ISE Industrial and Systems Engineering

Disjunctive Cuts for Non-Convex MINLP

Pietro Belotti
Lehigh University

Report: 09T-017

DISJUNCTIVE CUTS FOR NON-CONVEX MINLP

PIETRO BELOTTI*

Abstract. Mixed-Integer Nonlinear Programming (MINLP) problems present two main challenges: the integrality of a subset of variables and non-convex (nonlinear) objective function and constraints. Both types of non-convexity can be dealt with by applying, explicitly or implicitly, disjunctions on both integer and continuous variables.

Several solution approaches obtain a mixed-integer linear programming relaxation of the original problem, and rely on branch-and-cut techniques for solving the problem to global optimality. In the MINLP context, using disjunctions for branching has been subject to intense research, while the use of disjunctions as a means of generating valid linear inequalities has attracted some attention only recently.

We describe a straightforward extension of a separation method for disjunctive cuts that has shown to be very effective in Mixed-Integer Linear Programming (MILP). The theoretical and implementation aspects are very close to the MILP case, and, as the experimental results show, this extension obtains encouraging results in the MINLP case even when a simple separation method is used.

Key words. Disjunctions, Global Optimization, Couenne, Disjunctive cuts.

AMS(MOS) subject classifications. 90C57

1. Motivation: non-convex MINLP.

Mixed integer nonlinear programming is a powerful modeling tool for very generally defined problems in optimization [23]. A MINLP problem is defined as follows:

$$
\begin{array}{llll}
(\mathbf{P_0}) & \min & f(x) & \\
 & \text{s.t.} & g_j(x) \le 0 & \forall j = 1,2\ldots,m \\
 & & \ell_i \le x_i \le u_i & \forall i = 1,2\ldots,n \\
 & & x_i \in \mathbb{Z}^r \times \mathbb{R}^{n-r}, &
\end{array}
$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g_j : \mathbb{R}^n \to \mathbb{R}$, for all $j = 1,2\ldots,m$, are in general multivariate, non-convex functions, n is the number of variables, and x is the n-vector of variables. Throughout this paper, x will denote the variable vector and x_i its i-th component.

We assume that f and all g_j's are *factorable*, i.e., they can be computed in a finite number of simple steps, starting with model variables and real constants, using unary (e.g., log, exp, sin, cos, tan) and binary operators (e.g., $+$, $-$, $*$, $/$, $\hat{}$). Notice that this framework, although very general, excludes problems whose constraints or objective function are, for example, black-box functions or indefinite integrals such as the *error function*.

There are numerous applications of problem $\mathbf{P_0}$, in Chemical Engineering [9, 17, 25], Finance [14], and Computational Biology [27, 28, 38], to name only a few. Because general-purpose solvers for MINLP problems make no assumption besides factorable functions, this framework comprises

*Dept. of Industrial & Systems Engineering, Lehigh University, Bethlehem PA. Email: belotti@lehigh.edu several special cases that have been thoroughly studied, for example Mixed Integer Linear Programming (MILP), Convex Optimization, and convex MINLPs (i.e., MINLPs whose continuous relaxation is a convex nonlinear program). For this reason, however, problems exhibiting some structure are usually solved much more efficiently by a specialized solver, thus the motivation behind solving, with a general-purpose MINLP solver, a problem in the form $\mathbf{P_0}$ is usually its lack of structure.

The difficulty of MINLPs resides in two types of non-convexity: the integrality of a subset of variables and the fact that f and g_j are non-convex functions. Both types of non-convexity can be tackled by means of *disjunctions*, either applying them as branching rules within a branch-and-bound (BB) framework or by using them to obtain tighter relaxations. Although cuts derived from disjunctions have been developed for MILP decades ago, their extension to the nonlinear context is recent: non-convex quadratic constraints have been used by [47, 46] to obtain disjunctive cuts for Quadratically Constrained Quadratic Programming (QCQP). Our contribution is an extension, to the MINLP context, of a very well-known technique for generating disjunctive cuts for MILP problems. We present experimental evidence that disjunctive cuts can have a significant positive impact on the solution time of MINLP problems.

Exact solution methods for MINLP rely, in order to obtain valid lower bounds, on reformulation and linearization techniques [50, 51], which we briefly introduce in Section 2 as their definition is essential to understanding the type of disjunctions we use in this context. Section 3 describes the general disjunctions arising from reformulating an MINLP, and their use in separating disjunctive cuts is shown in Section 4.

A simple separation procedure, recalling the well-known CGLP procedure for MILP problems, is detailed in Section 5. This procedure has been incorporated in COUENNE, a general-purpose, Open Source solver for MINLP [7], and has been tested on a set of publicly available non-convex MINLP instances. These tests are presented in Section 6.

2. Exact solution methods for MINLP. Most MINLP solvers are BB algorithms whose lower bounding technique uses a Linear Programming (LP) relaxation constructed in two steps:

- *reformulation:* $\mathbf{P_0}$ is transformed in an equivalent MINLP with q new variables, a linear objective function, and a set of nonlinear equality constraints;
- *linearization:* each of the newly added nonlinear constraints is relaxed by replacing it with a set of linear inequalities.

The interested reader can find more information in [50, 51]. Here we offer a shortened version whose purpose is to understand the definition of disjunctions given in Section 3.

In reformulation, the expression trees of the objective function and of all constraints of $\mathbf{P_0}$ are decomposed so that each non-leaf node of an expression tree, which is an operator of a finite set, is associated with a new variable, usually called *auxiliary variable.* Reformulation yields the following problem:

$$
\begin{array}{llll}
\textbf{(P)} & \min & x_{n+q} & \\
& \text{s.t.} & x_k = \vartheta_k(x) & k = n+1, n+2 \ldots, n+q \\
& & \ell_i \le x_i \le u_i & i = 1, 2 \ldots, n+q \\
& & x \in \mathcal{X} &
\end{array}
$$

where $\mathcal{X} = \mathbb{Z}^r \times \mathbb{R}^{n-r} \times \mathbb{Z}^s \times \mathbb{R}^{q-s}$, that is, q auxiliary variables are introduced, s of which are integral[1] and $q-s$ continuous. Each auxiliary x_k is associated with a function $\vartheta_k(x)$ which is from a set of binary and unary operators $\{+, *, \hat{}\,, /, \sin, \cos, \exp, \log\}$ and depends, in general, on one or more of the variables $x_1, x_2 \ldots, x_{k-1}$. As an example, the problem

$$
\begin{array}{ll}
\min & (x_1)^4 \\
\text{s.t.} & x_1 \log x_2 \ge 4 \\
& x_1 \ge 0, x_2 \ge 2, x_1 \in \mathbb{Z}
\end{array}
$$

admits the following reformulation:

$$
\begin{array}{ll}
\min & x_5 \\
\text{s.t.} & x_3 = \log x_2 \\
& x_4 = x_1 x_3 \\
& x_5 = (x_1)^4 \\
& x_1 \ge 0, x_2 \ge 2 \\
& x_3 \ge \log 2, x_4 \ge 4, x_5 \ge 0, x_5 \in \mathbb{Z},
\end{array}
$$

where the integrality of x_5 derives from that of x_1, of which x_5 is a function, and the lower bound on x_3 is inferred from the lower bound on x_2 and on the fact that the logarithm is a monotone increasing function. The bound on x_4 is the right-hand side of the constraint of the original problem.

After reformulation, all nonlinear constraints of the problem are of the form $x_i = \vartheta_i(x)$ for all $i = n+1, n+2 \ldots, n+q$, with ϑ_i an operator from a well-defined set. It is then possible to *linearize* each such constraint by applying operator-specific algorithms.

Let us consider, as an example, a constraint $x_j = \vartheta_j(x) = (x_i)^3$ from the reformulation, and the non-convex set $S = \{(x_i, x_j) : \ell_i \le x_i \le u_i, x_j = (x_i)^3\}$, i.e., the bold curve in Figure 1(a). A convex (polyhedral) set containing S, depicted in Figure 1(b), is obtained through a procedure based on the function $\vartheta(x) = x^3$ and the bounds on x_i. We will not describe the method(s) yielding such a convex set, as this is out of the scope of this work; the interested reader is referred again to [50, 51]. Suffice it to say that this linear relaxation aims at approximating the *convex envelope* of

[1]The integrality of x_k, for $k = n+1, n+2 \ldots, n+q$, is inferred from the associated function ϑ_k and the integrality of its arguments.

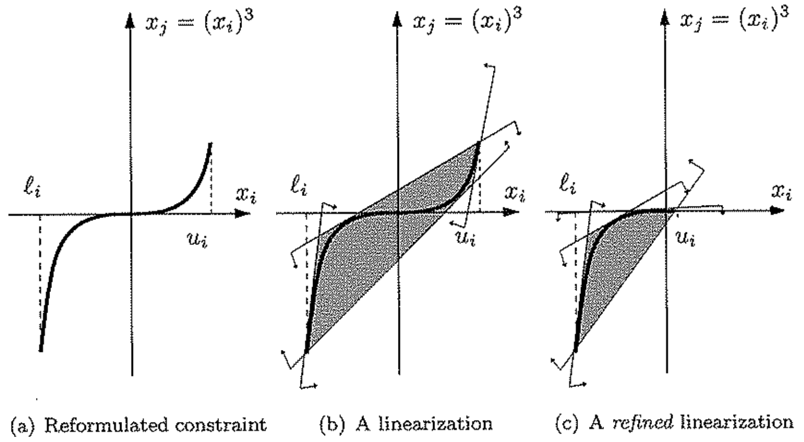


FIG. 1. *Linearization of constraint $x_j = (x_i)^3$: the bold line in (a) represents the non-convex set $\{(x_i, x_j) : \ell_i \leq x_i \leq u_i, x_j = (x_i)^3\}$, while the polyhedra in (b) and (c) are its linearizations for different bounds on x_i.*

S, i.e., the tightest convex set containing S. Figure 1(c) shows that the quality of the linearization highly depends on the bounds on the variable x_i: upon branching on x_i or after one of its bounds has been tightened, the linearization can also be strengthened. Hence, applying a disjunction through a branching rule may help improve the lower bound.

When applied to all nonlinear constraints appearing in the reformulated problem, this step generates an LP relaxation of the original problem $\mathbf{P_0}$, and therefore a valid lower bound. Also, if used as the bounding procedure in a BB algorithm, it allows to solve $\mathbf{P_0}$ to optimality.

It is worth pointing out that most linearization methods obtain a polyhedral linearization that is more accurate near the variable bounds, which is a desirable property, especially within a BB scheme: branching usually occurs where the linearization is poor, and branching far from the variable bounds helps prevent creating two very unbalanced subtrees.

The reformulation procedure can be improved, for instance, by limiting the number of new variables: an equality constraint $f(x) + ax_i + c = 0$ can be translated into $x_i = -\frac{1}{a}(f(x) + c)$, hence avoiding the need of an extra variable assigned to $f(x) + ax_i + c$.

The main weakness of the reformulation procedure is that the links between variables are somewhat broken in order to obtain simpler functions ϑ_k for linearization. Consider the convex constraint $x_1^2 - 2x_1x_2 + x_2^2 \leq 1$, which is clearly equivalent to $|x_1 - x_2| \leq 1$ and may be written in linear form as $-1 \leq (x_1 - x_2) \leq 1$. A plain reformulation procedure will simply decompose this constraint as

$$\begin{aligned} x_3 &= x_1^2 \\ x_4 &= x_1 x_2 \\ x_5 &= x_2^2 \\ x_6 &= x_3 - 2x_4 + x_5 \\ x_6 &\leq 1 \end{aligned}$$

and introduce three non-convex constraints, thus making the reformulation non-convex. A better solver would recognize, during reformulation, the quadratic form structure, and then for instance provide a linearization based on Outer Approximation [16], thus giving a better lower bound.

In general, depending on the reformulation procedure, i.e., on the solver's ability to recognize special constraints or expression structures, and depending even more on the set of operators for which a linearization method is available, a tighter linearization and hence a better lower bound can be obtained.

3. Disjunctions in MINLP. Disjunctions arise as a natural modeling tool in Mixed Integer Linear Programming: for an integer variable x_i and any integer a, the disjunction $x_i \leq a \vee x_i \geq a+1$ must hold for any feasible solution. One of the purposes of such disjunctions is to obtain some information on the convex hull of an MILP described as a system of linear inequalities intersected with a set of disjunctions [2].

Consider the simple MILP in two variables depicted in Figure 2. Its linear relaxation is described by the dark shaded area in Figure 2(a). If both variables are constrained to be integer, the only two feasible points are $(0,0)$ and $(1,1)$. Applying the disjunction $x_1 \leq 0 \vee x_1 \geq 1$ as a branching rule obtains the two subproblems identified by the bold segments in Figure 2(b). However, by using the disjunction to obtain the convex hull of the union of these two subproblems, or an inequality that separates the current LP solution from said convex hull, the tighter LP relaxation shown in Figure 2(c) can be obtained, with a consequent better lower bound.

There has been extensive research on how to efficiently add such inequalities, using a BB method coupled with a cut generating procedure [4, 5, 37]. Several generalizations can be introduced at the MILP level: for instance, rather than a disjunction on a single variable, one may think of the more general disjunction $\pi x \leq \pi_0 \vee \pi x \geq \pi_0 + 1$, where $(\pi, \pi_0) \in \mathbb{Z}^{p+1}$ and $x \in \mathbb{Z}^p$ is a vector of p integer variables. See [33] for a more complete discussion on the use of disjunctions in MILP. We will not elaborate on MILP disjunctions based on integer variables, and only anticipate that they are used in a similar way in our MINLP framework.

In MINLP problems, disjunctions are created from the non-convex constraints $x_k = \vartheta_k(x)$ that arise in the reformulation step. As pointed out in the previous section, both types of non-convexities present in an MINLP, i.e., integer variables and nonlinear functions, are relaxed to obtain a LP

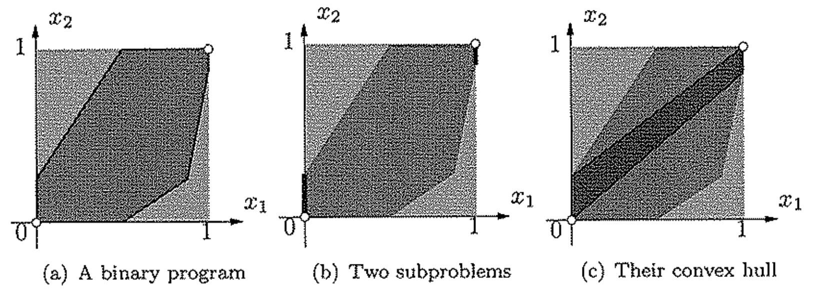


FIG. 2. *Applying a disjunction in a Mixed integer linear program.*

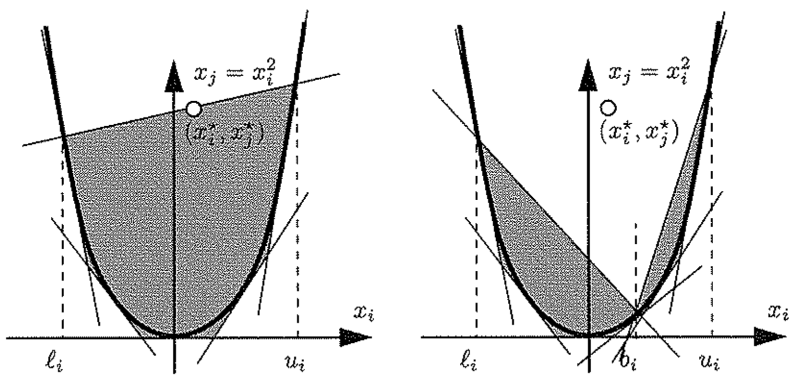
relaxation that provides a valid lower bound. The optimal solution to the LP relaxation, which we denote $x^\star$, may be infeasible for either the integrality constraints or for at least one of the constraints of the reformulation, $x_k = \vartheta_k(x)$. If x_i is an integer variable but the value $x_i^\star$ is fractional, the two resulting subproblems satisfy $x_i \leq \lfloor x_i^\star \rfloor$ and $x_i \geq \lceil x_i^\star \rceil$ respectively. If x_i is a continuous variable, it might be necessary to use a disjunction on x_i as illustrated in the following example.

Consider the function $x_j = \vartheta_j(x_i) = x_i^2$ and the LP solution $x^\star$ as in Figure 3(a). Clearly, $(x_i^\star, x_j^\star)$ lies in the linearization of $x_j = x_i^2$. We observe that $(x_i^\star, x_j^\star)$ is not necessarily a vertex of the linearization of $x_j = x_i^2$ since it is simply a projection of $x^\star$ (which is itself a vertex of the LP relaxation) onto the plane (x_i, x_j). Suppose also that $x_j^\star > (x_i^\star)^2$ (see Figure 3(a)). As $(x_i^\star, x_j^\star)$ is not separable with another linearization cut, the disjunction

$$x_i \leq b_i \quad \vee \quad x_i \geq b_i \tag{3.1}$$

must be applied, for some $b_i \in [\ell_i, u_i]$, for example as a branching rule. Branching creates two subproblems satisfying $x_i \leq b_i$ and $x_i \geq b_i$ respectively, and admitting two linearizations, both excluding $(x_i^\star, x_j^\star)$. The use of disjunctions in MINLP, and especially the problem of finding the most effective disjunction for branching, has been subject of research in the past. Several techniques have been proposed to select a branching variable, such as Violation Transfer [29, 52], and to select the branching point b_i [24, 29, 44, 48]. A generalization of the *reliability branching* technique [1] to the MINLP case has been recently presented [8].

As shown in Figure 3(b), a portion of the solution space included by the original relaxation can be eliminated by *refining* the linearization, i.e., by re-applying the linearization step to the subproblem. We observe that this is a very important point of departure with MILP, where branching alone, i.e., without any other subproblem pre-processing, eliminates both the fractional solution and a portion of the solution space. In MINLP, as branching may occur on continuous variables, the disjunction (3.1) does not



(a) Infeasible solution of a linearization (b) Linearizations of the subproblems

FIG. 3. *Branching on an MINLP disjunction. In (a), the shaded area is the linearization of the constraint $x_j = \vartheta_j(x_i)$ with $x_i \in [\ell_i, u_i]$, whereas (x_i^*, x_j^*) is the value of x_i and x_j in the optimum of the LP relaxation. In (b), a branching rule divides the interval $[\ell_i, u_i]$ to create two subproblems, with x_i bounded by $[\ell_i, b_i]$ and $[b_i, u_i]$, respectively, and which admit two new linearizations, the smaller shaded areas.*

eliminate any solution from the linearization, and an extra step is required in order to ensure termination of a BB algorithm: a refined linearization (see Figure 1(c)) can be obtained, in both subproblems, that takes into account the new bounds on x_i.

Moreover, MINLP solvers typically apply bound reduction methods [20, 36, 40] to eliminate suboptimal solutions. Several such reduction techniques have proven to make the BB more efficient, including those based on the LP relaxation [12, 39, 49], on the nonlinear constraints [13, 22, 34, 45, 48, 49, 53], or on the reduced costs of the LP relaxation [42, 43].

Another type of disjunction, and a procedure to derive valid cuts, has been studied for quadratically constrained quadratic programs (QCQPs) by [47, 46]. These problems can be reformulated as linear programs with an extra non-convex constraint of the form $Y = xx^\top$, where Y is an $n \times n$ matrix of auxiliary variables and x is the n-vector of variables. Such a constraint can be relaxed as $Y - xx^\top \succeq 0$, thus reducing a QCQP to a (convex) Semidefinite program, which yields good lower bounds [21, 41]. However, these SDP models are obtained from the original problem by relaxing the non-convex constraint $xx^\top - Y \succeq 0$. In [47], disjunctive cuts are generated from disjunctions derived from the nonlinear constraint $(v^\top x)^2 \geq v^\top Y v$, where vector v is obtained from the negative eigenvalues of the matrix $\bar{x}\bar{x}^\top - \bar{Y}$, and $(\bar{x}, \bar{Y})$ is a solution to the relaxation. This procedure is then modified [46] to generate cuts for the non-reformulated problem, thus avoiding the need of the auxiliary matrix variable Y.

Other classes of disjunctions are possible: an immediate example arises from mathematical programs with complementarity constraints (MPCCs), a modeling paradigm that finds numerous applications in industry [32]. Complementarity constraints can be expressed in the form $x_i x_j = 0$, which is a special case of quadratic constraint, but a plain reformulation of that constraint would not fully exploit the equivalent disjunctive form $x_i = 0 \vee x_j = 0$. It is therefore advisable for an MINLP solver to recognize such constraints and the corresponding disjunctions, as done for instance in [47].

4. Disjunctive cuts in non-convex MINLP.

Assume that the linearization step produces, from a reformulation in the form $\mathbf{P}$, an MILP relaxation that we denote $\min\{x_{n+q} : Ax \le a, \ell \le x \le u, x \in \mathcal{X}\}$, where $A \in \mathbb{R}^{K \times (n+q)}$, $a \in \mathbb{R}^K$, and K is the total number of linear inequalities generated for all of the nonlinear constraints $x_k = \vartheta_k(x)$ of the reformulation, while ℓ and u are the vectors of the lower and upper bounds on both original and auxiliary variables. Finally, denote the feasible set of the linear relaxation as $\mathbf{LP} = \{x \in \mathbb{R}^{n+q} : Ax \le a, \ell \le x \le u\}$.

Consider again a disjunction $x_i \le b_i \vee x_i \ge b_i$. As pointed out in the previous section, the disjunction alone does not eliminate any solution from the union of the two resulting subproblems:

$$\begin{aligned} \mathbf{LP}^{\mathrm{L}} &= \{x \in \mathbf{LP} : x_i \le b_i\} \\ \mathbf{LP}^{\mathrm{R}} &= \{x \in \mathbf{LP} : x_i \ge b_i\}, \end{aligned}$$

because $\mathbf{LP} = \mathbf{LP}^{\mathrm{L}} \cup \mathbf{LP}^{\mathrm{R}}$, while this does not hold for disjunctions on integer variables, where $\mathbf{LP}$ strictly contains the union of the two subproblems. Therefore, let us suppose that $\mathbf{LP}^{\mathrm{L}}$ and $\mathbf{LP}^{\mathrm{R}}$ are strengthened by applying further linearization inequalities (See Figure 1(c)) or through bound reduction. This will result in two tightened sets,

$$\begin{aligned} \mathbf{SLP}^{\mathrm{L}} &= \{x \in \mathbf{LP} : Bx \le b\}, \\ \mathbf{SLP}^{\mathrm{R}} &= \{x \in \mathbf{LP} : Cx \le c\}, \end{aligned}$$

where $B \in \mathbb{R}^{H' \times (n+q)}$, $b \in \mathbb{R}^{H'}$, $C \in \mathbb{R}^{H'' \times (n+q)}$, and $c \in \mathbb{R}^{H''}$ are the coefficient matrices and the right-hand side vectors of the inequalities (H' of them for the left side, H'' for the right side) that are added to refine or strengthen $\mathbf{LP}$ after each side of the disjunction is taken, and that contain the new bound on variable x_i and, possibly, new bounds on other variables.

We re-write these two sets in a more convenient form:

$$\begin{aligned} \mathbf{SLP}^{\mathrm{L}} &= \{x \in \mathbb{R}^{n+q} : A'x \le a'\} \\ \mathbf{SLP}^{\mathrm{R}} &= \{x \in \mathbb{R}^{n+q} : A''x \le a''\}, \end{aligned}$$

where

$$A' = \begin{pmatrix} A \\ B \\ -I \\ I \end{pmatrix}, \quad a' = \begin{pmatrix} a \\ b \\ \ell \\ u \end{pmatrix}; \qquad A'' = \begin{pmatrix} A \\ C \\ -I \\ I \end{pmatrix}, \quad a'' = \begin{pmatrix} a \\ c \\ \ell \\ u \end{pmatrix}$$

so as to include the initial linear constraints, the refined and strengthened linearization, and the variable bounds in a more compact notation. We denote as K' (resp. K'') the number of rows of A' (resp. A'') and the number of elements of a' (resp. a'').

As described by Balas [2], an inequality $\alpha x \leq \alpha_0$, where $\alpha \in \mathbb{R}^{n+q}$ and $\alpha_0 \in \mathbb{R}$, is valid for the convex hull of $\mathbf{SLP}^{\mathrm{L}} \cup \mathbf{SLP}^{\mathrm{R}}$ if α and α_0 satisfy:

$$\begin{array}{ll} \alpha \leq u^\top A', & \alpha_0 = u^\top a', \\ \alpha \leq v^\top A'', & \alpha_0 = v^\top a'', \end{array}$$

where $u \in \mathbb{R}_+^{K'}$ and $v \in \mathbb{R}_+^{K''}$. Given relaxation **LP** and its optimal solution $x^\star$, an automatic procedure for generating a cut of this type [4] consists in finding proper values of u and v such that the corresponding cut is maximally violated. This is equivalent to solving the following LP problem:

$$\begin{array}{lllllll} \max & \alpha^\top x^\star & -\alpha_0 & & & \\ \text{s.t.} & \alpha & & -u^\top A' & & \leq 0 \\ & \alpha & & & -v^\top A'' & \leq 0 \\ & & \alpha_0 & -u^\top a' & & = 0 \\ & & \alpha_0 & & -v^\top a'' & = 0 \\ & & & u^\top e & +v^\top e & = 1 \\ & & & u, & v & \geq 0 \end{array} \tag{4.1}$$

where e is the vector with all components equal to one. This resembles closely the Cut Generating Linear Problem (CGLP) introduced by Balas et al. [4]. An optimal solution (more precisely, any solution with positive objective value) provides a valid cut that is violated by the current solution $x^\star$. Its main disadvantage is its size: depending on the reformulation and the linearization process, the LP relaxation found can be relatively large compared with the number of variables and constraints of the original problem **P**, and solving (4.1) might prove ineffective. Balas et al. [3, 6] present a method to implicitly solve the CGLP by clever pivot operations on a slightly modified version of the original linear relaxation. It is worth noting that, unlike the MILP case, here A' and A'' differ for much more than a single column, which makes the generalization of such a procedure to MINLP less straightforward. We are not aware of any such generalization.

4.1. An example. In order to clarify the importance of refined formulations, bound reduction, and ultimately, of disjunctive cuts in the context of MINLP, consider the following continuous non-convex NLP:

$$(\mathbf{P}_0) \quad \begin{array}{ll} \min & x^2 \\ \text{s.t.} & x^4 \geq 1. \end{array}$$

It is immediate to check that its feasible set is the non-convex union of intervals $(-\infty, -1] \cup [+1, +\infty)$, and that its two global minima are $\{-1, +1\}$.

Its reformulation is as follows:

$$\begin{array}{lll} (\mathbf{REF}) & \min & w \\ & \text{s.t.} & w = x^2 \\ & & y = x^4 \\ & & y \geq 1, \end{array}$$

and it is apparent that reformulating breaks the link between the objective function and the constraint, a link identified with variable x. Its tightest convex relaxation is obtained by simply replacing the equality with inequality constraints:

$$\begin{array}{lll} (\mathbf{CR}) & \min & w \\ & \text{s.t.} & w \geq x^2 \\ & & y \geq x^4 \\ & & y \geq 1, \end{array}$$

and its optimal solution is $(x, w, y) = (0, 0, 1)$, whose value is $w = 0$ and which is infeasible for **REF** and hence for **P**. Applying the disjunction $x \leq 0 \vee x \geq 0$ yields two subproblems which admit the following two tightest relaxations:

$$\begin{array}{ll} (\mathbf{CR}_L) & \min\{w : y \geq 1, w \geq x^2, y \geq x^4, x \leq 0\} \\ (\mathbf{CR}_R) & \min\{w : y \geq 1, w \geq x^2, y \geq x^4, x \geq 0\} \end{array}$$

but both still admit the same optimal solution, $(x, w, y) = (0, 0, 1)$. Bound reduction is crucial here for both subproblems, as it strengthens the bounds on x using the lower bound on y. Indeed, $x \leq 0$ and $1 \leq y = x^2$ imply $x \leq -1$, and analogously $x \geq 0$ and $1 \leq y = x^2$ imply $x \geq 1$. Hence, the relaxations of the two tightened subproblems are

$$\begin{array}{ll} (\mathbf{CR}'_L) & \min\{w : y \geq 1, w \geq x^2, y \geq x^4, x \leq -1\} \\ (\mathbf{CR}'_R) & \min\{w : y \geq 1, w \geq x^2, y \geq x^4, x \geq +1\} \end{array}$$

and their optimal solutions are feasible for $\mathbf{P_0}$ and correspond to the two global optima $\{-1, +1\}$. Hence, the problem is solved after branching on the disjunction $x \leq 0 \vee x \geq 0$. However, the nonlinear inequality $x^2 \geq 1$ is valid for both $\mathbf{CR}_L$ and $\mathbf{CR}_R$ as it is implied by $x \leq -1$ in the former and by $x \geq 1$ in the latter. Since $w \geq x^2$, the (linear) disjunctive cut $w \geq 1$ is valid for both $(\mathbf{CR}'_L)$ and $(\mathbf{CR}'_R)$. If added to $(\mathbf{CR})$, a much better lower bound is obtained which allows to solve the problem without branching.

This simple example can be complicated arbitrarily by considering n variables subject each to a non-convex constraint:

$$\begin{array}{lll} \min & \sum_{i=1}^{n} x_i^2 & \\ \text{s.t.} & x_i^4 \geq 1 & \forall i = 1, 2 \ldots, n. \end{array}$$

A standard BB implementation needs to branch on all variables before closing the gap between lower and upper bound, requiring an exponential number of subproblems as all disjunctions have to be applied independently. However, the set of disjunctive cuts $w_i \geq 1 \forall i = 1, 2 \ldots, n$, where w_i is the variable associated with the expression x_i^2, allow to close the gap very quickly. Although this example uses a nonlinear convex envelope for the problem and the subproblems, it can be shown that the same disjunctive cut can be generated within a framework where linear relaxations are used.

A test with $n = 100$ has given the following result: with disjunctive cuts generated at all nodes (at most 20 per node), the problem was solved to global optimality (the optimal solution has a value of 100) in 57 seconds and 18 BB nodes on a Pentium M 2.0GHz laptop, 41 seconds of which were spent in the generation of such cuts. Without disjunctive cuts, the solver was stopped after two hours of computation, more than 319,000 active BB nodes, and a lower bound of 14.

As a side note, one might expect a BB procedure to enforce all disjunctions as branching rules. Hence, at any node of depth d in the BB tree the d disjunctions applied down to that level determine the bound to be equal to d. Because all optima have objective value equal to n, an exponential number of nodes will have to be explored, so that the global lower bound of the BB will increase as the logarithm of the number of nodes.

This example can be interpreted as follows: the decomposition of the expression trees at the reformulation phase is followed by a linearization that only takes into account, for each nonlinear constraint $y_i = x_i^4$, of the variables x_i and y_i only — this causes a bad lower bound as there is no link between the lower bound on y_i and that on w_i. The disjunctive cuts, while still based on a poor-quality LP relaxation, have a more "global" perspective of the problem, where the bound on y_i implies a bound on w_i.

5. A procedure to generate disjunctive cuts. The previous section describes an MINLP extension of a procedure to generate maximally violated disjunctive cuts, originally introduced for the MILP case [4]. The procedure is sketched it in Table 1 and is further discussed below. It consists of three steps: (i) select a disjunction; (ii) create two subproblems by applying the disjunction; (iii) solve the CGLP to generate a cut valid for both subproblems. Nonlinear extensions have appeared previously for the QCQP case: the disjunctive cuts in [47, 46] are obtained by applying the procedure to the disjunctions outlined in Section 3 for reformulations of QCQPs and to disjunctions of the form (3.1), with $b_i = \frac{\ell_i + u_i}{2}$, for all variables appearing in bilinear terms.

Branch-and-bound procedure. The MINLP solver is implemented as a BB whose lower bounding method is based on the reformulation and linearization steps as discussed in Section 2. An the beginning, the linearization procedure is run in order to obtain an initial MILP relaxation. At each node of the BB, two cut generation procedures are used: one round

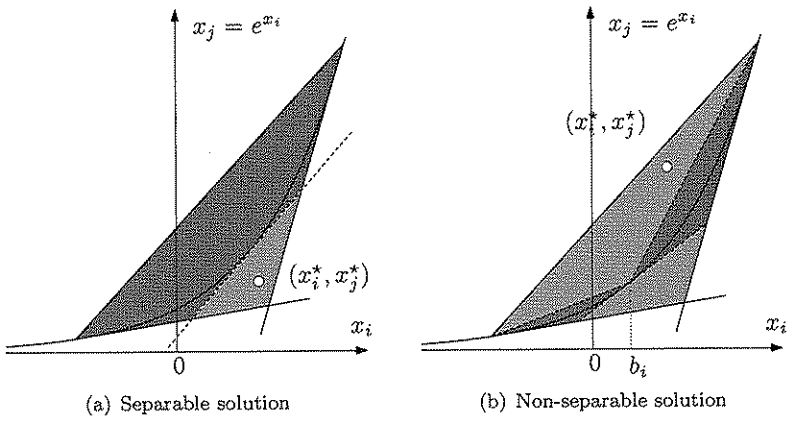


FIG. 4. *Separable and non-separable points. In (a), although the point is infeasible for* **P**, *another round of linearization cuts is preferred to a disjunction (either by branching or through a disjunctive cut), as much quicker to separate. In (b), no refinement of the linear relaxation is possible, and a disjunction must be applied.*

of cuts to refine the linear relaxation and one round of disjunctive cuts.

Calls to the disjunctive cut generator. While linearization cuts are separated quickly and hence are used throughout the BB, disjunctive cuts are quite CPU-intensive as they require solving a large LP for each disjunction. Therefore, in our setting, disjunctive cuts are only generated at BB nodes of depth lesser than 10, and at most 20 disjunctions per node are analyzed.

"Separability" of a nonlinear function. Disjunctions can help cut an LP solution through branching or disjunctive cuts, but they are not the only way to do so. Consider Figure 4, where the components $(x_i^\star, x_j^\star)$ of the LP solution associated with the nonlinear constraint $x_j = e^{x_i}$ are shown in two cases: below and above the curve. In both cases, a disjunction $x_i \leq b_i \vee x_i \geq b_i$ would create two BB nodes both excluding $x^\star$, and analogously with the corresponding disjunctive cut. However, the point in Figure 4(a) can be cut simply by refining the LP relaxation, thus avoiding the evaluation of a disjunction on x_i. Even if such a refinement is not carried out at the present BB node (for instance, because the maximum number of separation rounds has been reached), it will be performed at subsequent nodes or, if a disjunctive cut is chosen instead of branching, at a subsequent call to the linearization cut separator. For this reason, a disjunction is evaluated, for branching or for separating a disjunctive cut, only if a refinement is not possible.

Disjunctions on integer variables. Integer variable disjunctions $x_i \leq \lfloor x_i^\star \rfloor \vee x_i \geq \lceil x_i^\star \rceil$ are also used in this framework, as they may also lead to two subproblems with refined relaxation and tightened bounds. Although a much more efficient separation technique is available in the COIN-OR cut

input:	Linear relaxation, **LP**: (A, a, ℓ, u)
	Optimal solution of LP: x^*
	A disjunction $x_i \leq b_i \vee x_i \geq b_i$
output:	Coefficients of disjunctive cut: (α, α_0)
1.	Apply disjunction to **LP**
	$\Rightarrow$ get subproblems $\mathbf{LP}^{\mathrm{L}}$ and $\mathbf{LP}^{\mathrm{R}}$
2.	Apply bound reduction and a round of linearization cuts to both $\mathbf{LP}^{\mathrm{L}}$ and $\mathbf{LP}^{\mathrm{R}}$
	$\Rightarrow$ get $\mathbf{SLP}^{\mathrm{L}}(A', a')$ and $\mathbf{SLP}^{\mathrm{R}}(A'', a'')$
3.	Construct CGLP as in (4.1)
4.	Solve CGLP, obtain (α, α_0)
5.	Return (α, α_0)

TABLE 1
Procedure for generating a disjunctive cut for problem **P**.

generation library [31] for integer disjunctions, we were not able to plug in the refinement and the bound reduction steps, therefore we have chosen to generate this type of disjunctive cuts within our own procedure.

Rank of the generated inequalities. Suppose a disjunctive cut $\alpha x \leq \alpha_0$ for a disjunction on variable x_i has been separated, and another disjunction on variable x_k is at hand. In the setting used in our tests, we do not add $\alpha x \leq \alpha_0$ to the LP relaxation that will be used to generate the next cut, as that would increase the rank of the cuts and possibly introduce numerical errors. Although there is such an option in COUENNE, we did not test it yet. This means, together with the maximum node depth of 10 mentioned above, that the maximum rank of the cuts we separate is 10.

Implementation details. The MINLP solver framework of choice is COUENNE[2] [7], an Open Source software package included in the Coin-OR infrastructure [30]. It implements standard reformulation, linearization, and bound reduction methods, as well as a *reliability branching* scheme [8].

6. Experimental results. In order to assess the utility and effectiveness of disjunctive cuts for MINLP, we have performed a battery of tests on a set of 84 publicly available MINLP instances from the following repositories:

- *MacMINLP* [26] and *minlplib* [11]: a collection of MINLP instances, both convex and non-convex;
- *nConv:* a collection of non-convex MINLPs[3];
- *MIQQP:* Mixed-Integer quadratically constrained quadratic programs [35]; model `qpsi.mod` was used;
- *globallib:* a collection of continuous NLP problems [19];

[2]See `http://www.coin-or.org/Couenne`
[3]IBM/CMU MINLP project, see `http://egon.cheme.cmu.edu/ibm/page.htm`

- *boxQP:* continuous, non-convex, box-constrained quadratic problems; the smaller instances are from [54] and those with more than 60 variables are from [10];
- *airCond*, a 2D bin-packing problem for air conduct design [18].

Table 3 describes the parameters of each instance: number of variables (*var*), of integer variables (*ivar*), of constraints (*con*), and of auxiliary variables (*aux*), or, in the notation used above: n, r, m, and q. The latter parameter is a good indicator of the size of the LP relaxation, as it is proportional to the number of linearization inequalities added.

We have conducted tests to compare two distinct branching techniques with disjunctive cuts, in order to understand what combination is most effective. The following four variants of COUENNE have been tested:

- V0: no disjunctive cuts, and the basic branching scheme **br-plain** described in [8], Section 5.1;
- RB: no disjunctive cuts, and an extension of reliability branching [1] to MINLP denoted **int-br-rev** in [8];
- DC: disjunctive cuts separated until depth 10 of the BB tree and the **br-plain** branching scheme;
- DC+RB: disjunctive cuts separated until depth 10 of the BB tree and reliability branching.

The latter variant, apart from being a combination of more sophisticated methods, has a further advantage: since reliability branching is a method to rank branching rules, disjunctive cuts are separated only on the most promising disjunctions.

All tests were performed on a 2.66GHz processor with 64GB of RAM memory and Linux kernel 2.6.29. COUENNE version 0.2, compiled with *gcc 4.4.0* was used. A time limit of two hours was set for all variants.

Table 2 summarizes our results by pointing out what variants perform better overall. For each variant, column "solved" reports the number of instances solved by that variant before the time limit, while column "best time" reports the number of solved instances whose CPU time is best among the four variants, or at most 10% greater than the best time. An analogous measure is given in the third column, "best nodes," for the BB nodes. The last column, "best gap," refers to instances that were not solved by any of the variants, and reports for how many of these the variant had the best gap, or within 10% of the best.

Although this gives a somewhat limited perspective, it shows that the variants with disjunctive cuts, especially the one coupled with reliability branching, have an edge over the remaining variants. They in fact allow to solve more problems within the time limit, on average, and, even when a problem is too difficult to solve, the remaining gap is smaller more often when using disjunctive cuts.

Tables 4 and 5 report in detail the comparison between the four variants of COUENNE. If an algorithm solved an instance in less than two hours, its CPU time in seconds is reported. Otherwise, the *remaining gap*

	<2h			Unsolved
Variant	solved	best time	best nodes	best gap
v0	26	12	11	31
RB	26	13	11	**35**
DC	**35**	8	13	29
DC+RB	**39**	**20**	**29**	**37**

TABLE 2
Summary of the comparison between the four variants. The first three columns are on instances that could be solved within the time limit of two hours, and report: the number of instances solved ("solved"), the number of instances for which the variant obtained the best CPU time or within the 10% of the best ("best time"), and analogously for the number of BB nodes ("best nodes"). The last column, "best gap," reports the number of instances for which a variant obtained the best performance, or within 10% of the best, in terms of the remaining gap.

is reported as the following function:

$$\text{gap} = \frac{z_{\text{best}} - z_{\text{lower}}}{z_{\text{best}} - z_0}, \tag{6.1}$$

where z_{best} is the objective value of the best solution found by the four algorithms, z_{lower} is the lower bound obtained by this algorithm, and z_0 is the initial lower bound found by COUENNE, which is the same for all four variants. If no feasible solution was found by any variant, the lower bound is reported in brackets. The best performances are highlighted in bold.

Variants with disjunctive cuts seem to outperform the others, especially for the *boxQP* instances. For some instances, however, disjunctive cuts only seem to slow down the execution of the BB as the CPU time spent in separation does not produce any effective cut. The tradeoff between the effectiveness of the disjunctive cuts and the time spent generating them suggests that a faster cut generation would increase the advantage.

We further emphasize this fact by showing the amount of time spent by reliability branching and disjunctive cuts, which is included in the total CPU time reported. Tables 6 and 7 show, for a subset of instances for which branching time or separation time were relatively large (at least 500 seconds), the CPU time spent in both processes and the number of resulting cuts or BB nodes. This selection of instances shows that, in certain cases, the benefit of disjunctive cuts is worth the CPU time spent in the generation. This holds true especially for the *boxQP* instances, where a large amount of time is spent in generating disjunctive cuts but where these result in better bounds or CPU time. Again, the fact that the current separation algorithm is rather simple suggests that a more efficient implementation would obtain the same benefit in shorter time.

We also graphically represent the performance of the four algorithms using performance profiles [15]. Figure 5(a) depicts a comparison on the CPU time. This performance profile displays data about all instances that could be solved in less than two hours by at least one of the variants. Hence, it also compares the quality of a variant in terms of number of instances solved. Figure 5(b) is a performance profile on the number of nodes.

Figure 5(c) is a performance profile on the remaining gap, and reports on all instances for which *none* of the variants could obtain the optimal solution in two hours or less. Note that this is a slightly different version of a performance profile: rather than the *ratio* between gaps, as required by the definition of performance profile, this graph shows, for each algorithm, the number of instances (plotted on the y axis) with remaining gap below the corresponding entry on the x axis.

The three graphs show once again that, for the set of instances we have considered, using both reliability branching and disjunctive cuts pays off for both easy and difficult MINLP instances. The former are solved in shorter time, while for the latter we yield a better lower bound.

7. Concluding remarks. Disjunctive cuts are effective in MINLP solvers as they are in MILP. Although they are generated from an LP relaxation of a non-convex MINLP, they can dramatically improve the lower bound and hence the performance of a branch-and-bound method.

One disadvantage of the CGLP procedure, namely having to solve a large LP in order to obtain one single cut, carries over to the MINLP case. Some algorithms have been developed, for the MILP case, to overcome this issue [3, 6]. It remains to be seen whether their extension to the MINLP case can be as straightforward.

Acknowledgments. The author warmly thanks François Margot for all the useful discussions that led to the development of this work.

Part of this research was conducted while the author was a Postdoctoral Fellow at the Tepper School of Business, Carnegie Mellon University, Pittsburgh PA.

REFERENCES

[1] T. Achterberg, T. Koch, and A. Martin, *Branching rules revisited*, OR Letters, 33 (2005), pp. 42–54.

[2] E. Balas, *Disjunctive programming: Properties of the convex hull of feasible points*, Discrete Applied Mathematics, 89 (1998), pp. 3–44.

[3] E. Balas and P. Bonami, *New variants of Lift-and-Project cut generation from the LP tableau: Open Source implementation and testing*, in Integer Programming and Combinatorial Optimization, vol. 4513 of Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 2007, pp. 89–103.

[4] E. Balas, S. Ceria, and G. Cornuéjols, *A lift-and-project cutting plane algorithm for mixed 0-1 programs*, Mathematical Programming, 58 (1993), pp. 295–324.

[5] ———, *Mixed 0-1 programming by lift-and-project in a branch-and-cut framework*, Management Science, 42 (1996), pp. 1229–1246.

[6] E. Balas and M. Perregaard, *A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming*, Mathematical Programming, 94 (2003), pp. 221–245.

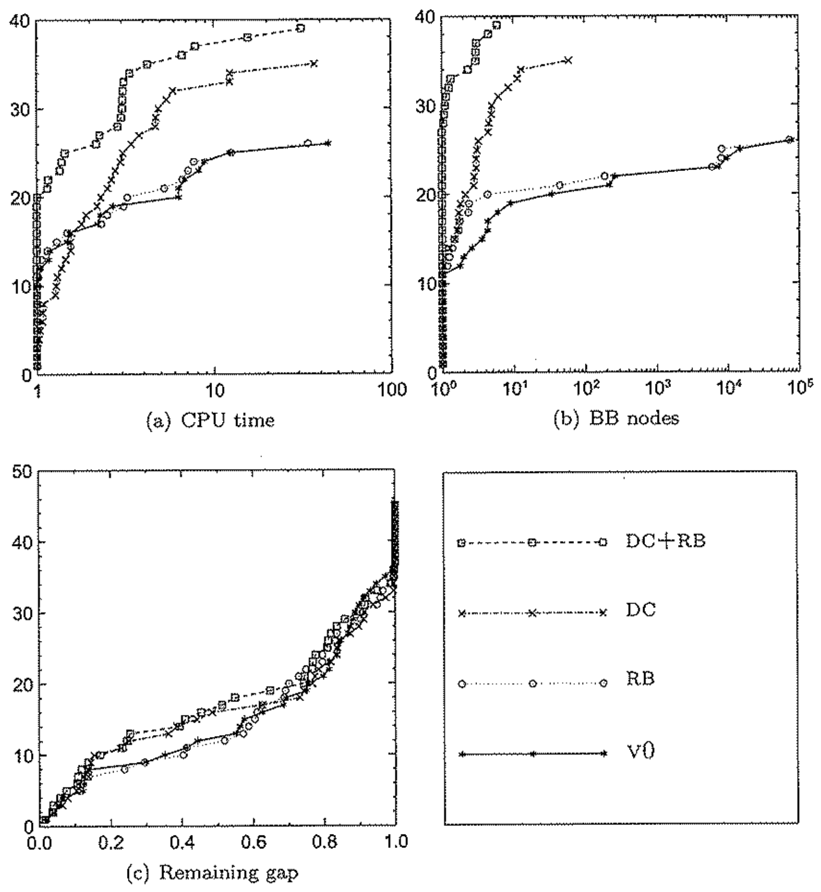


FIG. 5. *Performance profiles for the four variants of* COUENNE.

[7] P. BELOTTI, COUENNE*: a user's manual*, tech. rep., Lehigh University, 2009.
[8] P. BELOTTI, J. LEE, L. LIBERTI, F. MARGOT, AND A. WÄCHTER, *Branching and bounds tightening techniques for non-convex MINLP*, Optimization Methods and Software, accepted (2009).
[9] L. T. BIEGLER, I. E. GROSSMANN, AND A. W. WESTERBERG, *Systematic Methods of Chemical Process Design*, Prentice Hall, Upper Saddle River (NJ), 1997.
[10] S. BURER AND D. VANDENBUSSCHE, *Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound*, Comput. Optim. Appl., 43 (2009), pp. 181–195.
[11] M. R. BUSSIECK, A. S. DRUD, AND A. MEERAUS, *MINLPLib – a collection of test models for mixed-integer nonlinear programming*, INFORMS Journal of Computing, 15 (2003), pp. 114–119. http://www.gamsworld.org/minlp/minlplib/minlpstat.htm.
[12] A. CAPRARA AND M. LOCATELLI, *Global optimization problems and domain reduc-*

tion strategies, Mathematical Programming, (2009).

[13] E. Carrizosa, P. Hansen, and F. Messine, *Improving interval analysis bounds by translations*, Journal of Global Optimization, 29 (2004), pp. 157–172.

[14] G. Cornuéjols and R. Tütüncü, *Optimization Methods in Finance*, Cambridge University Press, Cambridge, 2006.

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

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

[17] C. A. Floudas, *Global optimization in design and control of chemical process systems*, Journal of Process Control, 10 (2001), pp. 125–134.

[18] A. Fügenschuh and L. Schewe, *Solving a nonlinear mixed-integer sheet metal design problem with linear approximations*, work in progress.

[19] GAMS Development Corp., *Gamsworld global optimization library*. `http://www.gamsworld.org/global/globallib/globalstat.htm`.

[20] E. Hansen, *Global Optimization Using Interval Analysis*, Marcel Dekker, Inc., New York, 1992.

[21] C. Helmberg and F. Rendl, *Solving quadratic (0,1)-problems by semidefinite programs and cutting planes*, Math. Prog., 82 (1998), pp. 291–315.

[22] J. Hooker, *Logic-based Methods for Optimization: Combining Optimization and Constraint Satisfaction*, Wiley, New York, 2000.

[23] R. Horst and H. Tuy, *Global Optimization: Deterministic Approaches*, Springer Verlag, Berlin, 1996.

[24] B. Kalantari and J. B. Rosen, *An algorithm for global minimization of linearly constrained concave quadratic functions*, Mathematics of Operations Research, 12 (1987), pp. 544–561.

[25] J. Kallrath, *Solving planning and design problems in the process industry using mixed integer and global optimization*, Annals of Operations Research, 140 (2005), pp. 339–373.

[26] S. Leyffer, *MacMINLP: AMPL collection of MINLPs*. `http://www-unix.mcs.anl.gov/~leyffer/MacMINLP`.

[27] L. Liberti, C. Lavor, and N. Maculan, *A branch-and-prune algorithm for the molecular distance geometry problem*, International Transactions in Operational Research, 15 (2008), pp. 1–17.

[28] L. Liberti, C. Lavor, M. A. C. Nascimento, and N. Maculan, *Reformulation in mathematical programming: an application to quantum chemistry*, Discrete Applied Mathematics, 157 (2009), pp. 1309–1318.

[29] M. L. Liu, N. V. Sahinidis, and J. P. Shectman, *Planning of chemical process networks via global concave minimization*, in Global Optimization in Engineering Design, I. Grossmann, ed., Springer, Boston, 1996, pp. 195–230.

[30] R. Lougee-Heimer, *The Common Optimization INterface for Operations Research*, IBM Journal of Research and Development, 47 (2004), pp. 57–66.

[31] R. Lougee-Heimer, *Cut generation library*. `http://projects.coin-or.org/Cgl`, 2006.

[32] Z. Q. Luo, J. S. Pang, and D. Ralph, *Mathematical Programming with Equilibrium Constraints*, Cambridge University Press, UK, 1996.

[33] A. Mahajan and T. K. Ralphs, *Experiments with branching using general disjunctions*, in Proceedings of the Eleventh INFORMS Computing Society Meeting, 2009. To Appear.

[34] F. Messine, *Deterministic global optimization using interval constraint propagation techniques*, RAIRO-RO, 38 (2004), pp. 277–294.

[35] H. Mittelmann, *A collection of mixed integer quadratically constrained quadratic programs*. `http://plato.asu.edu/ftp/ampl_files/miqp_ampl`.

[36] R. E. Moore, *Methods and Applications of Interval Analysis*, Siam, Philadelphia, 1979.

[37] J. H. OWEN AND S. MEHROTRA, *A disjunctive cutting plane procedure for general mixed-integer linear programs*, Mathematical Programming, 89 (2001), pp. 437–448.

[38] A. T. PHILLIPS AND J. B. ROSEN, *A quadratic assignment formulation of the molecular conformation problem*, tech. rep., CSD, Univ. of Minnesota, 1998.

[39] I. QUESADA AND I. E. GROSSMANN, *Global optimization of bilinear process networks and multicomponent flows*, Computers & Chemical Engineering, 19 (1995), pp. 1219–1242.

[40] H. RATSCHEK AND J. ROKNE, *Interval methods*, in Handbook of Global Optimization, R. Horst and P. M. Pardalos, eds., vol. 1, Kluwer Academic Publishers, Dordrecht, 1995, pp. 751–828.

[41] F. RENDL AND R. SOTIROV, *Bounds for the quadratic assignment problem using the bundle method*, Mathematical Programming, 109 (2007), pp. 505–524.

[42] H. S. RYOO AND N. V. SAHINIDIS, *Global optimization of nonconvex NLPs and MINLPs with applications in process design*, Computers & Chemical Engineering, 19 (1995), pp. 551–566.

[43] ———, *A branch-and-reduce approach to global optimization*, Journal of Global Optimization, 8 (1996), pp. 107–138.

[44] ———, *Global optimization of multiplicative programs*, Journal of Global Optimization, 26 (2003), pp. 387–418.

[45] N. V. SAHINIDIS, *Global optimization and constraint satisfaction: the branch-and-reduce approach*, in Global Optimization and Constraint Satisfaction, C. Bliek, C. Jermann, and A. Neumaier, eds., vol. 2861 of Lecture Notes in Computer Science, Springer, 2003, pp. 1–16.

[46] A. SAXENA, P. BONAMI, AND J. LEE, *Convex relaxations of non-convex mixed integer quadratically constrained programs: Projected formulations*, November 2008. Optimization Online.

[47] ———, *Disjunctive cuts for non-convex mixed integer quadratically constrained programs*, in Proceedings of the 13th Integer Programming and Combinatorial Optimization Conference, A. Lodi, A. Panconesi, and G. Rinaldi, eds., vol. 5035 of Lecture Notes in Computer Science, 2008, pp. 17–33.

[48] J. P. SHECTMAN AND N. V. SAHINIDIS, *A finite algorithm for global minimization of separable concave programs*, Journal of Global Optimization, 12 (1998), pp. 1–36.

[49] E. M. B. SMITH, *On the Optimal Design of Continuous Processes*, PhD thesis, Imperial College of Science, Technology and Medicine, University of London, Oct. 1996.

[50] E. M. B. SMITH AND C. C. PANTELIDES, *A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs*, Computers & Chem. Eng., 23 (1999), pp. 457–478.

[51] M. TAWARMALANI AND N. V. SAHINIDIS, *Convexification and global optimization in continuous and mixed-integer nonlinear programming: Theory, algorithms, software and applications*, vol. 65 of Nonconvex Optimization and Its Applications, Kluwer Academic Publishers, Dordrecht, 2002.

[52] ———, *Global optimization of mixed-integer nonlinear programs: A theoretical and computational study*, Mathematical Programming, 99 (2004), pp. 563–591.

[53] P. VAN HENTENRYCK, L. MICHEL, AND Y. DEVILLE, *Numerica, a Modeling Language for Global Optimization*, MIT Press, Cambridge, MA, 1997.

[54] D. VANDENBUSSCHE AND G. L. NEMHAUSER, *A branch-and-cut algorithm for nonconvex quadratic programs with box constraints*, Math. Prog., 102 (2005), pp. 559–575.

Name	var	aux	Name	var	ivar	con	aux	Name	var	ivar	con	aux
boxQP			*globallib*					*minlplib*				
sp020-100-1	20	206	catmix100	301	0	200	800	waterz	195	126	137	146
sp030-060-1	30	265	lnts100	499	0	400	1004	ravem	111	53	186	189
sp030-070-1	30	311	camsh200	399	0	400	600	ravempb	111	53	186	189
sp030-080-1	30	368	qp2	50	0	2	1277	enpro56	128	73	192	188
sp030-090-1	30	402	qp3	100	0	52	52	enpro56pb	128	73	192	188
sp030-100-1	30	457	catmix200	601	0	400	1600	csched2	401	308	138	217
sp040-030-1	40	234	turkey	512	0	278	187	water4	195	126	137	146
sp040-040-1	40	319	qp1	50	0	2	1277	enpro48	154	92	215	206
sp040-050-1	40	399	elec50	150	0	50	11226	enpro48pb	154	92	215	206
sp040-060-1	40	478	camsh400	799	0	800	1200	space25a	383	240	201	119
sp040-070-1	40	560	arki0002	2456	0	1976	4827	contvar	279	87	279	747
sp040-080-1	40	648	polygon50	98	0	1273	6074	space25	893	750	235	136
sp040-090-1	40	715	arki0019	510	0	1	4488	lop97icx	986	899	87	407
sp040-100-1	40	806	arki0015	1892	0	1408	2659	du-opt5	18	11	6	221
sp050-030-1	50	366	infeas1	272	0	342	2866	du-opt	20	13	8	222
sp050-040-1	50	498	lnts400	1999	0	1600	4004	waste	1425	400	1882	2298
sp050-050-1	50	636	camsh800	1599	0	1600	2400	lop97ic	1626	1539	87	4241
sp060-020-1	60	354	arki0010	3115	0	2890	1976	qapw	450	225	255	227
sp070-025-1	70	618	*nConv*					*MacMINLP*				
sp070-050-1	70	1227	c-sched47	233	140	138	217	trimlon4	24	24	24	41
sp070-075-1	70	1838	synheatmod	53	12	61	148	trimlon5	35	35	30	56
sp080-025-1	80	789	JoseSEN5c	987	38	1215	1845	trimlon6	168	168	72	217
sp080-050-1	80	1625	*MIQQP*					trimlon7	63	63	42	92
sp080-075-1	80	2388	ivalues	404	202	1	3802	space-25-r	843	750	160	111
sp090-025-1	90	1012	imisc07	519	259	212	696	space-25	893	750	235	136
sp090-050-1	90	2021	ibc1	2003	252	1913	1630	trimlon12	168	168	72	217
sp090-075-1	90	3033	iswath2	8617	2213	483	4807	space-960-i	5537	960	6497	3614
sp100-025-1	100	1251	imas284	301	150	68	366	*misc.*				
sp100-050-1	100	2520	ieilD76	3796	1898	75	3794	airCond	102	80	156	157
sp100-075-1	100	3728										

TABLE 3

Instances used in our tests. For each instance, "var" is the number of variables, "ivar" the number of integer variables, "con" the number of constraints, and "aux" the number of auxiliary variables generated at the reformulation step. Instances in the boxQP *group are continuous and only constrained by a bounding box, hence columns "ivar" and "con" are omitted.*

Name	V0	RB	DC	DC+RB	Name	V0	RB	DC	DC+RB
		boxQP					*globallib*		
sp020-100-1	57	70	21	9	catmix100	9	10	14	13
sp030-060-1	1925	1864	503	**280**	lnts100	30.0%	29.6%	**15.4%**	25.4%
sp030-070-1	3.4%	2.2%	1129	**718**	camsh200	79.7%	**61.1%**	78.5%	64.7%
sp030-080-1	13.7%	10.7%	**1406**	**1342**	qp2	12.5%	11.3%	13.8%	**10.5%**
sp030-090-1	2591	1662	695	**316**	qp3	6.6%	5.5%	6.5%	5.9%
sp030-100-1	12.9%	11.3%	**2169**	**1654**	catmix200	53	57	57	53
sp040-030-1	74	60	9	8	turkey	114	127	110	127
sp040-040-1	3.1%	5.2%	**393**	**308**	qp1	12.4%	12.2%	14.4%	**11.8%**
sp040-050-1	4.5%	8.2%	703	**370**	elec50	(353.6)	(353.6)	(353.6)	(353.6)
sp040-060-1	39.4%	36.3%	6467	**4145**	camsh400	84.7%	72.9%	84.1%	80.8%
sp040-070-1	27.5%	24.5%	2746	**1090**	arki0002	(0)	(0)	(0)	(0)
sp040-080-1	39.6%	40.4%	6.4%	**6389**	polygon50	(-20.2)	(-20.2)	(-20.2)	(-20.2)
sp040-090-1	41.2%	41.4%	7.9%	**1.7%**	arki0019	13.7%	13.7%	13.7%	13.7%
sp040-100-1	44.3%	40.4%	10.4%	**7.6%**	arki0015	83.6%	83.6%	83.6%	83.6%
sp050-030-1	354	361	**31**	**29**	infeas1	62.6%	62.6%	62.5%	**54.9%**
sp050-040-1	29.6%	28.5%	1669	**632**	lnts400	**5.7%**	**5.7%**	10.8%	10.8%
sp050-050-1	57.5%	57.2%	22.8%	**17.0%**	camsh800	**87.3%**	95.9%	97.4%	95.0%
sp060-020-1	1241	953	**36**	**28**	arki0010	1622	1538	2126	2079
sp070-025-1	40.1%	40.8%	2292	**824**			*nConv*		
sp070-050-1	69.3%	70.1%	**36.4%**	45.3%	c-sched47	4.2%	**0.8%**	4.0%	3.9%
sp070-075-1	81.4%	79.5%	76.7%	76.8%	synheatmod	1.2%	0.0%	14.8%	**141**
sp080-025-1	53.4%	49.5%	4715	**1241**	JoseSEN5c	**56.4%**	100.0%	100.0%	99.8%
sp080-050-1	81.6%	81.2%	74.4%	74.4%			*MIQQP*		
sp080-075-1	86.9%	88.7%	82.0%	81.2%	ivalues	**12.1%**	23.8%	25.1%	23.2%
sp090-025-1	68.6%	69.1%	38.6%	**24.5%**	imisc07	88.7%	**68.8%**	99.6%	76.6%
sp090-050-1	83.9%	83.4%	77.4%	77.6%	ibc1	(0.787)	(0.787)	(0.796)	**(0.813)**
sp090-075-1	94.8%	94.9%	87.4%	91.7%	iswath2	99.4%	100.0%	99.7%	**98.8%**
sp100-025-1	75.6%	75.0%	48.6%	**40.9%**	imas284	5007	**2273**	54.7%	6928
sp100-050-1	89.8%	90.9%	83.7%	82.0%	ieilD76	**35.2%**	99.0%	99.8%	99.6%
sp100-075-1	97.3%	96.6%	91.4%	91.4%					

TABLE 4

Comparison between the four methods. Each entry is either the CPU time, in seconds, taken to solve the instance or, if greater than two hours, the remaining gap (6.1) after two hours. If the remaining gap cannot be computed due to the lack of a feasible solution, the lower bound, in brackets, is shown instead.

Name	V0	RB	DC	DC+RB
		minlplib		
waterz	75.1%	**58.7%**	73.2%	85.9%
ravem	16	23	74	66
ravempb	18	24	55	42
enpro56	39	26	156	76
enpro56pb	55	24	301	81
csched2	2.2%	3.9%	**1.2%**	3.6%
water4	55.2%	52.0%	**44.2%**	51.3%
enpro48	58	37	204	114
enpro48pb	49	41	201	126
space25a	**(116.4)**	(107.2)	(107.3)	(100.1)
contvar	91.1%	90.2%	91.1%	**89.9%**
space25	(98.0)	(96.7)	(97.1)	**(177.6)**
lop97icx	93.0%	79.6%	93.9%	**39.3%**
du-opt5	73	170	251	159
du-opt	87	270	136	262
waste	(255.4)	**(439.3)**	(273.7)	(281.6)
lop97ic	(2543.5)	(2532.8)	(2539.4)	(2556.7)
qapw	(0)	**(20482)**	(0)	**(20482)**
		MacMINLP		
trimlon4	23	**4**	133	24
trimlon5	48.1%	**46**	35.6%	142
trimlon6	(16.17)	(18.69)	(16.18)	(18.52)
trimlon7	88.1%	**60.5%**	89.8%	74.3%
space-25-r	(74.2)	(68.6)	(75.0)	(69.6)
space-25	**(98.4)**	(89.6)	(96.3)	(91.9)
trimlon12	(16.1)	(18.6)	(16.1)	(18.5)
space-960-i	(6.5e+6)	(6.5e+6)	(6.5e+6)	(6.5e+6)
		misc.		
airCond	**187**	0.9%	876	1471

TABLE 5

Comparison between the four methods. Each entry is either the CPU time, in seconds, taken to solve the instance or, if greater than two hours, the remaining gap (6.1) after two hours.

	RB		DC		DC+RB			
Name	t_{br}	nodes	t_{sep}	cuts	t_{sep}	cuts	t_{br}	nodes
				boxQP				
sp040-060-1	8	340k	3104	17939	2286	11781	56	3k
sp040-070-1	10	277k	1510	5494	466	2042	56	277k
sp040-080-1	14	174k	4030	15289	3813	14831	95	4k
sp040-090-1	58	136k	4142	13733	3936	14604	115	3k
sp040-100-1	17	178k	4317	11415	3882	11959	126	2k
sp050-050-1	12	289k	4603	10842	4412	11114	127	6k
sp070-025-1	38	308k	1324	2215	383	921	28	308k
sp070-050-1	38	80k	5027	3823	4616	3622	477	80k
sp070-075-1	81	26k	6125	1220	5951	1089	46	26k
sp080-025-1	27	222k	2873	2818	716	1154	28	222k
sp080-050-1	66	35k	5888	1864	5561	1740	85	35k
sp080-075-1	119	4k	6449	720	6464	615	32	4k
sp090-025-1	32	170k	4672	3540	4386	3044	500	170k
sp090-050-1	86	26k	6064	1023	6335	838	37	26k
sp090-075-1	194	4k	6468	467	6862	250	8	4k
sp100-025-1	54	80k	5286	2990	4704	2372	659	80k
sp100-050-1	140	4k	6376	693	6363	733	34	4k
sp100-075-1	272	35k	6852	312	6835	302	3	35k
				globallib				
lnts100	6084	4k	3234	16	1195	6	4647	3k
camsh200	6242	10k	1772	2303	2123	2645	4192	9k
qp2	214	62k	2662	3505	1351	1662	444	41k
qp3	1298	803k	43	737	47	644	3272	484k
qp1	189	63k	3160	3525	1609	1736	240	45k
elec50	5677	45k	5494	45	5159	68	768	45k
camsh400	3494	33k	3433	818	5242	1263	560	53k
arki0002	6834	53k	3571	237	1641	181	5277	53k
arki0019	5761	53k	1846	39	1543	36	3812	53k
arki0015	2442	2k	2141	215	1442	106	2412	2k
infeas1	1306	2k	1495	45	1273	131	1397	2k
lnts400	457	2k	4767	1	4724	0	373	2k
camsh800	5001	23k	3671	188	2208	100	3172	23k

TABLE 6

Comparison of time spent in the separation of disjunctive cuts (t_{sep}) and in reliability branching (t_{br}). Also reported is the number of nodes ("nodes") and of disjunctive cuts generated ("cuts").

	RB		DC		DC+RB			
Name	t_{br}	nodes	t_{sep}	cuts	t_{sep}	cuts	t_{br}	nodes
				nConv				
JoseSEN5c	5166	1061k	4280	39	341	5	5315	1061k
				MIQQP				
ivalues	2816	10k	4223	700	4589	733	1571	10k
imisc07	6005	2k	6835	2621	5475	2131	1404	2k
ibc1	310	2k	6166	38	6368	167	46	2k
iswath2	827	2k	6567	31	6526	52	5	2k
imas284	443	62k	6769	1408	4474	700	381	72k
ieilD76	2934	9k	6994	75	6869	56	804	9k
				minlplib				
contvar	4284	2k	11	19	13	21	4706	3k
space25	272	1051k	2133	706	178	309	311	1100k
lop97icx	6540	14k	1142	353	4226	2603	2649	3k
waste	5204	13k	7090	715	6934	253	82	13k
lop97ic	3178	11k	4556	25	6734	147	3547	11k
qapw	6400	61k	104	0	34	0	6367	60k
				MacMINLP				
trimlon6	8500	62k	58	50	558	2649	5944	63k
trimlon12	8432	62k	58	50	562	2649	5941	62k
space-960-i	5859	62k	2127	20	1624	0	4971	62k
				misc.				
airCond	7123	112k	293	1736	103	1526	39	541k

TABLE 7

(Continued) Comparison of time spent in the separation of disjunctive cuts (t_{sep}) and in reliability branching (t_{br}). Also reported is the number of nodes ("nodes") and of disjunctive cuts generated ("cuts").