DISJUNCTIVE CUTS FOR NONCONVEX MINLP

PIETRO BELOTTI*

Abstract. Mixed Integer Nonlinear Programming (MINLP) problems present two main challenges: the integrality of a subset of variables and nonconvex (nonlinear) objective function and constraints. Many exact solvers for MINLP are branch-and-bound algorithms that compute a lower bound on the optimal solution using a linear programming relaxation of the original problem.

In order to solve these problems to optimality, disjunctions can be used to partition the solution set or to obtain strong lower bounds on the optimal solution of the problem. In the MINLP context, the use of disjunctions for branching has been subject to intense research, while the practical utility of disjunctions as a means of generating valid linear inequalities has attracted some attention only recently.

We describe an application to MINLP of a well-known separation method for disjunctive cuts that has shown to be very effective in Mixed Integer Linear Programming (MILP). As the experimental results show, this application obtains encouraging results in the MINLP case even when a simple separation method is used.

Key words. Disjunctions, MINLP, Couenne, Disjunctive cuts.

AMS(MOS) subject classifications. 90C57

1. Motivation: nonconvex MINLP. Mixed integer nonlinear programming is a powerful modeling tool for very generally defined problems in optimization [32]. A MINLP problem is defined as follows:

$$\begin{array}{lll} (\mathbf{P}_0) \quad \min & f(x) & \\ \text{s.t.} & g_j(x) \leq 0 & \forall j = 1, 2 \dots, m \\ & \ell_i \leq x_i \leq u_i & \forall i = 1, 2 \dots, n \\ & x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}, & \end{array}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g_j : \mathbb{R}^n \to \mathbb{R}$, for all $j = 1, 2 \dots, m$, are in general multivariate, nonconvex functions; n is the number of variables, r is the number of integer variables, and x is the n-vector of variables, whose i-th component is denoted by x_i; and ℓ_i and u_i are lower and upper bounds on variable x_i. We assume that all these bounds are finite, as otherwise the problem is undecidable [33].

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., $+$, $-$, $*$, $/$, $\hat{}$). Note 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 $\mathbf{P}_0$ in Chemical Engineering [14, 26, 35] and Computational Biology [39, 40, 50], among others. Special subclasses of MINLP, such as Mixed Integer Linear Programming (MILP),

*Dept. of Mathematical Sciences, Clemson University (email: pbelott@clemson.edu)

where f is linear and all g_i's are affine, and convex MINLPs (i.e., MINLPs whose continuous relaxation is a convex nonlinear program), admit special (and more efficient) solvers, therefore the only reason to use a general-purpose nonconvex MINLP solver is that the problem cannot be classified as any of those special cases.

Effective algorithms for nonconvex optimization aim at finding a relaxation and obtaining a good lower bound on the optimal solution value. In the MILP case, a lower bound can be found by solving the LP relaxation obtained by relaxing integrality on the variables. In the convex MINLP case, relaxing integrality yields a convex nonlinear problem and hence a lower bound. In the general case, finding a relaxation and a lower bound on the global optimum for $\mathbf{P}_0$ can be hard, since relaxing integrality yields a nonconvex NLP.

Disjunctions. When the relaxation does not obtain a strong lower bound, an approach to strengthening the relaxation is to use logical *disjunctions* that are satisfied by all solutions of $\mathbf{P}_0$. In their most general form, disjunctions are logical operators that return true whenever one or more of their operands are true [5]. In this work, we consider disjunctions involving linear inequalities, although more general disjunctions are possible. Let us denote as S the feasible set of $\mathbf{P}_0$, i.e., $S = \{x \in \mathbb{Z}^r \times \mathbb{R}^{n-r} : \ell_i \leq x_i \leq u_i \, \forall i = 1, 2 \dots, n, g_j(x) \leq 0 \, \forall j = 1, 2 \dots, m\}$. A disjunction is an operator on a set of systems of inequalities over the set S, written as

$$\bigcup_{h \in Q} \{x \in \mathbb{R}^n : A^h x \leq b^h, x \in S\}, \tag{1.1}$$

where $A^h \in \mathbb{Q}^{m_h \times n}$, $b^h \in \mathbb{Q}^{m_h}$, m_h is the number of inequalities in the h-th system, and Q is an index set. We say that the disjunction is true if there exist $x \in S$ and $h \in Q$ such that $A^h x \leq b^h$.

The general definition (1.1) comprises several classes of disjunctions. Two important ones are the *integer disjunction* $x_i \leq \alpha \vee x_i \geq \alpha + 1$, which is valid for any $\alpha \in \mathbb{Z}$ if x_i is an integer variable; and the *spatial disjunction* $x_i \leq \beta \vee x_i \geq \beta$, with $\beta \in \mathbb{R}$ and x_i a continuous variable.

If disjunctions are used in the branching step or to generate disjunctive cuts, two main problems need to be addressed: *(i)* how to describe the set of disjunctions and *(ii)* what disjunction should be used among the (possibly infinite) set of valid disjunctions.

Branch-and-bound algorithms. The *branch* operation partitions the feasible set of an optimization problem by imposing a disjunction. Each

term of a disjunction (1.1) is associated with a subproblem $\mathbf{P}_h$ of the form

$$\begin{array}{lll} (\mathbf{P}_h) \ \min & f(x) & \\ \text{s.t.} & g_j(x) \leq 0 & \forall j = 1, 2 \dots, m \\ & A^h x \leq b^h & \\ & \ell_i \leq x_i \leq u_i & \forall i = 1, 2 \dots, n \\ & x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}, & \end{array} \tag{1.2}$$

in which the constraints (1.2) are those imposed by branching. We denote the feasible region of $\mathbf{P}_h$ by S_h.

A *branch-and-bound* (BB) algorithm applies this branching method recursively. Any MINLP with a bounded feasible region can be solved by a BB in a finite number of steps [32]. Note that an optimal solution of a subproblem $\mathbf{P}_h$ is not necessarily optimal for $\mathbf{P}_0$, since the feasible regions of individual subproblems are subsets of the original feasible region and may not include an optimal solution of $\mathbf{P}_0$. Nevertheless, solutions obtained in this way are still feasible and hence yield valid upper bounds on the optimal value of $f(x)$. An advantage of applying a branching method is that it allows to discard any subproblem whose associated lower bound exceeds the best upper bound found so far.

Disjunctive cuts. Disjunctions can also be imposed *indirectly* by deriving one or more implied constraints from them [5]. A valid inequality with respect to S consists of a pair $(\alpha, \alpha_0) \in \mathbb{Q}^{n+1}$ such that $\alpha^\top x \leq \alpha_0$ for all $x \in S$. An inequality that is valid for S and violated by at least one member of the feasible region of the relaxation of $\mathbf{P}_0$ is called a *cut*.

A *disjunctive cut* with respect to a disjunction of the form (1.1) is an inequality that is valid for S_h for each $h \in Q$, or equivalently, for the convex hull of the union of these sets. Amending disjunctive cuts to the feasible region of a relaxation of $\mathbf{P}_0$ is often an effective way of tightening a relaxation of S.

Disjunctive cuts have long been studied in the MILP context [7, 9, 49]. They have also been generated from disjunctions arising from MINLPs with a certain structure. For MINLP with binary variables and whose continuous relaxation is convex, Stubbs and Mehrotra [60] generalized the procedure proposed by Balas, Ceria and Cornuéjols [7] and described a separation procedure based on a convex optimization problem.

A specialized procedure has been successfully applied to Mixed Integer Conic Programming (MICP) problems, which are MILPs amended by a set of constraints whose feasible region is a cone in $\mathbb{R}^n$. The second order cone and the cone of symmetric semidefinite matrices are among the most important classes of conic constraints in this class. Çezik and Iyengar [19] and lately Drewes [25] proposed, for MICP problems where all disjunctions are generated from binary variables, an application of the procedure to the

conic case, where disjunctive inequalities are obtained by solving a continuous conic optimization problem. Analogously, Frangioni and Gentile [27] describe a class of disjunctive cuts that can be used in convex MINLPs where binary variables are used to model *semi-continuous* variables, i.e., variables that take values in the set $\{0\} \cup [l, u]$ with $0 \notin [l, u]$.

Another interesting class of problems is that of Mathematical Programs with Complementarity Constraints (MPCC) [57], i.e., MINLPs that contain nonconvex constraints $x^\top y = 0$, with $x \in \mathbb{R}^k_+$, $y \in \mathbb{R}^k_+$. These can be more easily stated as $x_i y_i = 0$ for all $i = 1, 2 \dots, k$, and give rise to simple disjunctions $x_i = 0 \vee y_i = 0$. Júdice et al. [34] study an MPCC where the only nonlinear constraints are the complementarity constraints, and therefore relaxing the latter yields an LP. Disjunctive cuts are generated from solutions of the LP that violate a complementarity constraint (i.e., solutions where $x_i > 0$ and $y_i > 0$ for some $i \in \{1, 2 \dots, k\}$) through the observation that both variables are basic and by applying standard disjunctive arguments to the corresponding tableau rows.

Scope of the paper and outline. While the application to MILP of cuts derived from disjunctions has been studied for decades and efficient implementations are available, their practical application to MINLP has recently attracted a lot of attention: a notable example is the work by Saxena et al. [55, 56], where disjunctive cuts are used to solve MINLPs with quadratic nonconvex objective and constraints.

In particular, [55] point out that " ... even though the results presented in this paper focussed on MIQCPs, they are equally applicable to a much wider class of nonconvex MINLPs. All we need is an automatic system that can take a nonconvex MINLP and extract a corresponding MIQCP relaxation. Development of software such as Couenne [11, 12] is a step in this direction."

Our contribution is an attempt to follow that indication: we present an application of disjunctive programming to the context of nonconvex MINLP problems. We provide an Open Source implementation of a procedure that generates disjunctive cuts for MINLP problems, and present a set of computational results that substantiate the practical utility of this application.

Several exact solution methods for nonconvex MINLP rely, in order to obtain valid lower bounds, on reformulation and linearization techniques [46, 59, 61], which we briefly introduce in the next section as their definition is essential to describe 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 based on the cut generating LP (CGLP) is discussed in Section 5. This procedure has been added to COUENNE, a general-purpose, Open Source solver for MINLP [11], and tested on a set of publicly available nonconvex MINLP instances. These tests are presented

in Section 6.

2. Lower bounds of an MINLP. Several 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 a set of new variables, a linear objective function, and a set of nonlinear equality constraints;
- *linearization*: each nonlinear equality constraint is relaxed by replacing it with a set of linear inequalities.

Here we provide a brief description whose purpose is to introduce the derivation of disjunctions in Section 3.

Reformulation. If f and all g_i's are factorable, they can be represented by *expression trees*, i.e., n-ary trees where every node is either a constant, a variable, or an n-ary operator whose arguments are represented as children of the node, and which are in turn expression trees [21, Chapter 3]. While all leaves of an expression tree are variables or constants, each non-leaf node, which is an operator of a finite set $\mathcal{O} = \{+, *, \hat{}, /, \sin, \cos, \exp, \log\}$, is associated with a new variable, denoted as *auxiliary variable*. As a result, we obtain a *reformulation* of $\mathbf{P}_0$ [46, 59, 61]:

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

where $X = \mathbb{Z}^r \times \mathbb{R}^{n-r} \times \mathbb{Z}^s \times \mathbb{R}^{q-s}$, that is, q auxiliary variables are introduced, s of which are integral and $q - s$ continuous[1]. The auxiliary x_{n+q} is associated to the operator that defines the objective function of $\mathbf{P}_0$. Each auxiliary x_k is associated with a function $\vartheta_k(x)$ from $\mathcal{O}$ and depends, in general, on one or more of the variables $x_1, x_2 \dots, x_{k-1}$. Let us define the bounding box $[\ell, u]$ of $\mathbf{P}$ as the set $\{x \in \mathbb{R}^{n+q} : \ell_i \le x_i \le u_i, i = 1, 2 \dots, n+q\}$.

In the reformulation, all nonlinear constraints are of the form $x_k = \vartheta_k(x)$ for all $k = n+1, n+2 \dots, n+q$, where ϑ_k is an operator from $\mathcal{O}$. It is then possible to obtain an LP relaxation of $\mathbf{P}$, or *linearization* for short, by applying operator-specific algorithms to each such constraint [59].

Linearization. Consider a nonlinear constraint $x_k = \vartheta_k(x)$ in the reformulation $\mathbf{P}$. The set $\Theta_k = \{x \in X \cap [\ell, u] : x_k = \vartheta_k(x)\}$ is, in general, nonconvex. The nonconvex set of feasible solutions of the reformulation is therefore $\Theta = \bigcap_{k=n+1}^{n+q} \Theta_k$.

While finding a convex relaxation of Θ might prove very difficult, obtaining a convex relaxation of each Θ_k, for all $k = n+1, n+2 \dots, n+q$, is

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

in general easier and has been extensively studied [46, 59, 61, 41]. Several exact MINLP solvers [54, 12, 42] seek an LP relaxation of Θ_k defined by the set $\mathbf{LP}_k = \{x \in [\ell, u] : B^k x \leq c^k\}$, with $B^k \in \mathbb{Q}^{m_k \times (n+q)}$, $c^k \in \mathbb{Q}^{m_k}$, and m_k is the number of inequalities of the relaxation. In general, m_k depends on ϑ_k and, for certain constraints, a larger m_k yields a better lower bound. However, in order to keep the size of the linearization limited, a relatively small value is used. For the linearization procedure we have used, $m_k \leq 4$ for all k. Let us denote $\mathbf{LP} = \bigcap_{k=n+1}^{n+q} \mathbf{LP}_k$. Because $\mathbf{LP}_k \supseteq \Theta_k$ for all $k = n+1, n+2 \dots, n+q$, we have $\mathbf{LP} \supseteq \Theta$. Hence, minimizing x_{n+q} over the convex set $\mathbf{LP}$ gives a lower bound on the optimal solution value of $\mathbf{P_0}$.

The aforementioned MINLP solvers are branch-and-bound procedures that obtain, at every BB node, a linearization of $\mathbf{P}$. It is crucial, as pointed out by McCormick [46], that for each $k = n+1, n+2 \dots, n+q$, the linearization of Θ_k be *exact* at the lower and upper bounds on the variables appearing as arguments of ϑ_k, i.e., the linearization must be such that if a solution $x^\star$ is feasible for the LP relaxation but not for $\mathbf{P}$, then there exists an i such that $\ell_i < x_i^\star < u_i$, so that a spatial disjunction $x_i \leq x_i^\star \vee x_i \geq x_i^\star$ can be used.

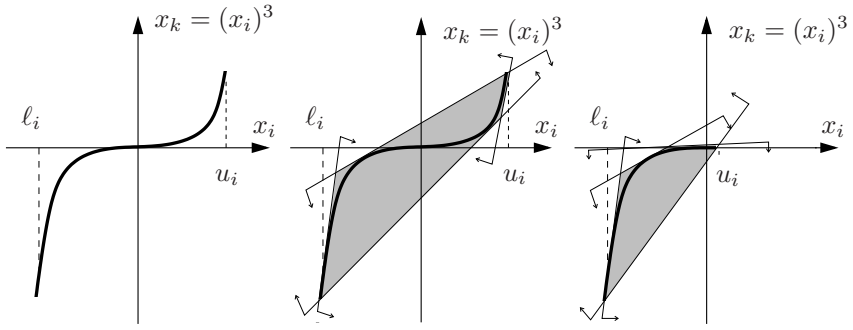
In general, for all $k = n+1, n+2 \dots, n+q$, both B^k and c^k depend on the variable bounds ℓ and u: the tighter the variable bounds, the stronger the lower bound obtained by minimizing x_{n+q} over $\mathbf{LP}$. For such an approach to work effectively, several *bound reduction* techniques have been developed [30, 48, 52, 51, 58, 38]. Most of these techniques use the nonconvex constraints of the reformulation or the linear constraints of the linearization, or a combination of both. An experimental comparison of bound reduction techniques used in a MINLP solver is given in [12].

Let us consider, as an example, a constraint $x_k = \vartheta_k(x) = (x_i)^3$, and the nonconvex set $\Theta_k = \{x \in \mathbb{R}^{n+q} \cap [\ell, u] : x_k = (x_i)^3\}$, whose projection on the (x_i, x_k) space is the bold curve in Figure 1(a). A linearization of Θ_k is in Figure 1(b), and is obtained through a procedure based on the function $\vartheta_k(x) = (x_i)^3$ and the bounds on x_i. As shown in Fig. 1(c), when tighter bounds are known for x_i a better relaxation can be obtained: the set $\Theta'_k = \{x \in \mathbb{R}^{n+q} \cap [\ell, u'] : x_k = (x_i)^3\}$, where $u'_j = u_j$ for $j \neq i$ and $u'_i < u_i$, admits a tighter linearization $\mathbf{LP}'$, which is a proper subset of $\mathbf{LP}$ since the linearization is exact at u'_i.

3. Disjunctions in MINLP. In order to apply the paradigm of disjunctive programming to nonconvex MINLP, we need *(i)* a description of the set of valid disjunctions in $\mathbf{P}$ and *(ii)* a procedure for selecting, from said set, a disjunction violated by an optimal solution $x^\star$ to the LP relaxation.

Deriving disjunctions. Nonconvex constraints in the reformulation $\mathbf{P}$ above are of two types:

- integrality of a subset of variables: x_i for $i \in \{1, 2 \dots, r, n+1, n+1 \dots, n+s\}$;
- nonconvex constraints $x_k = \vartheta_k(x)$ for $k = n+1, n+2 \dots, n+q$.



(a) Reformulated constraint

(b) A linearization

(c) A *refined* linearization

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

MILP problems and convex MINLPs contain nonconvex constraints of the first type only. When these constraints are relaxed, one obtains a continuous relaxation which yields a lower bound on the optimal solution value and a solution vector $x^\star$ that satisfies the convex constraints, but may violate the integrality requirements. If $x_i^\star \notin \mathbb{Z}$ for one of the integer variables, then $x^\star$ violates the *variable disjunction* $x_i \le \lfloor x_i^\star \rfloor \vee x_i \ge \lceil x_i^\star \rceil$.

There has been extensive research on how to use this type of disjunction, how to derive disjunctive cuts from them, and how to efficiently add such inequalities, using a BB method coupled with a cut generating procedure [7, 8, 49]. Several generalizations can be introduced at the MILP level: a well known example is the *split disjunction* $\pi x \le \pi_0 \vee \pi x \ge \pi_0 + 1$, where $(\pi, \pi_0) \in \mathbb{Z}^{p+1}$ and $x \in \mathbb{Z}^p$ is a vector of p integer variables [10, 17, 22, 36].

The reformulation of an MINLP problem is subject to both types of nonconvex constraints. The optimal solution to the LP relaxation, $x^\star$, may be infeasible for the integrality constraints or for one or more of the constraints of the reformulation, $x_k = \vartheta_k(x)$. In this work, we will primarily consider disjunctions arising from constraints of the second type, and will focus on the problem of finding a valid disjunction that is violated by a solution to the LP relaxation of $\mathbf{P}_0$.

For the sake of simplicity, consider a nonconvex constraint $x_k = \vartheta_k(x_i)$ with ϑ_k univariate and x_i continuous (disjunctions can be derived with a similar procedure when ϑ_k is multivariate and/or x_i is integer), and assume this constraint is violated by a solution $x^\star$ of the LP relaxation, i.e., $x_k^\star \ne \vartheta_k(x_i^\star)$. The *spatial disjunction*

$$x_i \le \beta \quad \vee \quad x_i \ge \beta, \tag{3.1}$$

although valid for any $\beta \in [\ell_i, u_i]$, is *not* violated by $x^\star$. This is a point of departure with MILP, for which disjunctions over integer variables are valid and, obviously, violated by fractional solutions.

A violated disjunction for MINLP, however, can be derived by applying bound reduction and the linearization procedure. Suppose that the linearization for the set $\Theta'_k = \{x \in X \cap [\ell, u] : x_i \leq \beta, x_k = \vartheta_k(x_i)\}$ is $\mathbf{LP}'_k = \{x \in [\ell, u] : B'x \leq c'\}$, and analogously, the linearization for $\Theta''_k = \{x \in X \cap [\ell, u] : x_i \geq \beta, x_k = \vartheta_k(x_i)\}$ is $\mathbf{LP}''_k = \{x \in [\ell, u] : B''x \leq c''\}$; we assume that the linear constraints include the new upper (resp. lower) bound β on x_i. Assuming that the two linearizations $\mathbf{LP}'_k$ and $\mathbf{LP}''_k$ are *exact* at the bounds on x_i, we have $x^\star \notin \mathbf{LP}'_k \cup \mathbf{LP}''_k$. Hence, the disjunction

$$x \in \mathbf{LP}'_k \quad \vee \quad x \in \mathbf{LP}''_k \tag{3.2}$$

is valid and violated by $x^\star$, and can be used to generate a disjunctive cut or a branching rule.

Although it might seem intuitive to set $\beta = x^\star_i$, in practice spatial disjunctions (3.1) are often selected such that $\beta \neq x^\star_i$, for instance because $x^\star_i$ is very close to one of the bounds on x_i. Even so, there are procedures to derive a valid disjunction (3.2) violated by $x^\star$. The rules for selecting a good value of β have been discussed, among others, in [12, 62] and will not be presented here.

As an example, consider the constraint $x_k = \vartheta_k(x_i) = (x_i)^2$ and the components $(x^\star_i, x^\star_k)$ of the LP solution $x^\star$ as in Figure 2(a). Since $x^\star$ lies in the convex hull of $\Theta_k = \{x \in \mathbb{R}^{n+q}, x \in [\ell, u], x_k = (x_i)^2\}$, refining the linearization, by adding linearization inequalities for other auxiliary variables, may not be sufficient to separate $x^\star$. In this case, a valid disjunction that is violated by $x^\star$ can be used to either create two subproblems or to derive a disjunctive cut. We observe that $(x^\star_i, x^\star_k)$ is not necessarily a vertex of the linearization of $x_k = (x_i)^2$ since it is simply a projection of $x^\star$ (which is a vertex of the LP relaxation) onto the plane (x_i, x_k). Starting from the valid disjunction $x_i \leq \beta \vee x_i \geq \beta$, one can create two linearizations $\mathbf{LP}'$ and $\mathbf{LP}''$, shown as shaded areas in Figure 2(b). The corresponding disjunction $x \in \mathbf{LP}' \vee x \in \mathbf{LP}''$ is valid and violated by $x^\star$.

Selecting a disjunction. In its simplest form, the problem of finding the most effective disjunction considers variable disjunctions only and can be stated as follows: given an optimal solution $x^\star$ of an LP relaxation of $\mathbf{P}$, an index set I of integer variables with fractional value in $x^\star$, and an index set N of violated nonconvex constraints of $\mathbf{P}$, i.e., $N = \{k \in \{n+1, n+1 \dots, n+q\} : x^\star_k \neq \vartheta_k(x^\star)\}$, choose either *(i)* an element $i \in I$ defining the integer disjunction $x_i \leq \lfloor x^\star_i \rfloor \vee x_i \geq \lceil x^\star_i \rceil$ or *(ii)* an element k of N, a variable x_i that is an argument of $\vartheta_k(x)$, and a scalar β defining the spatial disjunction $x_i \leq \beta \vee x_i \geq \beta$.

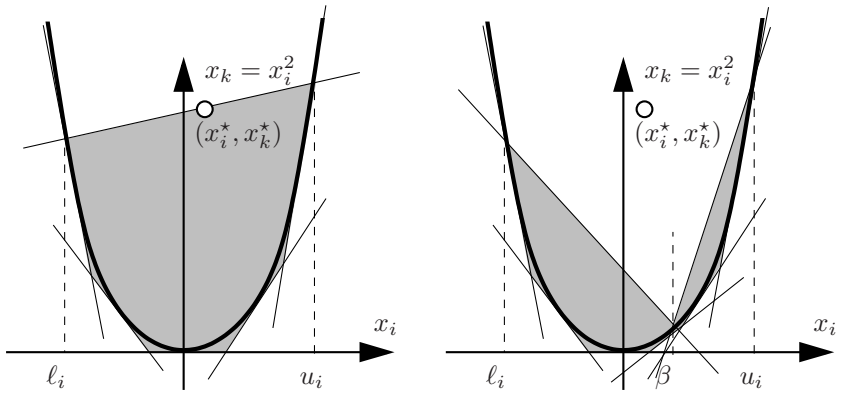
In order to select a disjunction, an *infeasibility* parameter, which depends on the solution $x^\star$ of the LP relaxation, is associated with each

variable x_i of the reformulation. A null infeasibility corresponds to a disjunction satisfied by $x^\star$, and a nonzero infeasibility corresponds to a disjunction violated by $x^\star$. Ideally, a large infeasibility should indicate a better disjunction, for example in terms of improvement in the lower bound of the two problems generated by branching on that disjunction. The infeasibility $\Omega_{\text{int}}(x_i)$ of an integer variable x_i is computed as $\min\{x_i^\star - \lfloor x_i^\star \rfloor, \lceil x_i^\star \rceil - x_i^\star\}$. For a variable x_j appearing in one or more nonlinear expressions, consider the set D_j of indices k of nonconvex constraints $x_k = \vartheta_k(x)$ where ϑ_k contains x_j as an argument, and define $\gamma_k = |x_k^\star - \vartheta_k(x^\star)|$ for all k in D_j. We define the infeasibility $\Omega_{\text{nl}}(x_i)$ of x_j as a convex combination of $\min_{k\in D_j} \gamma_k$, $\sum_{k\in D_j} \gamma_k$, and $\max_{k\in D_j} \gamma_k$. If x_j is also integer, the maximum between $\Omega_{\text{nl}}(x_i)$ and $\Omega_{\text{int}}(x_i)$ is used. We refer the reader to [12, §5] for a more detailed description, which is omitted here for the sake of conciseness.

A standard procedure for selecting disjunctions sorts them in nonincreasing order of infeasibility. When selecting a disjunction for branching, the one with maximum infeasibility is chosen. If a set of disjunctive cuts is desired, the first p disjunctions in the sorted list are chosen, for a given $p \geq 1$, and p disjunctive cuts are generated.

As discussed in [1, 12], this infeasibility measure may not be the best way to rank disjunctions. More sophisticated techniques have been proposed for disjunction selection in the branching process, for example strong branching [4], pseudocosts branching [13], reliability branching [1], and Violation Transfer [43, 62]. A generalization of reliability branching [1] to the MINLP case has been recently presented [12].

Disjunctions in special cases of MINLP. Some classes of MINLP problems exhibit nonconvex constraints that give rise to special disjunctions, and are used to generate disjunctive cuts. One such example has been studied for quadratically constrained quadratic programs (QCQPs) by Saxena et al. [55, 56]. These problems contain nonlinear terms of the form $x_i x_j$. Applying a reformulation such as that described in Section 2 would obtain auxiliary variables $y_{ij} = x_i x_j$ for $1 \leq i \leq j \leq n$. In matricial form, this corresponds to the nonconvex constraint $Y = xx^\top$, where Y is a symmetric, $n \times n$ matrix of auxiliary variables and x is the n-vector of variables (notice that there are $n(n+1)/2$ new variables instead of n^2, since $y_{ij} = y_{ji}$). Rather than applying a linearization to each nonconvex constraint $y_{ij} = x_i x_j$, such a constraint can be relaxed as $Y - xx^\top \succeq 0$, thus reducing a QCQP to a (convex) Semidefinite program, which yields comparably good lower bounds [31, 53, 3]. However, these SDP models are obtained from the original problem by relaxing the nonconvex constraint $xx^\top - Y \succeq 0$. In [55], this constraint is used to generate disjunctions: given a vector $v \in \mathbb{R}^n$, the non-convex constraint $(v^\top x)^2 \geq v^\top Y v$, which can be rewritten as $w^2 \geq z$ after a change of variables, is obtained from the negative eigenvalues of the matrix $\bar{x}\bar{x}^\top - \bar{Y}$, where $(\bar{x}, \bar{Y})$ is a solution to the relaxation. This is then used to generate a disjunction and disjunctive cuts



(a) Infeasible solution of a linearization

(b) Linearized subproblems

FIG. 2. *MINLP Disjunctions and nonconvex constraints. In (a), the shaded area is the linearization of the constraint $x_k = \vartheta_k(x_i)$ with $x_i \in [\ell_i, u_i]$, whereas $(x_i^\star, x_k^\star)$ is the value of x_i and x_k in the optimum of the LP relaxation. In (b), the spatial disjunction $x_i \leq \beta \vee x_i \geq \beta$ generates two sets $\Theta'_k = \{x \in X \cap [\ell, u] : x_k = x_i^2, x_i \leq \beta\}$ and $\Theta''_k = \{x \in X \cap [\ell, u] : x_k = x_i^2, x_i \geq \beta\}$. The corresponding linearizations, $\mathbf{LP}'_k$ and $\mathbf{LP}''_k$, are the smaller shaded areas.*

in x and Y. In a more recent development [56], this procedure is modified to generate disjunctive cuts in the original variables x only.

4. Disjunctive cuts in nonconvex MINLP. Assume that the linearization step produces an LP relaxation that we denote $\min\{x_{n+q} : Ax \leq a, \ell \leq x \leq u\}$, where $A \in \mathbb{Q}^{K \times (n+q)}$, $a \in \mathbb{Q}^K$, and K is the total number of linear inequalities generated for all of the nonlinear constraints $x_k = \vartheta_k(x)$ of the reformulation (we recall that $K \leq 4(n+q)$), while ℓ and u are the vectors of the lower and upper bounds on both original and auxiliary variables. Let us denote the feasible set of the linear relaxation as $\mathbf{LP} = \{x \in \mathbb{R}^{n+q} : Ax \leq a, \ell \leq x \leq u\}$.

Consider a disjunction $x_i \leq \beta \vee x_i \geq \beta$. The two LP relaxations

$$\mathbf{LP}' = \{x \in \mathbf{LP} : x_i \leq \beta\}$$
$$\mathbf{LP}'' = \{x \in \mathbf{LP} : x_i \geq \beta\},$$

are created by amending to $\mathbf{LP}$ one constraint of the disjunction. As pointed out in the previous section, the disjunction alone does not eliminate any solution from $\mathbf{LP}' \cup \mathbf{LP}''$, because $\mathbf{LP} = \mathbf{LP}' \cup \mathbf{LP}''$, while this does not hold for disjunctions on integer variables, where $\mathbf{LP}$ strictly contains the union of the two subproblems. Consider two problems $\mathbf{P}'$ and $\mathbf{P}''$, obtained by intersecting $\mathbf{P}$ with the constraints $x_i \leq \beta$ and $x_i \geq \beta$,

respectively. Apply bound reduction and the linearization procedure to $\mathbf{P}'$ and $\mathbf{P}''$, and denote the tightened problems as $\mathbf{SLP}'$ and $\mathbf{SLP}''$ (see Figure 2(b)). The two sets can be described as

$$\begin{aligned}\mathbf{SLP}' &= \{x \in \mathbf{LP} : B'x \leq c'\},\\ \mathbf{SLP}'' &= \{x \in \mathbf{LP} : B''x \leq c''\},\end{aligned}$$

where $B' \in \mathbb{Q}^{H' \times (n+q)}$, $c' \in \mathbb{Q}^{H'}$, $B'' \in \mathbb{Q}^{H'' \times (n+q)}$, and $c'' \in \mathbb{Q}^{H''}$ are the coefficient matrices and the right-hand side vectors of the inequalities added to the linearizations, which 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:

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

where

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

so as to include the initial linear constraints, the new linearizations inequalities, 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 [5], an inequality $\alpha^\top x \leq \alpha_0$, where $\alpha \in \mathbb{Q}^{n+q}$ and $\alpha_0 \in \mathbb{Q}$, is valid for the convex hull of $\mathbf{SLP}' \cup \mathbf{SLP}''$ if α and α_0 satisfy:

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

where $u \in \mathbb{R}^{K'}_+$ and $v \in \mathbb{R}^{K''}_+$. Given an LP relaxation and its optimal solution $x^\star$, an automatic procedure for generating a cut of this type consists of finding vectors u and v such that the corresponding cut is maximally violated [7]. This requires solving the Cut Generating Linear Programming (CGLP) 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. An optimal solution (more precisely, any solution with positive objective value) provides a valid disjunctive cut that is violated by the current solution $x^\star$. Its main disadvantage is its size: the CGLP has $(n+q+1)+K'+K''$ variables and $2(n+q)+3$ constraints, and is hence at least twice as large as the LP used to compute a lower bound. Given that the optimal solution of the CGLP is used, in our implementation, to produce just one disjunctive cut, solving one problem (4.1) for each disjunction of a set of violated ones, at every branch-and-bound node, might prove ineffective. To this purpose, Balas et al. [6, 9] present a method to implicitly solve the CGLP for binary disjunctions by applying pivot operations to the original linear relaxation, only with a different choice of variables. It is worth noting that, unlike the MILP case, here A' and A'' differ for much more than a single column. As shown in [2], this implies that the result by Balas et al. does not hold in this case.

An example. Consider the continuous nonconvex nonlinear program $\mathbf{P}_0 : \min\{x^2 : x^4 \geq 1\}$. It is immediate to check that its feasible region is the nonconvex union of intervals $(-\infty, -1] \cup [+1, +\infty)$, and that its two global minima are -1 and $+1$. Its reformulation is as follows:

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

It is crucial to note here that, although the problem is trivial and can be solved by inspection, state-of-the-art MINLP solvers that use reformulation *ignore* the relationship between the objective function and the constraint, i.e., $y = w^2$. The tightest convex relaxations of the two nonlinear constraints are obtained by simply replacing the equality with inequality, therefore any LP relaxation generated by a MINLP solver is a relaxation of

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

whose optimal solution is $(x, w, y) = (0, 0, 1)$, whose value is $w = 0$ and which is infeasible for $\mathbf{P}$ and hence for $\mathbf{P_0}$. Imposing the disjunction $x \leq 0 \vee x \geq 0$ to $\mathbf{P}$ yields two subproblems with the following convex relaxations:

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

both with 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

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

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 $\mathbf{CR}'$ and $\mathbf{CR}''$ 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 ($\mathbf{SCR}'$) and ($\mathbf{SCR}''$). If added to ($\mathbf{CR}$), a lower bound of 1 is obtained which allows to solve the problem without branching. It is easy to prove that even using a linear relaxation and applying the CGLP procedure yields the same disjunctive cut. This simple example can be complicated by considering n variables subject each to a nonconvex constraint:

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

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 one needs to branch on all disjunctions. However, the disjunctive cuts $w_i \geq 1 \forall i = 1, 2 \dots, n$, where w_i is the variable associated with the expression x_i^2, allow to find an optimal solution immediately. Although this example uses a nonlinear convex hull 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 and a CGLP-based method for generating disjunctive cuts is used.

A test with $n = 100$ was conducted using the two variants of COUENNE DC and V0 described in Section 6, respectively with and without disjunctive cuts. 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 procedure is sketched in Table 1, and its details are discussed below. It requires a description of $\mathbf{P}$, the feasible region of its linear relaxation $\mathbf{LP} = \{x \in \mathbb{R}^{n+q} : Ax \leq a\}$, lower and upper bounds ℓ, u, an optimal solution $x^\star$ of the relaxation, and disjunction $x_i \leq \beta \vee x_i \geq \beta$. Branch-and-bound procedures for MINLP recursively reduce the bounding box $[\ell, u]$, therefore the procedure sketched can be applied at every branch-and-bound node.

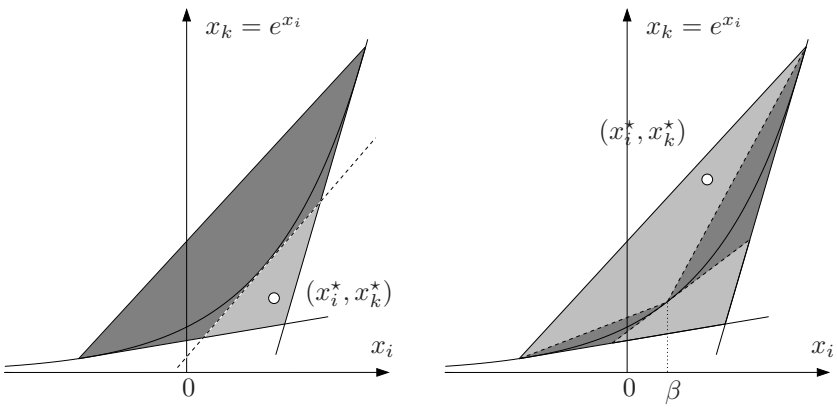
Implementation details. We have used COUENNE [11, 23], an Open Source software package included in the Coin-OR infrastructure [44], for all experiments. It implements reformulation, linearization, and bound reduction methods, as well as a *reliability branching* scheme [12]. It also recognizes complementarity constraints, i.e., constraints of the form $x_i x_j = 0$, with $i \neq j$, as the disjunction $x_i = 0 \vee x_j = 0$.

COUENNE is a BB whose lower bounding method is based on the reformulation and linearization steps as discussed in Section 2. At the beginning, the linearization procedure is run in order to obtain an initial LP relaxation. At each node of the BB, two cut generation procedures are used: up to four rounds of cuts to refine the linear relaxation and at most one round of disjunctive cuts.

Calls to the disjunctive cut generator. While the procedure that generates a linear relaxation of the MINLP is relatively fast and hence is carried out multiple times at every node of the BB tree, separating disjunctive cuts using a CGLP is CPU-intensive as it requires solving a relatively large LP for each disjunction. Therefore, in our experiments, disjunctive cuts are only generated at BB nodes of depth lesser than 10, and at most 20 violated disjunctions per node are used.

For each node where disjunctive cuts are separated, the first 20 disjunctions of a list provided by the branch-and-bound algorithm (which, in COUENNE's case, is CBC [18]) are selected. This list is sorted in non-increasing order of the infeasibility of variable x_i of the reformulation and a solution $x^\star$ of the LP relaxation.

Disjunctions and convex/concave functions. Disjunctions can help cut an LP solution through branching or disjunctive cuts, but they are not the only way to do so. In order to generate disjunctive cuts only when necessary, in our experiments a disjunction from a nonlinear constraint $x_k = \vartheta_k(x)$ is *not* used, for branching or for separating a disjunctive cut, if it is possible to add a linearization inequality for ϑ_k that cuts $x^\star$, i.e., if $x_k^\star > \vartheta_k(x^\star)$ and ϑ_k is concave or $x_k^\star < \vartheta_k(x^\star)$ and ϑ_k is convex. Consider Figure 3, where the components $(x_i^\star, x_k^\star)$ of the LP solution associated



(a) Separable solution

(b) Non-separable solution

FIG. 3. *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 it is much quicker to separate. In (b), no refinement of the linear relaxation is possible, and a disjunction must be applied.*

with the nonlinear constraint $x_k = e^{x_i}$ are shown in two cases: in the first, $x_k^\star < e^{x_i^\star}$, while in the second $x_k^\star > e^{x_i^\star}$. In both cases, a disjunction $x_i \leq \beta \vee x_i \geq \beta$ could be considered by the MINLP solver as a means to create two new subproblems and the two LP relaxations, both excluding $x^\star$, or to create a disjunctive cut. However, the point $x^\star$ in Figure 3(a) can be cut simply by refining the LP relaxation, thus avoiding the evaluation of a disjunction on x_i.

Disjunctions on integer variables. Disjunctions of the form $x_i \leq \lfloor x_i^\star \rfloor \vee x_i \geq \lceil x_i^\star \rceil$ are also used in this framework, as they may also lead to two subproblems with refined relaxation and tightened bounds. A more efficient separation technique for generating disjunctive cuts based on integer disjunctions, CGLLANDP [20], is available in the COIN-OR cut generation library [45]. COUENNE has an option for separating cuts using CGLLANDP, but can also use the procedure described above on integer disjunctions. The two procedures are different: while the first uses a variant of the method proposed by Balas et al. [6, 9] and is hence faster, the CGLP-based procedure in COUENNE takes advantage of the reduced bounds and the refined linearization, which are not available to CGLLANDP. In our experiments, we have turned off CGLLANDP and separated disjunctive cuts on integer variables with the method described in this paper, with the only difference that the disjunction used is of the form $x_i \leq \beta \vee x_i \geq \beta + 1$, with $\beta \in \mathbb{Z}$.

Branching priorities. In the software framework we chose for our implementation, integer and nonlinear disjunctions have a *priority*, i.e., a scalar

Input:	A problem $\mathbf{P}$ and a linear relaxation, $\mathbf{LP}$: (A, a, ℓ, u)
	Optimal solution of $\mathbf{LP}$: $x^\star$
	A disjunction $x_i \leq \beta \vee x_i \geq \beta$
1	Create $\mathbf{P}'$ from $\mathbf{P}$ by imposing $x_i \leq \beta$
2	Create $\mathbf{P}''$ from $\mathbf{P}$ by imposing $x_i \geq \beta$
3	Apply bound reduction to $\mathbf{P}'$, obtain $[\ell', u']$
4	Apply bound reduction to $\mathbf{P}''$, obtain $[\ell'', u'']$
5	Generate linear relaxations $\mathbf{SLP}'$ of $\mathbf{P}'$, defined by A', a'
6	Generate linear relaxations $\mathbf{SLP}''$ of $\mathbf{P}''$, defined by A'', a''
7	Construct CGLP (4.1) and solve it
8	Return (α, α_0)

TABLE 1
Procedure for generating a disjunctive cut for problem $\mathbf{P}$.

that determines precedences in choosing disjunctions: a disjunction can be selected only if there are no violated disjunctions with smaller priority. In our experiments, we have assigned the same priority to all disjunctions, therefore disjunctions are chosen based on their infeasibility only.

Rank of the generated inequalities. Suppose an LP relaxation $\mathbf{LP}$ and K disjunctions are available at a branch-and-bound node. In our implementation, after generating the disjunctive cut $\alpha^\top x \leq \alpha_0$ for the j-th disjunction, the cut for the $(j+1)$-st disjunction is generated using a CGLP constructed from the same LP relaxation $\mathbf{LP}$, i.e., *without* the new cut $\alpha^\top x \leq \alpha_0$ (COUENNE has an option to amend the CGLP with the cuts generated in the same round, which was not tested). This is done to limit the rank of the new cut, given that high-rank cuts may introduce numerical errors. In fact, the K-th generated cut would otherwise be of rank K even if $\mathbf{LP}$ contained no disjunctive cuts, and, in the worst case, its rank would be Kd if generated in a branch-and-bound node at depth d. Therefore, given the maximum node depth of 10 mentioned above, the maximum rank of the cuts separated in the tests below is 10. If, in the future, disjunctive cuts are generated at deeper nodes of the BB tree, mechanisms for controlling the rank of each cut, such as that used by [55], might be necessary.

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

- *MacMINLP* [37] and *minlplib* [16]: a collection of MINLP instances, both convex and nonconvex;
- *nConv:* a collection of nonconvex MINLPs[2];
- *MIQQP:* Mixed Integer quadratically constrained quadratic pro-

[2]See `https://projects.coin-or.org/Couenne/browser/problems/nConv`

grams [47]; model `qpsi.mod` was used;
- *globallib:* a collection of continuous NLP problems [29];
- *boxQP:* continuous, nonconvex, box-constrained quadratic problems; the smaller instances are from [63] and those with more than 60 variables are from [15];
- *airCond*, a 2D bin-packing problem for air conduct design [28].

While most of these instances are nonconvex MINLP problems, the *boxQP* instances belong to a specific class of problems for which much more efficient methods exist. In particular, the one presented by Burer et al. [15] solves all these instances more quickly than COUENNE. It should also be noted that, for this type of problems, there are much stronger relaxations than that obtained through reformulation, see for example Anstreicher [3].

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 [12], Section 5.1;
- RB: no disjunctive cuts, and an extension of reliability branching [1] to MINLP denoted **int-br-rev** in [12];
- 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 and therefore disjunctions, disjunctive cuts are separated only using the most promising disjunctions.

All tests were performed on a computer with a 2.66GHz processor, 64GB of RAM, 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.

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*

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

is given, 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

	<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 report, for those instances that could be solved within the time limit of two hours, 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, among those that could not be solved within two hours by any of the variant, for which a variant obtained the smallest gap, or within 10% of the smallest, in terms of the remaining gap.

variants. If no feasible solution was found by any variant, the lower bound is reported in brackets. The best performances are highlighted in bold.

Table 2 summarizes our results by pointing out what variants perform better overall. For each variant, the 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 before the time limit 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.

Variants with disjunctive cuts seem to outperform the others, especially for the *boxQP* instances (we recall that more efficient solvers are available for this type of problem [15]). For some instances, however, disjunctive cuts only seem to slow down 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 especially for the *boxQP* instances: although a large amount of time is spent in generating disjunctive cuts, this results in a better lower bound or a lower 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 variants using performance profiles [24]. Figure 4(a) depicts a comparison on the CPU time. This performance profile only considers 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 4(b) is a performance profile on the number of nodes.

Figure 4(c) is a comparison on the remaining gap, and reports on all instances for which *none* of the variants could obtain an optimal solution in two hours or less. Note that this is not a performance profile: rather than the *ratio* between gaps, 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 pay 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 as effective in MINLP solvers as they are in MILP. Although they are generated from an LP relaxation of a nonconvex MINLP, they can dramatically improve the lower bound and hence the performance of a branch-and-bound method.

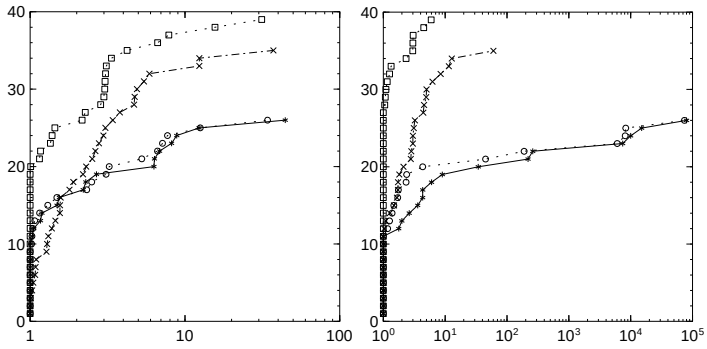
One disadvantage of the CGLP procedure, namely having to solve a large LP in order to obtain a single cut, carries over to the MINLP case. Some algorithms have been developed, for the MILP case, to overcome this issue [6, 9]. Unfortunately, as shown in [2], their extension to the MINLP case is not as straightforward.

Acknowledgments. The author warmly thanks François Margot for all the useful discussions that led to the development of this work, and an anonymous referee whose feedback helped improve this article. Part of this research was conducted while the author was a Postdoctoral Fellow at the Tepper School of Business, Carnegie Mellon University, Pittsburgh PA.

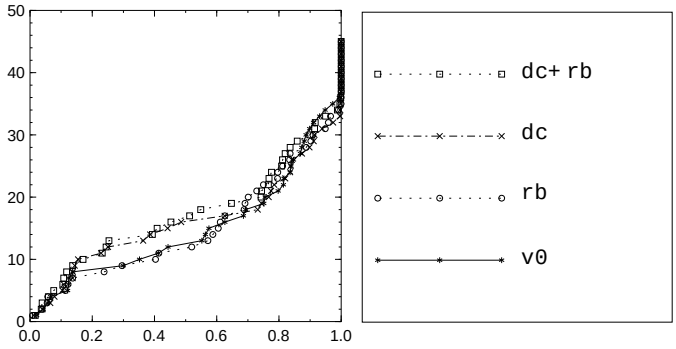
REFERENCES

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

[2] K. Andersen, G. Cornuéjols, and Y. Li, *Split closure and intersection cuts*, Mathematical Programming, 102 (2005), pp. 457–493.



(a) CPU time (b) BB nodes



(c) Remaining gap

FIG. 4. *Performance profiles for the four variants of* COUENNE.

[3] K. M. ANSTREICHER, *Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming*, Journal of Global Optimization, 43 (2009), pp. 471–484.

[4] D. L. APPLEGATE, R. E. BIXBY, V. CHVÁTAL, AND W. J. COOK, *The Traveling Salesman Problem, A Computational Study*, Princeton, 2006.

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

[6] 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.

[7] 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.

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

[9] 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.
[10] E. Balas and A. Saxena, *Optimizing over the split closure*, Mathematical Programming, Series A, 113 (2008), pp. 219–240.
[11] P. Belotti, couenne*: a user's manual*, tech. rep., Lehigh University, 2009.
[12] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter, *Branching and bounds tightening techniques for non-convex MINLP*, Optimization Methods and Software, 24 (2009), pp. 597–634.
[13] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent, *Experiments in mixed–integer linear programming*, Mathematical Programming, 1 (1971), pp. 76–94.
[14] L. T. Biegler, I. E. Grossmann, and A. W. Westerberg, *Systematic Methods of Chemical Process Design*, Prentice Hall, Upper Saddle River (NJ), 1997.
[15] 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.
[16] 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. Available online at `http://www.gamsworld.org/minlp/minlplib/minlpstat.htm`.
[17] A. Caprara and A. N. Letchford, *On the separation of split cuts and related inequalities*, Mathematical Programming, Series B, 94 (2003), pp. 279–294.
[18] CBC-2.1, *Available from* `https://projects.coin-or.org/cbc`.
[19] M. T. Çezik and G. Iyengar, *Cuts for Mixed 0-1 Conic Programming*, Mathematical Programming, Ser. A, 104 (2005), pp. 179–200.
[20] CglLandP, `https://projects.coin-or.org/Cgl/wiki/CglLandP`.
[21] J. S. Cohen, *Computer algebra and symbolic computation: elementary algorithms*, A. K. Peters, Ltd., 2002.
[22] W. Cook, R. Kannan, and A. Schrijver, *Chvátal closures for mixed integer programming problems*, Mathematical Programming, 47 (1990), pp. 155–174.
[23] Couenne, `https://www.coin-or.org/Couenne`.
[24] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming, 91 (2002), pp. 201–213.
[25] S. Drewes, *Mixed Integer Second Order Cone Programming*, PhD thesis, Technische Universität Darmstadt, 2009.
[26] C. A. Floudas, *Global optimization in design and control of chemical process systems*, Journal of Process Control, 10 (2001), pp. 125–134.
[27] A. Frangioni and C. Gentile, *Perspective cuts for a class of convex 0-1 mixed integer programs*, Mathematical Programming, 106 (2006), pp. 225–236.
[28] A. Fügenschuh and L. Schewe, *Solving a nonlinear mixed-integer sheet metal design problem with linear approximations*, work in progress.
[29] GAMS Development Corp., *Gamsworld global optimization library*. `http://www.gamsworld.org/global/globallib/globalstat.htm`.
[30] E. Hansen, *Global Optimization Using Interval Analysis*, Marcel Dekker, Inc., New York, 1992.
[31] C. Helmberg and F. Rendl, *Solving quadratic (0,1)-problems by semidefinite programs and cutting planes*, Mathematical Programming, 82 (1998), pp. 291–315.
[32] R. Horst and H. Tuy, *Global Optimization: Deterministic Approaches*, Springer Verlag, Berlin, 1996.
[33] R. C. Jeroslow, *There cannot be any algorithm for integer programming with quadratic constraints*, Operations Research, 21 (1973), pp. 221–224.
[34] J. J. Júdice, H. D. Sherali, I. M. Ribeiro, and A. M. Faustino, *A complementarity-based partitioning and disjunctive cut algorithm for mathematical programming problems with equilibrium constraints*, Journal of Global

Optimization, 36 (2006), pp. 89–114.
[35] 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.
[36] M. Karamanov and G. Cornuéjols, *Branching on general disjunctions*, Mathematical Programming, (2007).
[37] S. Leyffer, `MacMINLP`*: ampl collection of MINLPs*. Available at `http://www-unix.mcs.anl.gov/~leyffer/MacMINLP`.
[38] L. Liberti, *Writing global optimization software*, in Global Optimization: from Theory to Implementation, L. Liberti and N. Maculan, eds., Springer, Berlin, 2006, pp. 211–262.
[39] 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.
[40] 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.
[41] L. Liberti and C. C. Pantelides, *Convex envelopes of monomials of odd degree*, Journal of Global Optimization, 25 (2003), pp. 157–168.
[42] Lindo Systems, *Lindo solver suite*. Available online at `http://www.gams.com/solvers/lindoglobal.pdf`.
[43] 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.
[44] R. Lougee-Heimer, *The Common Optimization INterface for Operations Research*, IBM Journal of Research and Development, 47 (2004), pp. 57–66.
[45] R. Lougee-Heimer, *Cut generation library*. `http://projects.coin-or.org/Cgl`, 2006.
[46] G. P. McCormick, *Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems*, Mathematical Programming, 10 (1976), pp. 146–175.
[47] H. Mittelmann, *A collection of mixed integer quadratically constrained quadratic programs*. `http://plato.asu.edu/ftp/ampl_files/miqp_ampl`.
[48] R. E. Moore, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.
[49] J. H. Owen and S. Mehrotra, *A disjunctive cutting plane procedure for general mixed-integer linear programs*, Mathematical Programming, 89 (2001), pp. 437–448.
[50] A. T. Phillips and J. B. Rosen, *A quadratic assignment formulation of the molecular conformation problem*, tech. rep., CSD, Univ. of Minnesota, 1998.
[51] I. Quesada and I. E. Grossmann, *Global optimization of bilinear process networks and multicomponent flows*, Computers & Chemical Engineering, 19 (1995), pp. 1219–1242.
[52] 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.
[53] F. Rendl and R. Sotirov, *Bounds for the quadratic assignment problem using the bundle method*, Mathematical Programming, 109 (2007), pp. 505–524.
[54] N. V. Sahinidis, *BARON: A general purpose global optimization software package*, Journal of Global Optimization, 8 (1996), pp. 201–205.
[55] A. Saxena, P. Bonami, and J. Lee, *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.
[56] ———, *Convex relaxations of non-convex mixed integer quadratically constrained*

programs: Projected formulations, Mathematical Programming, (2011). To appear.

[57] H. Scheel and S. Scholtes, *Mathematical programs with complementarity constraints: Stationarity, optimality, and sensitivity*, Mathematics of Operations Research, 25 (2000), pp. 1–22.

[58] 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.

[59] 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.

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

[61] 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.

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

[63] D. Vandenbussche and G. L. Nemhauser, *A branch-and-cut algorithm for nonconvex quadratic programs with box constraints*, Mathematical Programming, 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. If the remaining gap cannot be computed due to the lack of a feasible solution, the lower bound, in brackets, is shown instead.

	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").