. We present a brief description of every class of problems below, and highlight some of their defining characteristics. Note that those whose name ends with an "M" or an "H" refer to problems that were originally formulated in generalized disjunctive form [25]. The MINLP version of the problems was obtained using either a "big-M" transformation (end in "M"), or a "convex hull" transformation (end in "H") [13].

The `BatchS` problems are multi-product batch plant design problems with multiple units in parallel and intermediate storage tanks [26, 33]. These problems consist of determining the volume of the equipment, the number of units in parallel, and the volume and location of the intermediate storage tanks. The nonlinearities in this set of problems stem from exactly one constraint that contains an exponential term. The relative integrality gap (defined as $\frac{optval(P)-optval(\tilde{P})}{|optval(P)|}$ where $optval()$ returns the optimal value of the problem) of the big-M version of this set of problems is about 15% on average.

The `CLay` problems are constrained layout problems [27, 28], where non-overlapping units represented by rectangles must be placed within the confines of certain designated areas formulated as circular nonlinear constraints, such that the cost of connecting these units is minimized. The nonlinearities in this set of problems are all quadratic and correspond to Euclidean-distance constraints, and the integrality gap for all instances presented is equal to 100%. Note that these problem are intentionally poorly modeled in order to have a large integrality gap and no feasible solution near the optimal solution of the continuous relaxation.

The `FLay` problems concern farm land layout [27, 28], where one would like to determine the optimal length and width of a number of rectangular patches of land with fixed area, such that the perimeter of the set of patches is minimized. The nonlinearities in this set of problems stem from a set of hyperbolic constraints, and the integrality gap is, on average, about 44%.

$z^U := +\infty$;
$(x^0, y^0) :=$ optimal solution of $\tilde{P}$; $T := \{(x^0, y^0)\}$; $X^0 := X$;
while Local search time limit κ is not exceeded **do**
  Solve $P^{\mathrm{OA}}(T)$(without exceeding resource limits);
  if a solution (x^*, y^*) of $P^{\mathrm{OA}}(T)$ is found
    then
      Let $\bar{y}$ be the optimal solution of P_{x^*}
        (or of $P^{\mathrm{F}}_{x^*}$ if P_{x^*} is infeasible);
      if $(x^*, \bar{y})$ is feasible for P
        then $z^U := \min(z^U, f(x^*, \bar{y}))$;
      fi
      $T := T \cup \{(x^*, \bar{y})\}$;
  fi
od
Initialize list of active nodes: $\mathcal{L} := \{0\}$; $i := 0$; $l := 0$; $L := 10$
while $\mathcal{L} \neq \emptyset$ **do**
  Select a node k in $\mathcal{L}$;
  if $l \equiv 0 \pmod{L}$
    then *(solve the MINLP relaxation at the node)*
      Let $(\bar{x}, \bar{y})$ be the optimal solution of $\tilde{P}_{X^k}$;
      if $\bar{x}$ is integer
        then $z^U := \min(z^U, f(\bar{x}, \bar{y}))$;
      fi
      $T := T \cup \{(\bar{x}, \bar{y})\}$;
  fi
  Apply MILP cuts to $\tilde{P}^{\mathrm{OA}}_{X^k}(T)$, let (α^k, x^k, y^k) be the resulting optimal solution;
  while x^k is integer and $\alpha^k < z^U$ **do**
    if P_{x^k} is feasible
      then let $\bar{y}$ be its optimal solution;
      else let $\bar{y}$ be the optimal solution of $P^{\mathrm{F}}_{x^k}$
    fi
    if $(x^k, \bar{y})$ is feasible for P
      then $z^U := \min(z^U, f(x^k, \bar{y}))$;
    fi
    $T := T \cup \{(x^k, \bar{y})\}$;
    Let (α^k, x^k, y^k) be the optimal solution of updated $\tilde{P}^{\mathrm{OA}}_{X^k}(T)$;
  od
  if $\alpha^k < z^U$ *(branch)*
    then Select a variable x_j with x^k_jfractional;
      $X^{i+1} := X^k \cap \{x \in \mathbb{R}^n : x_j \leq \lfloor x^k_j \rfloor\}$;
      $X^{i+2} := X^k \cap \{x \in \mathbb{R}^n : x_j \geq \lceil x^k_j \rceil\}$;
      $\mathcal{L} := \mathcal{L} \cup \{i+1, i+2\} \setminus \{k\}$;
      $i := i + 2$; $l := l + 1$;
    else *(fathom the node by bounds)*
      $\mathcal{L} := \mathcal{L} \setminus \{k\}$;
  fi
od

Figure 3: Hybrid Algorithm

The Fo7_2 problem is a block layout design with unequal areas [5], and is concerned with finding the most efficient arrangement of a given number of departments with unequal area requirements within a facility. This problem has the same defining characteristics as the farm layout problems described above

The Water problems are large inverse problems for the determination of contamination sources in municipal water networks. The constraints are linear and come from discretized dynamic models of the network water quality model. The objective is the least squares error between calculated and measured network concentrations with a regularization term to force a unique solution. Integer variables are added to restrict the allowable contamination scenarios, giving the final form as a mixed integer quadratic program (MIQPs) [18].

The RSyn problems concern retrofit-synthesis problems [27, 28], in which one would like to simultaneously redesign part of an existing plant and synthesize (from scratch) part of a new one. Specifically, one is interested in determining whether certain units should be included in the design of the new plant, and whether certain modifications such as improvements in yield, capacity and energy reduction should be performed on the existing plant. In addition it is required that economic potential is maximized given a certain time horizon and limited capital investments. The nonlinearities in this set of problems stem from the synthesis portion of the model, and correspond to logarithmic functions. The integrality gap for the convex hull version of these problems tends to be small and of the order of about 1%.

The SLay problems are Safety Layout problems [27, 28], 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 pre-defined "safety point" is minimized. This 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 integrality gap is approximately equal to 6% on average. Furthermore, this class of problems is known to be symmetric, a feature shared with the other layout problems Clay and FLay.

The Syn problems are Synthesis problems [9, 32], and correspond to the synthesis portions of the RSyn class described above. As such, they have similar characteristics to the latter set of problems, although it should be noted that the big-M version of this class tends to have very poor relaxations resulting in integrality gaps of over 600%.

The cutting stock problems, trimloss, [15], are where 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 over production, is minimized. The nonlinearities in this set of problems arise from square root transformations that were used to convexify a set of bilinear constraints. The integrality gap is about 75% on average.

All these problems are available on the web [21] in Ampl and Gams formats, except the Water problems available only in Ampl format. (This implies that we are unable to run the Water problems with SBB or Dicopt. We kept these problems in the test set, since they have a large number of continuous variables, a feature absent from other problems.) These 41 problems were selected from a library of more than 150 convex problems that we collected [27]. We selected problems that were solved in less than 3 hours by an earlier implementation of the hybrid algorithm described in the previous section. We rejected problems that were solved in less than 30 seconds. In addition, for each class of problems, we added the smallest problem in the class that was not solved in 3 hours. Improvements in the hybrid algorithm mean that most problems are now solved within the time limit by the current version of the hybrid. Characteristics of the instances are reported in Table 1. The optimal value is known for all problems, except trimloss5.

We compare three types of algorithms: Branch-and-bound algorithms solving NLPs at the nodes of the tree (*NLP BB* for short), outer approximations algorithms as described in Section 2

Problem name	var	integer	constr	nnz	opt. sol value	continuous relaxation
BatchS101006M	278	129	1019	79	769440.42	734943
BatchS121208M	406	203	1511	95	1241125.51	1202360
BatchS151208M	445	203	1781	98	1543472.39	1499910
BatchS201210M	558	251	2327	103	2296535.15	2255300
CLay0203H	90	18	132	30	41573.30	0.00
CLay0204H	164	32	234	40	6545	0.00
CLay0205M	80	50	135	10	8092.5	0.00
CLay0205H	260	50	365	50	8092.5	0.00
CLay0303M	33	21	66	6	26669.10	0.00
CLay0303H	99	21	150	45	26669.13	0.00
CLay0305M	85	55	155	10	8092.5	0.00
FLay04H	234	24	282	4	54.40	30.98
FLay05M	62	40	65	5	64.49	34.64
FLay05H	382	40	465	5	64.49	34.64
Fo7_2	114	42	211	14	17.74	0.00
Water0202	106711	7	107209	4017	125.19	60.28
Water0303	107222	14	108217	4521	207.98	75.52
Water0202R	384	14	556	17205	424.54	0.00
RSyn0810M03H	1185	252	1935	90	-2722.44	-2797.66
RSyn0815M03H	1347	282	2217	156	-2827.92	-2916.02
RSyn0820M03H	1467	312	2448	201	-2028.81	-2102.39
RSyn0820M04H	1956	416	3528	268	-2450.77	-2509.27
RSyn0830M03H	1758	372	2934	291	-1543.05	-1589.61
RSyn0840M04H	2344	496	4236	388	-2529.07	-2579.75
RSyn0840M03H	2040	432	3447	402	-2742.64	-2806.52
RSyn0840M04H	2720	576	4980	536	-2564.50	-2618.98
SLay10M	290	180	405	20	129579.88	119090
SLay07H	476	84	609	14	64748.82	61757.1
SLay08H	632	112	812	16	84960.21	80754.9
Slay09M	234	144	324	18	107805.75	103126
SLay09H	810	144	1044	18	107805.75	103126
Syn20M04M	420	160	1052	56	-3532.74	-9864.89
Syn30M03M	480	180	1041	60	-654.15	-4535.1
Syn30M04M	640	240	1568	80	-865.72	-6171.15
Syn40M02M	420	160	812	56	-388.77	-4555.35
Syn40M03M	630	240	1398	84	-395.14	-6190.65
Syn40M03H	1146	240	1998	402	-395.14	-417.45
Syn40M04M	840	320	2104	112	-901.75	-9168.63
Syn40M04H	1528	320	2904	536	-901.75	-920.15
trimloss4	105	85	64	36	8.3	1.70
trimloss5	161	131	90	55	$\leq$11.2	1.17

Table 1: Test set statistics. Number of variables, number of integer variables, number of constraints, number of non zero entries in the Hessian of the Lagrangian, optimal solution value and value of the continuous relaxation are listed. (The optimum for trimloss5 is not known.)

(*OA* for short) and one hybrid algorithm as described in Section 3.3. We compare our implementation `I-BB` of an NLP BB with the commercial software `SBB` [29] (version Level 009) using the CONOPT NLP solver (version 3.14g-016-054), and our implementation `I-OA` of an OA algorithm with the commercial software `Dicopt` [7, 34] (version 2x-c), using CPLEX (version 9.0) for the MILP subproblems and CONOPT (same version as above) for the NLPs. Finally, we also list results obtained with our hybrid algorithm `I-Hyb`. The machine used in the tests is an IBM IntellistationZ Pro with an Intel Xeon 3.2GHz CPU, 2 gigabytes of RAM and running Linux Fedora Core 3.

Note that the computing time we use for the results obtained with `SBB` are the CPU times reported by the software. This time is much smaller than the wall clock time on large problems, as `SBB` writes (sometimes huge) files during the computation, and the time for reading/writing files is not included in the reported CPU times. For example, on problem Syn40M02M which hits the three hour time limit, the actual user time is almost 13 hours and the real time (including system time) is more than 31 hours.

Figure 4 is a performance plot [8] of the five algorithms. The curve plotted for algorithm A gives the proportion of problems that are solved within factor p of the time required by the best algorithm. More precisely, if $t^*(P)$ is the smallest time needed by one of the five algorithms to solve problem P, then P is solved within a factor p by any algorithm requiring at most time $p\ t^*(P)$. For $p = 1$, the plotted point for algorithm A represents the proportion of problems for which A is fastest. For p very large, the plotted point is the proportion of problems solved by the algorithm in less than 3 hours. (Note that the three `Water` problems are not used for this plot.) It is clear from the plot that `I-Hyb` dominates `I-BB` and `I-OA`. It also dominates `SBB` except for very small values of p. The number of problems solved in 3 hours by `I-Hyb` is larger than the comparable number for the other algorithms, indicating that `I-Hyb` is overall the most robust algorithm. `Dicopt` solves more problems very early, and that the performance of `I-Hyb` becomes competitive for $p = 28$. This value for p might seems quite high, but it is obtained due to a number of problems that are solved in a few seconds with `Dicopt`, but require much more time with `I-Hyb`. Indeed, in a performance plot based on solution time differences instead of ratios, `I-Hyb` is the best performer as soon as running for 45 seconds beyond the best solution time is allowed.

Table 2 presents the running times of the five algorithms. Failures are listed as `***` and are caused by numerical difficulties in `Ipopt`, except on `SLay07H` for `I-OA` where the failure is caused by numerical problems in the LP solver and on `RSyn0815M03H` for `Dicopt` where the failure occurs with `CONOPT`.

Note that `Dicopt` seems particularly efficient on the `BatchS`, `RSyn` and `Syn` problems, while `SBB` dominates on the `CLay` and `FLay` problems. For all the `SLay` problems except one, either `SBB` or `I-BB` works best. For `trimloss4`, only `I-Hyb` is able to solve it in 3 hours. An interesting question is to determine which problems are well suited for OA algorithms and which ones should be attacked with an NLP BB approach. We suspect that the quality of the linear relaxation obtained by the OA algorithms is the driving criterion: When this approximation is good, the solution of the MILP is close to the solution of the MINLP and OA works well. On the other hand, when the approximation is poor, solving the MILP does not help to find good feasible solutions and helps only moderately in improving the LP relaxation.

Table 3 compares the `I-BB`, `I-Hyb` and `I-OA` algorithms. As noted before, `I-Hyb` is almost always faster than `I-BB`, the only notable exceptions being the `CLay`, `FLay` and `SLay` problems where none of the two dominates clearly the other. Comparing `I-Hyb` with `I-OA`, they have comparable performances on the `CLay`, `RSyn` and `Water` problems that can be solved in less than 3 minutes. On the other classes, `I-Hyb` dominates `I-OA`.

Comparing the number of nodes of `I-BB` and `I-Hyb`, the latter usually has a smaller enumer-

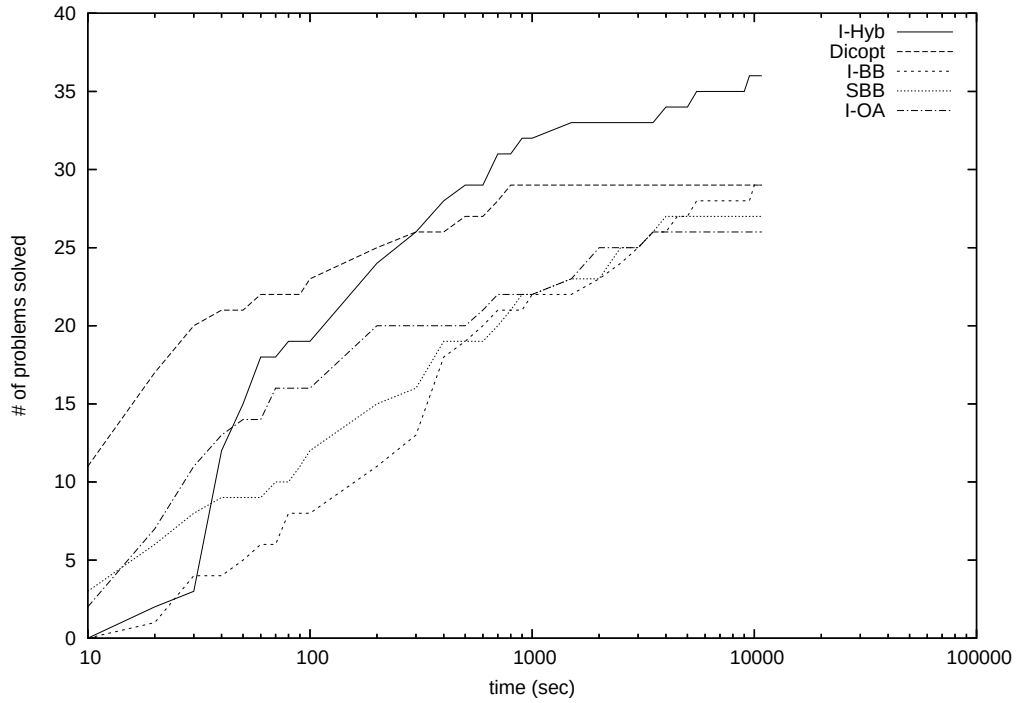

Figure 4: Relative performance of the five algorithms.

ation tree, illustrating the benefit of having the OA approximation available for bounding and branching decisions.

The number of NLPs solved by `I-Hyb` is much larger than for `I-OA`. The percentage of the time spent solved NLPs in `I-OA` is usually well below 5% of the total time, while this is sometimes the dominant part for `I-Hyb`.

Table 4 compares the two NLP BB algorithms. A rough estimation is that the number of nodes is comparable for the two algorithms. It is worth noting that the time spent per node for `SBB` is on the order of 0.27 second, while it is about 0.21 second for `I-BB` (average taken over all problems solved in less than 3 hours by both algorithms). Since in those two algorithm the bulk of the time is spent solving NLPs, this indicates that the warm-starting advantage of the active set solver used by `SBB` over the interior point solver used by I-BB does not confer any advantage in terms of computing time in our test set.

Table 5 compares the two OA algorithms. Usually, `I-OA` requires fewer iterations than `Dicopt`, but this does not translate to a large advantage in running times. This is explained by the average time required to solve MILPs: `Dicopt` requires about 2.1 seconds per iteration while `I-OA` requires 81.9 seconds (average taken over problems solved in less than 3 hours by both algorithms). At least two factors play a role in this large difference: First, `Dicopt` uses `CPLEX` to solve MILPs, while `I-OA` uses `Cbc`. On average `CPLEX` is faster than `Cbc` on benchmark problems (such as

problems in MIPLIB [4]), but not by a factor of 40. The difference is that these benchmark problems are relatively difficult, often requiring more than one hour to be solved. It turns out that many of the MILPs that are solved in our experiments are fairly easy and that the relative performance of `Cbc` on these easy problems is comparatively worse than on harder problems. Second, when the OA algorithm solves an NLP that has more than one optimal solution, the solution obtained by an interior point solver will tend to be on the center of the optimal "face", while the solution obtained by an active set solver will lie at an "extreme point" of the "face". The OA constraints (1) and (2) will be different and thus the difficulty of the MILPs that have to be solved might be different. To illustrate this point we run `I-OA` using CPLEX as an LP solver on the problem `Syn40M03M`. The problem is still solved in 5 iterations but in 267 seconds (instead of 1815 seconds with `Cbc`) which is still much larger than the 1.2 seconds needed by `Dicopt`.

Geometric considerations lead us to believe that the MILPs to be solved when using `I-OA` are, in general, harder than those for `Dicopt`. On the other hand, these MILPs give sometimes a stronger lower bound on the value of the optimal solution. The percentage of the gap between the optimal value and the linear relaxation closed by solving the first MILP in `I-OA` is displayed in Table 5. On our test problems, the corresponding gap for `Dicopt` is similar except for the `FLay` and `Fo7_2` problems where a difference of more than 20% can be observed. It is interesting to note that these problems are the ones having nonlinearities coming from hyperbolic constraints.

We also note that the number of iterations is strongly correlated with the gap reduction obtained in the first master iteration. Both algorithms require a similar number of master iterations, with the notable exception of the `CLay` problems, where `Dicopt` requires a number of iterations much larger than `I-OA`. This might be explained by the fact that these problems have symmetries and thus the MILPs may have a lot of optimal integer solutions of which only a few are NLP feasible.

References

[1] T. Achterberg, T. Koch and A. Martin. Branching rules revisited. *Operations Research Letters*, 3:42–54, 2005.

[2] E. Balas, S. Ceria and G. Cornuejols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.

[3] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere and O. Vincent. Experiments in mixed-integer programming. *Mathematical Programming*, 1:76–94, 1971.

[4] R.E. Bixby, S. Ceria, C.M. McZeal, M.W.P. Savelsbergh, MIPLIB 3.0, `http://www.caam.rice.edu/~bixby/miplib/miplib.html`.

[5] I. Castillo, J. Westerlund, S. Emet and T. Westerlund. Optimization of Block Layout Design Problems with Unequal Areas: a Comparison of MILP and MINLP Optimization Methods. *Computers & Chemical Engineering*, 2005 (to appear).

[6] COIN-OR. `www.coin-or.org`

[7] `Dicopt`. `http://www.gams.com/solvers/dicopt/main.htm`

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

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

[10] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[11] A.M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

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

[13] I.E. Grossmann and S. Lee. Generalized Disjunctive Programming: Nonlinear Convex Hull Relaxation and Algorithms. *Computational Optimization and Applications*, 26:83–100, 2003

[14] O.K. Gupta and V. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[15] I. Harjunkoski, T. Westerlund, R. Pörn and H. Skrifvars. Different transformations for solving non–convex trim loss problems by MINLP. *European Journal of Operational Research*, 105:594–603, 1998.

[16] V. Jain and I.E. Grossmann. Cyclic scheduling of continuous parallel units with decaying performance. *AIChE Journal*, 44:1623–1636, 1998.

[17] M. Jünger and D. Naddef, eds. *Computational Combinatorial Optimization*. Lecture Notes in Computer Science 2241, Springer, 2001.

[18] C.D. Laird, L.T. Biegler and B.G. van Bloemen Waanders. A Mixed Integer Approach for Obtaining Unique Solutions in Source Inversion of Drinking Water Networks. *ASCE Journal of Water Resource Management and Planning*, 2006 (to appear).

[19] S. Leyffer. Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization & Applications*, 18:295–309, 2001.

[20] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal of Computing* 11:173–187, 1999.

[21] `http://egon.cheme.cmu.edu/ibm/page.htm`

[22] J. Nocedal, A. Wächter and R.A. Waltz. Adaptive barrier strategies for nonlinear interior methods. *Research Report RC 23563*, IBM T. J. Watson Research Center, Yorktown, USA, 2005.

[23] M.W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large scale symmetric travelling salesman problems. *SIAM Review* 33:60–100, 1991.

[24] I. Quesada and I.E. Grossmann. An LP/NLP based branched and bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.

[25] R. Raman and I.E. Grossmann. Modeling and computational techniques for logic based integer programming. *Computers and Chemical Engineering*, 18:563–578, 1994.

[26] D.E. Ravemark. *Optimization models for design and operation of chemical batch processes*. PhD thesis, Swiss Federal Institute of Technology, 1995.

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

[28] N.W. Sawaya. *A generalized disjunctive framework for solving discrete-continuous optimization problems with convex relaxations*. PhD thesis, Chemical Engineering Department, Carnegie Mellon University, 2006.

[29] `SBB. http://www.conopt.com/sbb/SBB_announcement.htm`

[30] R. Stubbs and S. Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86:515–532, 1999.

[31] M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.

[32] M. Turkay and I.E. Grossmann. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers & Chemical Engineering*, 20:959–978, 1996.

[33] A. Vechietti and I.E. Grossmann. LOGMIP: a disjunctive 0-1 non-linear optimizer for process systems models. *Computers & Chemical Engineering*, 23:555–565, 1999.

[34] J. Viswanathan and I.E. Grossmann. A combined penalty function and outer-approximation method for MINLP optimization. *Computers & Chemical Engineering*, 14:769, 1990.

[35] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming* 2005.

[36] T. Westerlund and F. Pettersson. An extended cutting plane method for solving convex MINLP problems. *Computers and Chemical Engineering*, 19:131–136, 1995.

[37] L.A. Wolsey. *Integer Programming*. Wiley, 1998.

Problem name	SBB	I-BB	I-Hyb	I-OA	Dicopt
BatchS101006M	728.46	124.87	72.90	551.56	**14.69**
BatchS121208M	2316.28	1894.99	71.76	173.54	**9.3**
BatchS151208M	3871.95	2732.32	632.39	3360.58	**22.53**
BatchS201210M	> 3hr	5304.94	816.16	>3hr	**28.44**
CLay0203H	**3.39**	20.15	39.45	6.91	5.011
CLay0204H	15.32	496.06	48.02	7.85	**0.59**
CLay0205M	**19.49**	284.01	114.19	179.72	156.67
CLay0205H	358.72	***	428.93	1048.37	**157.74**
CLay0303M	**0.87**	13.44	55.84	11.15	99.91
CLay0303H	**2.86**	28.30	69.98	26.45	58.5
CLay0305M	**65.77**	644.29	79.87	659.44	769.73
FLay04H	**28.49**	75.89	68.12	148.14	439.5
FLay05M	**266.74**	955.89	7989.17	>3hr	> 3hr
FLay05H	**3047.4**	4066.47	***	>3hr	> 3hr
Fo7_2	>3hr	9833.10	**7103.02**	>3hr	> 3hr
Water0202	N/A	545.95	**390.11**	974.19	N/A
Water0303	N/A	1523.86	**1153.18**	>3hr	N/A
Water0202R	N/A	328.20	156.45	**115.28**	N/A
RSyn0810M03H	84.31	301.51	14.70	15.87	**10.52**
RSyn0815M03H	150.41	188.10	27.78	**27.76**	***
RSyn0820M03H	690.51	369.06	62.85	32.66	**8.3**
RSyn0820M04H	384.62	342.76	57.35	43.68	**18.97**
RSyn0830M03H	98.84	389.56	**7.82**	11.80	10.03
RSyn0840M04H	817.25	2182.70	118.43	69.74	**14.25**
RSyn0840M03H	151.37	319.17	13.27	20.02	**12.45**
RSyn0840M04H	1032.87	3119.50	22.94	60.32	**20.68**
SLay10M	**19.45**	514.56	197.93	>3hr	> 3hr
SLay07H	160.54	53.22	73.46	***	**37.97**
SLay08H	394.46	**107.05**	120.54	>3hr	228.38
SLay09MH	**21.51**	70.95	39.95	>3hr	> 3hr
SLay09H	2383.73	**260.69**	328.35	>3hr	> 3hr
Syn20M04M	> 3hr	> 3hr	52.50	174.60	**0.27**
Syn30M03M	> 3hr	> 3hr	16.89	15.06	**0.72**
Syn30M04M	> 3hr	> 3hr	**118.01**	1544.91	647.5
Syn40M02M	> 3hr	> 3hr	15.55	15.32	**0.44**
Syn40M03M	> 3hr	> 3hr	238.34	1875.38	**1.2**
Syn40M03H	36.3	24.07	17.33	24.32	**2.04**
Syn40M04M	> 3hr	> 3hr	256.23	>3hr	**2.06**
Syn40M04H	81.52	46.64	28.62	34.95	**3.51**
trimloss4	>3hr	***	**178.60**	>3hr	> 3hr
trimloss5	>3hr	***	>3hr	>3hr	> 3hr

Table 2: Comparison of running times (in seconds) for the five algorithms (bold face for best running time).

Problem name	I-BB		I-Hyb					I-OA		
	time	nodes	time	nodes	NLP	OAIF	% time NLP	time	nodes	NLP
BatchS101006M	124.87	1532	**72.90**	502	47	5	31.56	551.56	78404	10
BatchS121208M	1894.99	14144	**71.76**	206	20	3	20.44	173.54	15083	4
BatchS151208M	2732.32	14677	**632.39**	4352	343	5	66.12	3360.58	258804	6
BatchS201210M	5304.94	18378	**816.16**	3608	327	4	66.33	>3hr	>613631	>2
CLay0203H	20.15	206	39.45	138	101	41	35.78	**6.91**	2564	11
CLay0204H	496.06	4392	48.02	928	42	1	24.55	**7.85**	4490	1
CLay0205M	284.01	11171	**114.19**	5670	394	10	11.21	179.72	155440	6
CLay0205H	***	***	**428.93**	7035	417	8	61.99	1048.37	355333	4
CLay0303M	13.44	454	55.84	634	324	41	36.29	**11.15**	5688	13
CLay0303H	28.30	257	69.98	504	191	15	55.89	**26.45**	7119	13
CLay0305M	644.29	19759	**79.87**	3905	264	9	11.64	659.44	495398	8
FLay04H	75.89	2376	**68.12**	2094	574	425	34.85	148.14	49961	21
FLay05M	**955.89**	93632	7989.17	70904	18942	13752	4.74	6566.21	4582107	54
FLay05H	**4066.47**	88220	***	***	***	***	***	>3hr	>2063219	>18
Fo7_2	9833.10	236974	**7103.02**	53908	4306	4	12.99	>3hr	>2351933	>1
Water0202	545.95	32	**390.11**	14	16	14	71.71	974.19	118	7
Water0303	1523.86	80	**1153.18**	72	33	27	50.39	>3hr	>1525	>24
Water0202R	328.20	190	156.45	188	65	53	90.35	**115.28**	2356	27
RSyn0810M03H	301.51	1164	**14.70**	0	4	3	11.80	15.87	201	3
RSyn0815M03H	188.10	638	27.78	0	6	5	7.98	**27.76**	301	5
RSyn0820M03H	369.06	1043	62.85	16	12	8	46.68	**32.66**	414	2
RSyn0820M04H	342.76	621	57.35	24	14	11	37.79	**43.68**	551	3
RSyn0830M03H	389.56	943	**7.82**	0	3	2	19.66	11.80	137	2
RSyn0840M04H	2182.70	3230	118.43	74	24	17	57.11	**69.74**	486	3
RSyn0840M03H	319.17	640	**13.27**	0	4	3	20.18	20.02	375	3
RSyn0840M04H	3119.50	3933	**22.94**	0	3	2	11.98	60.32	462	2
SLay10M	514.56	16072	**197.93**	6548	563	34	18.75	>3hr	>2034251	>1
SLay07H	**53.22**	775	73.46	954	100	13	11.85	***	***	***
SLay08H	**107.05**	1187	120.54	1372	157	3	13.55	>3hr	>1643379	>1
SLay09MH	70.95	2488	**39.95**	1309	147	9	16.99	>3hr	>3646655	>14
SLay09H	**260.69**	2239	328.35	5022	389	5	1.66	>3hr	>1100614	>1
Syn20M04M	> 3hr	>100475	**52.50**	358	34	5	11.57	174.60	14550	2
Syn30M03M	> 3hr	>132373	16.89	0	4	3	2.24	**15.06**	746	3
Syn30M04M	> 3hr	>79144	**118.01**	886	119	41	18.08	1544.91	141255	4
Syn40M02M	> 3hr	>179089	15.55	0	4	3	2.18	**15.32**	1180	3
Syn40M03M	> 3hr	>84501	**238.34**	2154	309	118	20.07	1875.38	163802	5
Syn40M03H	24.07	96	**17.33**	0	5	4	51.96	24.32	422	4
Syn40M04M	> 3hr	>57992	**256.23**	1322	129	6	14.16	>3hr	952994	>1
Syn40M04H	46.64	126	**28.62**	0	6	5	53.97	34.95	543	4
trimloss4	***	***	**178.60**	12773	985	4	15.33	>3hr	>11869930	>2
trimloss5	***	***	>3hr	> 104937	> 9715	> 0	***	>3hr	>3240229	>1

Table 3: Detailed comparison of `I-BB`, `I-Hyb`, `I-OA`. Cpu times (in seconds), number of nodes, and number of NLPs solved are displayed. For `I-Hyb`, the number of times the OA linear relaxation has an integer solution (OAIF), forcing the solution of an NLP, and the percent of time spent solving NLPs are also listed (bold face for best running time).

Problem name	SBB	I-BB
BatchS101006M	22590	1532
BatchS121208M	53580	14144
BatchS151208M	68484	14677
BatchS201210M	>161338	18378
CLay0203H	420	206
CLay0204H	2808	4392
CLay0205M	10749	11171
CLay0205H	29730	***
CLay0303M	387	454
CLay0303H	298	257
CLay0305M	21352	19759
FLay04H	2808	2376
FLay05M	86812	93632
FLay05H	112388	88220
Fo7_2	>217106	236974
RSyn0810M03H	232	1164
RSyn0815M03H	364	638
RSyn0820M03H	1540	1043
RSyn0820M04H	314	621
RSyn0830M03H	193	943
RSyn0840M04H	877	3230
RSyn0840M03H	222	640
RSyn0840M04H	833	3933
SLay10M	1858	16072
SLay07H	6335	775
SLay08H	7634	1187
SLay09MH	2227	2488
SLay09H	31015	2239
Syn20M04M	>497221	>100475
Syn30M03M	>328268	>132373
Syn30M04M	>160695	>79144
Syn40M02M	>492640	>179089
Syn40M03M	>177817	>84501
Syn40M03H	72	96
Syn40M04M	>212265	>57992
Syn40M04H	100	126
trimloss4	>1277211	***
trimloss5	>1000000	***

Table 4: Comparison of number of nodes in the two NLP BB.

Problem name	I-OA	Dicopt	% of gap closed
BatchS101006M	10	14	75.26
BatchS121208M	4	5	82.83
BatchS151208M	6	7	78.76
BatchS201210M	>2	4	86.64
CLay0203H	11	77	8.56
CLay0204H	1	2	100
CLay0205M	6	120	99.90
CLay0205H	4	24	99.90
CLay0303M	13	516	13.34
CLay0303H	13	333	13.34
CLay0305M	8	282	99.90
FLay04H	21	332	72.39
FLay05M	>54	>799	26.79
FLay05H	>18	>757	26.79
Fo7_2	>1	>614	87.19*
RSyn0810M03H	3	4	64.89
RSyn0815M03H	5	***	71.67
RSyn0820M03H	2	3	90.58
RSyn0820M04H	3	4	68.85
RSyn0830M03H	2	3	85.26
RSyn0840M04H	3	3	80.01
RSyn0840M03H	3	3	75.55
RSyn0840M04H	2	3	74.74
SLay10M	>1	>600	2.12*
SLay07H	***	54	8.95
SLay08H	>1	70	0*
SLay09MH	>14	>691	10.79
SLay09H	>1	>234	0*
Syn20M04M	2	3	42.21
Syn30M03M	3	5	63.93
Syn30M04M	4	249	64.52
Syn40M02M	3	4	77.63
Syn40M03M	5	5	79.28
Syn40M03H	4	5	0*
Syn40M04M	>1	4	75.56*
Syn40M04H	4	3	0.08
trimloss4	>2	>97	24.13
trimloss5	>1	>75	21.41*

Table 5: Comparison of number of iterations in the two OA algorithms and percentage of gap closed by solving the first MILP in `I-OA` ("$*$" indicates that the first MILP is not solved within the time limit ; therefore the given number is only a lower bound on the gap closed).