RC24620 (W0808-031) August 13, 2008
Mathematics

IBM Research Report

Branching and Bounds Tightening Techniques for Non-Convex MINLP

Pietro Belotti[1], Jon Lee[2], Leo Liberti[3], François Margot[1], Andreas Waechter[2]

[1]Tepper School of Business
Carnegie Mellon University
Pittsburgh, PA
USA

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

[3]LIX
École Polytechnique
91128 Palaiseau
France

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

Branching and bounds tightening techniques for non-convex MINLP

Pietro Belotti[⋆,†], Jon Lee[‡], Leo Liberti[◇], François Margot[†], and Andreas Wächter[‡]

Many industrial problems can be naturally formulated using Mixed Integer Nonlinear Programming (MINLP). Motivated by the demand for Open-Source solvers for real-world MINLP problems, we have developed a spatial Branch-and-Bound software package named COUENNE (Convex Over- and Under-ENvelopes for Nonlinear Estimation). In this paper, we present the structure of COUENNE and discuss in detail our work on two of its components: bounds tightening and branching strategies. We then present experimental results on a set of MINLP instances including some industrial applications. We also compare the performance of COUENNE with a state-of-the-art solver for nonconvex MINLPs.

1. Nonconvex MINLP: Introduction and applications

Consider the following Mixed Integer Nonlinear Programming (MINLP) problem:

$$\begin{array}{lll} (\mathbf{P}') \ \min & f(x) & \\ s.t. & g_j(x) \leq 0 & \forall j \in M \\ & x_i^l \leq x_i \leq x_i^u & \forall i \in N_0 \\ & x_i \in \mathbb{Z} & \forall i \in N_0^I \subseteq N_0, \end{array}$$

where f and the g_j are possibly nonconvex functions, $n = |N_0|$ is the number of variables, and $x = (x_i)_{i \in N_0}$ is the n-vector of variables. We assume that f and all g_j's are *factorable*, i.e., they are expressed as $\sum_i \prod_k h_{ik}(x)$, where all functions $h_{ik}(x)$ are univariate — notice that binary operators such as division or power are also included in this framework (this definition extends that given in [53] by considering an arbitrary number of factors in the product).

This general type of problem finds applications in a number of fields: Chemical Engineering [16, 31, 41], Portfolio Optimization [23], and Computational Biology [48, 49, 65] are just a few examples. Both the integrality of a subset of variables and the possible nonconvexity of f and the g_j create difficulties when solving this type of problem.

When f is linear and all of the g_j are affine, we have a Mixed Integer Linear Programming (MILP) problem, a class of difficult problems for which practical Branch-and-Cut methods have been successfully developed [40, 63, 85]. These methods have three essential ingredients: a bounding scheme returning a lower

⋆Corresponding author (email: belotti@cmu.edu).
†Tepper School of Business, Carnegie Mellon University, Pittsburgh PA.
‡IBM T.J. Watson Research Center, Yorktown Heights NY.
◇LIX, École Polytechnique, 91128 Palaiseau, France.

bound on the optimal value of the problem, a branching scheme partitioning a problem into two or more smaller problems, and a cutting scheme adding valid inequalities to the problem. The key ingredient in the process is the computation of an optimal solution $\bar{x}$ of the linear relaxation of the problem (i.e., the relaxation obtained by ignoring the integrality requirement on the variables). The bounding scheme is then simply the computation of the objective value of the solution $\bar{x}$ of the relaxation. A simple branching scheme is to select an integer variable x_i whose value $\bar{x}_i$ is not integer and to create two problems — one obtained by appending the inequality $x_i \leq \lfloor \bar{x}_i \rfloor$ to the problem and the other obtained by appending the inequality $x_i \geq \lfloor \bar{x}_i \rfloor + 1$ to the problem. Finally, the cutting scheme uses various algebraic techniques for generating inequalities satisfied by all feasible solutions, but violated by $\bar{x}$.

The adaptation of these techniques to solve MINLP is a challenging research area. The present work squarely lies in this area, as we pursue the implementation of a general-purpose algorithm for solving MINLP based on existing software tools for MILP. The numerous industrial applications that are inherently nonlinear and nonconvex provide a strong incentive for developing efficient MINLP solvers.

When the functions f and the g_j are nonlinear but convex, a Branch-and-Cut approach mimicking closely the steps of the MILP case can be used [17, 28, 66]. However, when some of these functions are nonconvex, many additional difficulties have to be overcome. As an example, computing a lower bound on the optimal value of the problem becomes much harder. Indeed, relaxing the integrality of the variables yields a nonconvex Nonlinear Program (NLP) with potentially many local minima. A valid lower bound can be obtained by finding a global minimum of the relaxed problem, a task not even aimed at by most current NLP solvers, or via convex relaxations of $\mathbf{P}'$. As a result, the simple branching scheme described above is not sufficient, as branching might be necessary even if the problem has no integer variables.

The method of choice for solving nonconvex MINLPs is Branch-and-Bound (BB), an implicit enumeration of the local minima based on a *divide and conquer* approach. This method sequentially solves *subproblems* of $\mathbf{P}'$ that are obtained by partitioning the original solution space. The initial problem is subdivided into two or more subproblems and each is solved separately, if needed by recursively partitioning it into smaller subproblems. For each subproblem, one computes lower and upper bounds for the objective function value, bounds that are used to discard from the search those subproblems guaranteed to contain no global optimum. In the MINLP literature, BB algorithms are also called *spatial Branch-and-Bound* (sBB), and we use this term throughout the paper.

A vast body of literature covers solution techniques for problem $\mathbf{P}'$. The reader may refer to [39, 78, 79], the surveys [36, 45], and references therein for a general introduction on MINLP solvers. One of the first papers on continuous global optimization by Branch-and-Bound dates from 1969 [27]. In the 1970s and 1980s, work on continuous or mixed integer global optimization was scarce. Most of the papers published in this period dealt either with applications of global optimization to very specific cases, or with theoretical results concerning convergence proofs. One notable exception was the work of McCormick [53] who considered symbolic transformations of problems to formulate a convex relaxation of nonconvex NLPs. Branch-and-Bound techniques were proposed in order to solve particular classes of problems (e.g. concave minimization problems [38]). The first method that was able to deal with problems in the form $\mathbf{P}'$ was the Branch-and-Reduce algorithm [68, 69]. Shortly afterwards, Floudas' team published their first article on the αBB Branch-and-Bound method [10] which was then thoroughly explored and anal-

ysed in several subsequent papers [3, 4, 5, 6, 7, 8]. One notable limitation of the αBB algorithm is that it relies on the functions being twice differentiable in the continuous variables. Since the introduction of the αBB algorithm, a number of Branch-and-Select algorithms geared towards the most generic nonconvex MINLP formulation appeared in the literature, e.g. the Symbolic Reformulation sBB approach [75, 76, 77], the Reduced Space Branch-and-Bound approach [26] (which only applies to continuous NLPs), the Branch-and-Contract algorithm [86] (which also only applies to continuous NLPs) and Barton's Branch-and-Cut framework [42]. Several Interval Analysis based global optimization methods [61, 80, 87] also belong to the sBB class of algorithms.

Several implementations are available for both the general problem $\mathbf{P}'$ and some of its special cases. Among the exact approaches to solve $\mathbf{P}'$ we mention BARON [71] and LINDOGLOBAL [52], which take advantage of a factorable formulation of $\mathbf{P}'$ to derive valid bounds and exact solutions, while αBB [6] and LAGO [60] use linear and quadratic approximations of f and the g_j instead. A large amount of work has also been carried out for the special case of *convex* MINLPs, where f and the g_j are all convex [83] or *quasi-convex* [58]. Well-known codes for solving convex MINLPs are: SBB [35], MINLP_BB [28, 29, 44], BONMIN [17] and FILMINT [1]. Notice that solvers for convex MINLPs can be used on nonconvex problems as heuristics, as they may provide a feasible solution. In particular, the Branch-and-Bound algorithmic option of BONMIN has some parameter options that can be set appropriately so as to encourage a search for good feasible solutions.

In this paper, we describe an Open-Source software package called COUENNE (Convex Over- and Under-ENvelopes for Nonlinear Estimation), implementing an sBB for problems of the form $\mathbf{P}'$. COUENNE is built within the COIN-OR framework [22]. Following the example of many other Open-Source codes, instead of focusing on the effectiveness of the algorithm, we have chosen to create a flexible structure, that any user with working knowledge of the C++ language can easily change and specialize to a particular class of MINLP problems, or improve by adding new techniques to the general structure.

The next section describes the basic spatial Branch-and-Bound algorithm and introduces the specific building blocks (linearization, bounds tightening, branching, and heuristics) corresponding to subsequent sections of this paper. In Section 7, we present experimental tests conducted on a set of publicly available MINLP instances. Section 8 closes the paper with a few comments.

2. Spatial Branch-and-Bound

An sBB method creates a hierarchy of nodes represented by a binary tree, known as the *sBB tree*. At the root of the tree is the initial problem $\mathbf{P}'$; partitioning $\mathbf{P}'$ is equivalent to creating two *descendent* nodes 1 and 2, with corresponding problems $\mathbf{P}_1$ and $\mathbf{P}_2$, which can in turn be partitioned. In the following, k denotes a node of the sBB tree containing a restriction $\mathbf{P}_k$ of $\mathbf{P}'$, while $\mathbf{CP}_k$ is the continuous (nonlinear) relaxation of $\mathbf{P}_k$ and $\hat{x}^k$ a local minimum of $\mathbf{CP}_k$. A valid lower bound on $\mathbf{P}_k$ can be obtained through its *convex* relaxation. COUENNE creates a linear relaxation of $\mathbf{P}_k$, which we denote $\mathbf{LP}_k$, and $\bar{x}^k$ is an optimal solution of $\mathbf{LP}_k$.

A schematic description of the sBB algorithm is provided in Table 1; the right column gives a pointer to the section describing in detail the corresponding step as implemented in COUENNE. The essential components of any sBB algorithm are as follows:

- a method to compute a *lower bound* on $\mathbf{P}_k$;

Input:	Problem $\mathbf{P}'$	
Output:	An optimal solution of $\mathbf{P}'$	
	Define set L of subproblems	
	$L \leftarrow \{\mathbf{P}'\}$; $z^u \leftarrow +\infty$	
	while $L \neq \emptyset$	
	choose $\mathbf{P}_k \in L$	
	$L \leftarrow L \setminus \{\mathbf{P}_k\}$	
	apply bounds tightening to $\mathbf{P}_k$	(Sec. 4)
	if $\mathbf{P}_k$ is feasible, **then**	
	generate a linear relaxation $\mathbf{LP}_k$ of $\mathbf{P}_k$	(Sec. 3)
	repeat	
	solve $\mathbf{LP}_k$; let $\bar{z}^k$ be the optimal objective value	
	refine linearization $\mathbf{LP}_k$	
	until $\bar{z}^k$ does not improve sufficiently	
	(optional) find a local optimum $\hat{z}^k$ of $\mathbf{CP}_k$	(Sec. 6)
	$z^u \leftarrow \min\{z^u, \hat{z}^k\}$	
	if $\bar{z}^k \leq z^u - \epsilon$ **then**	
	choose a variable x_i	(Sec. 5.1)
	choose a branching point x_i^b	(Sec. 5.5)
	create subproblems:	
	$\mathbf{P}_{k-}$ with $x_i \leq x_i^b$,	
	$\mathbf{P}_{k+}$ with $x_i \geq x_i^b$	
	$L \leftarrow L \cup \{\mathbf{P}_{k-}, \mathbf{P}_{k+}\}$	
	output z^u	

Table 1. An sBB algorithm for the MINLP $\mathbf{P}'$.

- a method to compute an *upper bound* or a *feasible solution* for $\mathbf{P}_k$;
- a *branching technique* to partition $\mathbf{P}_k$;
- a *bounds tightening* procedure to reduce the feasible space of $\mathbf{P}_k$ — and hence the running time of the algorithm — without eliminating any optimal solution.

The bounds tightening component is not essential for the convergence of the sBB, but an efficient implementation of this component can dramatically improve performances.

This paper focuses on two parts of the sBB algorithm: *branching* and *bounds tightening*. We discuss lower bounding by linearization in Section 3. In Section 4, we describe an implementation of bounds tightening, used before generating the linear relaxation $\mathbf{LP}_k$ at each sBB node and in other parts of the sBB. In Section 5, we extend the idea of *reliability branching* [2] to continuous variables in an MINLP setting, and compare it with standard spatial branching techniques as well as one introduced by Tawarmalani and Sahinidis [79].

3. Linearization of nonconvex MINLPs

The linear relaxation $\mathbf{LP}_0$ of the root node problem $\mathbf{P}_0$ is constructed in two steps: *reformulation* and *linearization*. The former translates the problem to an equivalent one that is easier to deal with from a symbolic viewpoint; the latter generates the actual linear relaxation. Reformulation is only required at the root node $\mathbf{P}_0$, while linearization is also performed at all other nodes $\mathbf{P}_k$. More specifically, the linearization $\mathbf{LP}_k$ of node $\mathbf{P}_k$ is a refinement of the one of the ancestor of $\mathbf{P}_k$.

Although this technique is well known in the Global Optimization community, we describe it in detail to introduce notation and concepts that are useful to discuss

other features of COUENNE.

3.1. *Reformulation*

A factorable function $h(x)$ is either a simple univariate function $h(x_i) = x_i$ for some i, or it can be obtained using one of the following operations on other factorable functions $h_j(x)$:

- a linear combination, $h(x) = a_0 + \sum_{j=1}^{k} a_j h_j(x)$;
- a product, $h(x) = \prod_{j=1}^{k} h_j(x)$;
- a quotient, $h(x) = h_1(x)/h_2(x)$;
- a power, $h(x) = h_1(x)^{h_2(x)}$; or
- a composition of univariate functions, $h(x) = h_1(h_2(x))$ (e.g. sine, logarithm, exponential, absolute value...).

The reformulation corresponding to $h(x)$ is built recursively as follows: if $h(x)$ is obtained using functions $h_j(x)$ for $j = 1, \ldots, k$, a new *auxiliary* variable w_{n+j} is introduced as well as the *defining* constraint $w_{n+j} = h_j(x)$ for $j = 1, \ldots, k$.

Let us define the set of operators as

$$\Theta = \{\text{sum}, \text{product}, \text{quotient}, \text{power}, \exp, \log, \sin, \cos, \text{abs}\}.$$

The reformulation is applied recursively to $h_j(x)$ until all auxiliary variables are associated with operators of Θ, and each can thus be defined as $w_{n+j} = \vartheta_j(x, w_{n+1}, w_{n+2} \ldots, w_{n+j-1})$ with $\vartheta_j \in \Theta$.

The power function $\vartheta(x_i, x_j) = x_j^{x_i}$ is transformed into the univariate term $\vartheta'(x_k) = e^{x_k}$, where a new auxiliary variable x_k is associated with $x_i \log x_j$. This transformation gives an equivalent problem assuming that $x_j > 0$ was implied by the other constraints. Otherwise, all solutions x such that $x_j = 0$ are excluded.

Notice that this reformulation lifts the original problem to a larger space by adding a set of auxiliary variables indexed by $Q = \{n+1, n+2, \ldots, n+q\}$. It is an *opt-reformulation* [46, 47], i.e. a reformulation where all local and global optima of the original problem are mapped into local and global optima of the reformulation. This allows us to use one linearization technique for each operator of Θ. The reformulation is, in general, as follows:

$$\begin{array}{lll}
(\mathbf{P''}) \quad \min & w_{n+q} & \\
\quad s.t. & w_i = \vartheta_i(x, w_{n+1}, w_{n+2} \ldots, w_{i-1}) & i \in Q \\
& w_i^l \leq w_i \leq w_i^u & i \in Q \\
& x_i^l \leq x_i \leq x_i^u & i \in N_0 \\
& x_i \in \mathbb{Z} & i \in N_0^I \subseteq N_0 \\
& w_i \in \mathbb{Z} & i \in Q_I \subseteq Q,
\end{array}$$

where $\vartheta_i \in \Theta$ for all $i \in Q$; we assume that the objective function is replaced by the last generated auxiliary variable x_{n+q}. We remark that the bounding box $[w^l, w^u]$ and the integrality of auxiliaries w can be inferred from the bounds and integrality of the original variables x and from the constraints of $\mathbf{P'}$. From now on we drop the notation $w_i = \vartheta_i(x, w_{n+1}, w_{n+2} \ldots, w_{j-1})$ and use instead the more compact $w_i = \vartheta_i(x, w)$, keeping in mind that $\vartheta_i(x, w)$ only depends on x and $w_{n+1}, \ldots, w_{i-1}$.

3.2. *Structure of the reformulation*

Reformulating $\mathbf{P}'$ produces an equivalent problem $\mathbf{P}''$ whose underlying structure — effectively represented as a directed acyclic graph (DAG) — can be used to improve the solution approach. Consider the directed graph $G = (V, A)$ with $V = \{1, 2, \ldots, n+q\}$ and an arc (i, j) if there is $i \in V$ such that $\vartheta_i(x, w)$ explicitly depends on the j-th variable (which can be either original or auxiliary). By definition, original variables have no outgoing arcs, as there is no ϑ_i corresponding to original variables. Different occurrences of the same expression can either be assigned to different auxiliary variables, one per occurrence, or to the same variable. In both cases we obtain an opt-reformulation of the original problem. The latter case, which is implemented in COUENNE, allows for a DAG with larger and fewer connected components, which in turn yields improved results on bounds tightening procedures.

Once the reformulation is carried out, the distinction between original and auxiliary variables can be ignored for most purposes, and we simply obtain the following MINLP with more variables and nonlinear constraints exhibiting a simpler structure than the original one:

$$\begin{array}{lll} (\mathbf{P}) \ \min & x_{n+q} & \\ s.t. & x_i = \vartheta_i(x) & i \in Q \subseteq N \\ & x_i^l \leq x_i \leq x_i^u & i \in N \\ & x_i \in \mathbb{Z} & i \in N_I \subseteq N, \end{array}$$

where N comprises both original and auxiliary variables and N_I is the set of indices of integer variables, including auxiliary variables constrained to be integer. Problem $\mathbf{P}$ is equivalent to the Smith's MINLP standard form [77]. In what follows, we assume that $\mathbf{P}$ is the problem at hand, and we make no distinction between original and auxiliary variables. As $\mathbf{P}$ and $\mathbf{P}'$ are equivalent, there is no need to re-define the problems $\mathbf{P}_k$ considered at each node of the sBB tree: these are simply restrictions of the root node problem $\mathbf{P}$ obtained through branching rules.

3.3. *Linearization*

Consider the variable $x_j = \vartheta_j(x)$ created in the reformulation phase, with $\vartheta_j \in \Theta$ and the bounds $B = [x^l, x^u]$ on x. An inequality $ax \geq b$ is a *linearization inequality* for $x_j = \vartheta_j(x)$ if it is satisfied by all points of the set $\{x \in B : x_j = \vartheta_j(x)\}$. A *linearization* for $x_j = \vartheta_j(x)$ in B, or a ϑ_j*-linearization*, is a system of linear inequalities $A^j x \geq b^j$ such that $X_{\mathbf{LP}} := \{x \in B : A^j x \geq b^j\} \supseteq \{x \in B : x_j = \vartheta_j(x)\}$. In the univariate case $x_j = \vartheta_j(x_i)$, with $x_i^l \leq x_i \leq x_i^u$, a ϑ_j-linearization is a set $\{(x_i, x_j) : a_h x_i + b_h x_j \geq c_h, h = 1, \ldots, H\}$, such that all inequalities are satisfied by all points of the set $\{(x_i, x_j) : x_j = \vartheta_j(x_i), x_i^l \leq x_i \leq x_i^u\}$.

If a ϑ_j-linearization is created for each variable $x_j = \vartheta_j(x), j \in Q$, the overall linear problem $\mathbf{LP}_0$, defined as $\min\{x_{n+q} : Ax \geq b\}$, is a linear relaxation of $\mathbf{P}_0$, hence all feasible solutions of $\mathbf{P}_0$ are also feasible for $\mathbf{LP}_0$ and an optimal solution $\bar{x}$ of $\mathbf{LP}_0$ provides a valid lower bound $\bar{x}_{n+q}$ for the optimal value of $\mathbf{P}_0$.

Analogously to reformulation, several degrees of freedom can be used to improve the performance of the lower bounding procedure, for instance by providing a more compact linear relaxation.

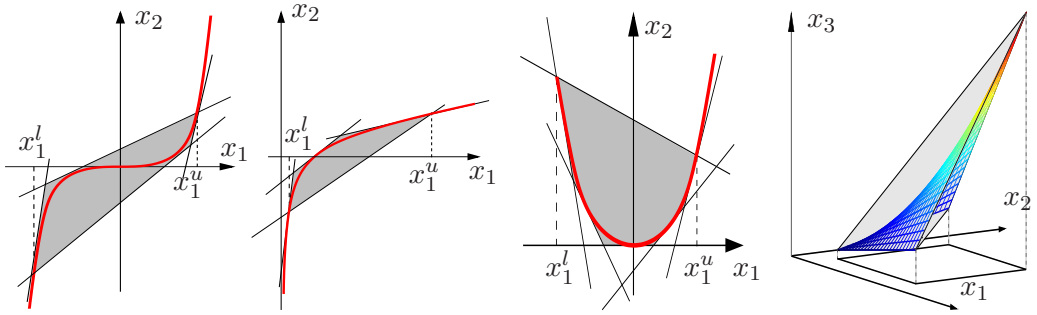
In order to obtain a good lower bound, COUENNE seeks a tight linearization while keeping low the number of inequalities in the linearization. Consider a variable defined as a univariate continuously differentiable function, $x_j = \vartheta_j(x_i)$ with $x_i \in [x_i^l, x_i^u]$. If ϑ_j is convex, then for any $\tilde{x}_i \in [x_i^l, x_i^u]$ the inequality $x_j \geq$

$\vartheta_j(\tilde{x}_i) + \frac{\partial \vartheta_j}{\partial x_i}(\tilde{x}_i)(x_i - \tilde{x}_i)$ is valid for the ϑ_j-linearization. Initially, COUENNE adds a linearization inequality for each $\tilde{x}_i$ in a discretization of the interval $[x_i^l, x_i^u]$, so that a reasonable approximation is obtained. The number of points in the discretization is a parameter that can be set prior to the execution. On the other hand, the tightest upper bound for ϑ_j is given by the inequality $x_j \leq \vartheta_j(x_i^l) + \frac{\vartheta_j(x_i^u) - \vartheta_j(x_i^l)}{x_i^u - x_i^l}(x_i - x_i^l)$. It is satisfied at equality only at the bounds of x_i. For concave functions, similar considerations apply. For functions that are neither convex nor concave (odd powers and trigonometric functions), an effort is made to use, in a separate fashion, the subintervals of $[x_i^l, x_i^u]$ where ϑ_j is convex or concave. For odd powers, we apply the linearization procedure described by Liberti and Pantelides [50], which uses a Newton method to find the minimum of a univariate function. We use a similar technique for sines and cosines.

The two remaining operators are product and quotient. These are treated in a unified manner with a linearization for the set $\{(x_1, x_2, x_3) : x_3 = x_1 x_2, x_i^l \leq x_i \leq x_i^u, i = 1, 2, 3\}$[1]. If x_1 and x_2 are bounded and x_3 is not, the well-known linearization inequalities by McCormick [53] are known to be the tightest (Al-Khayyal et al. [9]):

$$\begin{aligned} x_3 &\geq x_2^l x_1 + x_1^l x_2 - x_2^l x_1^l & \qquad x_3 &\leq x_2^l x_1 + x_1^u x_2 - x_2^l x_1^u \\ x_3 &\geq x_2^u x_1 + x_1^u x_2 - x_2^u x_1^u & \qquad x_3 &\leq x_2^u x_1 + x_1^l x_2 - x_2^u x_1^l. \end{aligned}$$

When the bounds on x_3 are stricter than those inferred by x_1 and x_2, other considerations apply, which we do not include here for the sake of conciseness. Figure 1 provides a geometric interpretation of how some operators are linearized.



(a) $x_2 = x_1^3$ (b) $x_2 = \log x_1$ (c) $x_2 = x_1^2$ (d) $x_3 = x_1 x_2$

Figure 1. Linear approximation of nonlinear functions.

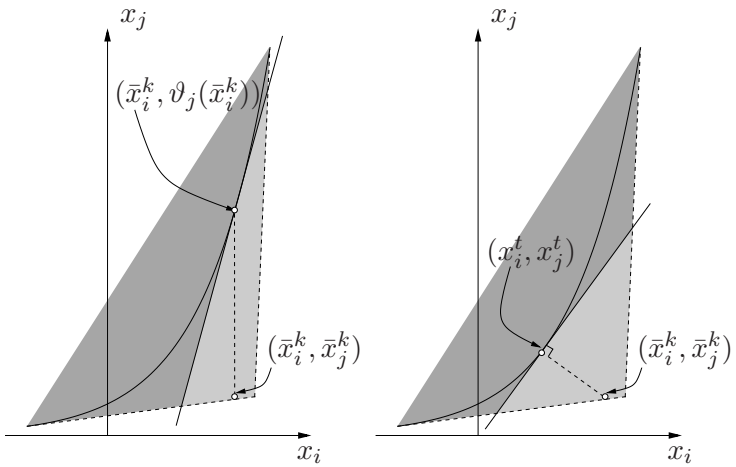
The initial linearization $\mathbf{LP}_0$ is created at the root node, and can be improved with a *refinement* procedure both at the root node and at any node k of the sBB tree. Specifically, if the solution $\bar{x}^k$ of $\mathbf{LP}_k$ is infeasible for $\mathbf{P}_k$, one may improve the lower bound $\bar{x}_{n+q}^k$ by branching (this will be discussed in Section 5) or by *refining* $\mathbf{LP}_k$, i.e. by amending linearization inequalities, yielding an *incremental* linearization. Such a tighter relaxation $\mathbf{LP}_k'$ of $\mathbf{P}_k$, which does not contain $\bar{x}^k$, is a convex set. Hence one faces a *separation* problem: a linear inequality is sought that separates $\bar{x}^k$ from $\mathbf{LP}_k'$. If $\bar{x}^k$ is feasible for any linearization of $\mathbf{P}_k$, and thus the separation problem is infeasible, branching is necessary. Otherwise, the inequality is appended and the new relaxation $\mathbf{LP}_k'$ can be solved. This refinement step can be repeated for a given number of iterations, until a feasible solution for $\mathbf{P}_k$ is

[1]The sets $\{(x_1, x_2, x_3) : x_3 = x_1 x_2, x_i^l \leq x_i \leq x_i^u, i = 1, 2, 3\}$ and $\{(x_1, x_2, x_3) : x_2 = x_3/x_1, x_i^l \leq x_i \leq x_i^u, i = 1, 2, 3\}$ are equivalent assuming $x_1^u < 0$ or $x_1^l > 0$. If $x_1^l < 0 < x_1^u$, no linearization inequalities are appended for $x_2 = x_3/x_1$, and branching using branching rules $x_1 \leq 0$ and $x_1 \geq 0$ may be necessary.

found, or until branching becomes necessary. Although refining the linearization is seldom sufficient, it can improve the lower bound and thus avoid creating too many sBB nodes by branching.

In COUENNE, the relaxation is improved by solving the separation problem for each auxiliary variable $x_j = \vartheta_j(x)$. Refining a ϑ_j-linearization consists of finding a linearization inequality $ax \geq b$ that is violated by $\bar{x}^k$. Consider the variable $x_j = \vartheta_j(x_i)$ with ϑ_j convex univariate. If $\bar{x}^k_j > \vartheta_j(\bar{x}^k_i)$, no inequality can be appended as $(\bar{x}^k_i, \bar{x}^k_j)$ is contained in every convex relaxation of ϑ_j. If $\bar{x}^k_j < \vartheta_j(\bar{x}^k_i)$, then one linearization inequality violated by $\bar{x}^k$ is tangent to ϑ_j in $(\bar{x}^k_i, \vartheta_j(\bar{x}^k_i))$, and is given by $x_j \geq \vartheta_j(\bar{x}^k_i) + \frac{\partial \vartheta_j}{\partial x_i}(\bar{x}^k_i)(x_i - \bar{x}^k_i)$ as in Figure 2a, where the darker shade shows the refined ϑ_j-linearization and the lighter one shows the area cut by the new linearization inequality. However, the *deepest cut*, i.e., a cut with maximum violation with respect to $(\bar{x}^k_i, \bar{x}^k_j)$, corresponds to the line tangent to ϑ_j at the point (x^t_i, x^t_j) of the curve closest to $(\bar{x}^k_i, \bar{x}^k_j)$; see Figure 2b. COUENNE implements the latter approach for all univariate functions, computing (x^t_i, x^t_j) by solving a one-dimensional, convex optimization problem by means of a Newton method.

Notice that, although $\bar{x}^k$ is a vertex of $\mathbf{LP}_k$, $(\bar{x}^k_i, \bar{x}^k_j)$ needs *not* be a vertex of the ϑ_j-linearization (the shaded area in Figure 2); indeed, the ϑ_j-linearization is only a relaxation of $\mathbf{LP}_k$ defined by inequalities in x_i and x_j, and, in general, it contains the projection of $\mathbf{LP}_k$ onto the (x_i, x_j) space.



(a) Current LP point (b) Closest point on the curve

Figure 2. Refining the linear approximation of the convex function $x_j = e^{x_i}$.

In general, the quality of the linearization of $x_j = \vartheta_j(x)$ improves when the bounding box of x is reduced. This suggests that an sBB approach is well suited to solve MINLP problems: partitioning a problem into two subproblems is likely to improve the linearization, and therefore the lower bound, of both of them.

4. Bounds tightening

Bounds tightening (BT) for a variable x_i is the process of reducing the interval $[x^l_i, x^u_i]$ while guaranteeing that the optimal value of the problem is not changed. As a result, one gets a reduction of the solution set and an improved linearization. BT is performed by propagating the constraints' effect to the variable bounds in various ways [37, 57, 67]. As BT procedures are of varying complexity, the most

time consuming ones are not performed at all sBB nodes but only at the root node or up to a limited depth. One such procedure is the Optimality-Based Bounds Tightening (OBBT). It formalizes and solves auxiliary optimization problems in order to tighten the variable bounds as much as possible [45, 66, 75].

Fast bounds tightening procedures can instead be performed at all sBB nodes. The usual procedures are similar to interval propagation in Constraint Programming (CP) and are known in the Global Optimization literature as Feasibility-Based Bounds Tightening (FBBT) [21, 45, 54, 74, 75, 81]. COUENNE implements both OBBT and FBBT, as well as *reduced cost* bounds tightening, a well-known LP based technique [68, 69].

4.1. *Optimality-based bounds tightening*

OBBT uses the existing linearization $\mathbf{LP}_k$ to improve the bounding box [74]. Consider the bounding box $B_k = [x^{l,k}, x^{u,k}]$ of $\mathbf{P}_k$ and the solution set of the linearization, $X_{\mathbf{LP}_k} := \{x \in B_k : Ax \geq b\}$. The following are valid bounds for any variable x_i:

$$\min\{x_i : x \in X_{\mathbf{LP}_k}\} \leq x_i \leq \max\{x_i : x \in X_{\mathbf{LP}_k}\}.$$

While this procedure is useless in MILP, in MINLP the bounds tightened by OBBT provide extra information that is in turn used to improve the linearization. OBBT requires solving at most $2(n+q)$ LP problems. Normally, it is applied only at the root node; in COUENNE, however, it may also be applied at other nodes of the sBB tree up to a limited depth defined by the value of the parameter L_{obbt}, typically less than 10. For nodes at depth $\lambda > L_{\text{obbt}}$, COUENNE applies OBBT with probability $2^{L_{\text{obbt}}-\lambda}$.

4.2. *Feasibility-based bounds tightening*

The FBBT procedure takes as input the reformulation $\mathbf{P}_k$ at node k and a bounding box $B_k = [x^{l,k}, x^{u,k}]$, and returns tighter bounds (including, possibly, a better lower bound for $x^{l,k}_{n+q}$, i.e. on the objective function) or proves that $\mathbf{P}_k$ is infeasible, or that its lower bound is above the best feasible solution. In the remainder of the paper, we drop the notation $B_k = [x^{l,k}, x^{u,k}]$ and $x^{l,k}_{n+q}$ in favor of the more compact $B = [x^l, x^u]$ and x^l_{n+q} when it is clear from the context.

FBBT allows the creation or refinement of a tighter linearization of a MINLP. Its low computational cost makes it a useful procedure to be applied at every sBB node. The trade-off is that FBBT provides in general weaker bounds than the ones obtained from OBBT. Another main difference is that FBBT relies on $\mathbf{P}_k$ rather than on $\mathbf{LP}_k$ to tighten the bounds. We also use FBBT in other settings than described in this section (see Sections 4.3 and 6).

Consider the reformulation $\mathbf{P}$ and the directed acyclic graph $G = (V, A)$ associated with it (see Section 3.2). Graph G provides a straightforward way to connect the change in the bounds of a variable of $\mathbf{P}$ with the rest of the problem. For instance, if the bounds of a variable x_i ($i \in N$) change, the bounds on all variables depending on x_i may also change. This *upward* propagation of the bounds is easily done by checking all nodes $\{i \in V : (i, j) \in A\}$. The tightened bounds can, in turn, trigger other bound changes that are checked by repeating this procedure. This allows the tightening of bounds of all variables using those of the original variables.

Consider a variable defined as $x_i = \vartheta_i(x)$ with $x \in B$. The bounds on x are used to derive a tighter bound on x_i by applying techniques of interval arithmetics on the operator $\vartheta_i(\cdot)$:

$$\min_{x \in B} \vartheta_i(x) \leq x_i \leq \max_{x \in B} \vartheta_i(x).$$

This can be done in the reverse direction, giving a *downward* propagation whereby a change in the bounds of a variable x_i triggers a change in the bounds of all variables on which x_i depends, i.e., all nodes in $\{j \in V : (i,j) \in A\}$. In fact, if the bounds on x_i do not depend only on those of x and $\vartheta_i(\cdot)$ but are also given explicitly as branching rules, constraints, or an upper bound $\hat{x}_{n+q}$, tighter bounds may be inferred on x. In MILP, a special case of this procedure allows tightening of the bounds of a set of variables in a bounded linear expression. Suppose $x_{h+1} = a_0 + \sum_{i=1}^{h} a_i x_i$, where $x_i^l \leq x_i \leq x_i^u$ for all $i \in \{1, 2, \ldots, h, h+1\}$. Let us denote $H_i^+ = \{j \in 1, 2, \ldots, h : j \neq i, a_j > 0\}$ and $H_i^- = \{j \in 1, 2, \ldots, h : j \neq i, a_j < 0\}$. Then a common preprocessing step is as follows:

$$\begin{aligned}
\forall i \in \{1, 2, \ldots, h\} : a_i > 0,\ x_i &\geq \tfrac{1}{a_i}\left(x_{h+1}^l - \textstyle\sum_{j \in H_i^+} a_j x_j^u - \sum_{j \in H_i^-} a_j x_j^l - a_0\right), \\
x_i &\leq \tfrac{1}{a_i}\left(x_{h+1}^u - \textstyle\sum_{j \in H_i^+} a_j x_j^l - \sum_{j \in H_i^-} a_j x_j^u - a_0\right); \\
\forall i \in \{1, 2, \ldots, h\} : a_i < 0,\ x_i &\geq \tfrac{1}{a_i}\left(x_{h+1}^u - \textstyle\sum_{j \in H_i^+} a_j x_j^l - \sum_{j \in H_i^-} a_j x_j^u - a_0\right), \\
x_i &\leq \tfrac{1}{a_i}\left(x_{h+1}^l - \textstyle\sum_{j \in H_i^+} a_j x_j^u - \sum_{j \in H_i^-} a_j x_j^l - a_0\right).
\end{aligned}$$

In general, for a variable $x_{h+1} = \vartheta_{h+1}(x_1, x_2, \ldots, x_h)$, one aims at improving the bounds of each $x_i, i = 1, 2, \ldots, h$, by using the bounds of x_{h+1} and of all other variables in the expression:

$$\begin{aligned}
x_i \geq \min\{x_i : x \in B, x_{h+1}^l \leq \vartheta_{h+1}(x_1, x_2 \ldots, x_h) \leq x_{h+1}^u\} \quad \forall i = 1, 2 \ldots, h, \\
x_i \leq \max\{x_i : x \in B, x_{h+1}^l \leq \vartheta_{h+1}(x_1, x_2 \ldots, x_h) \leq x_{h+1}^u\} \quad \forall i = 1, 2 \ldots, h.
\end{aligned}$$

For monotone, univariate functions $x_j = \vartheta_j(x_i)$, this simplifies to

$$x_j^l \leq x_j \leq x_j^u \Rightarrow \begin{cases} \vartheta_j^{-1}(x_j^l) \leq x_i \leq \vartheta_j^{-1}(x_j^u) & \text{if } \vartheta_j(\cdot) \text{ is increasing,} \\ \vartheta_j^{-1}(x_j^u) \leq x_i \leq \vartheta_j^{-1}(x_j^l) & \text{if } \vartheta_j(\cdot) \text{ is decreasing,} \end{cases}$$

while the procedure is more involved for multivariate operators. The procedure, which consists of repeatedly alternating *upward* and *downward* bound propagation, is sketched in Table 2. Notice that a limit *max_iter* on the number of iterations is set as an iteration may produce only very small improvements. For example, consider two constraints $x_1 = ax_2$ and $ax_1 = x_2$, with $0 < a < 1$ and both variables bounded in $[0, 1]$. Although the only feasible value is $(0, 0)$, at iteration p the bounds on both x_1 and x_2 are tightened to $[0, a^{-p}]$, thus making the procedure enter an infinite loop. The complexity of each iteration of the *do-while* loop in Table 2 can be estimated as linear in the number of variables as long as only univariate and bivariate functions are used in $\mathbf{P}$.

Consider again the DAG $G = (V, A)$ associated with $\mathbf{P}$. An iteration of the procedure outlined above scans the DAG from the bottom up, starting from the nodes with no outgoing arcs, whose variables have bounds as given initially or changed through branching rules. Then the downward pass is performed, where the bounds on the variables with no entering arcs are used to infer tighter bounds on other variables.

Input	problem $\mathbf{P}$, initial bounding box B_0
Output	tightened bounding box B

$B \leftarrow B_0$
$iter \leftarrow 0$
do
  $tightened \leftarrow$ false
  apply upward propagation to $(\mathbf{P}, B)$
  if infeasible, return $\emptyset$
  if at least one bound has tightened
    apply downward propagation to $(\mathbf{P}, B)$
    if infeasible, return $\emptyset$
    if at least one bound has tightened,
      $tightened \leftarrow$ true
  $iter \leftarrow iter + 1$
while $iter < max\_iter \wedge tightened =$ true
return B

Table 2. Feasibility-based Bounds tightening (FBBT).

4.3. *Aggressive FBBT*

Suppose that, at an sBB node k with bounding box $B = [x^l, x^u]$, a local minimum $\hat{x}^k \in B$ of $\mathbf{CP}_k$ is known. In order to check if it is worth restricting the search around $\hat{x}^k$, FBBT is applied to portions of B that exclude $\hat{x}^k$. The portions yielding a problem that is infeasible or a lower bound above the cutoff value (the objective value at $\hat{x}^k$) can be excluded, and tighter bounds are obtained.

COUENNE implements this on a per-variable basis: for each variable x_i, FBBT is applied to the fictitious bounding box $B \cap \{x : x_i \leq \tilde{x}_i\}$, where a suitable $\tilde{x}_i$ is chosen in $]x_i^l, \hat{x}_i^k[$. If the resulting problem is infeasible or has lower bound above the best upper bound, then $\tilde{x}_i \leq x_i \leq x_i^u$ is a valid tightening. Analogously, if a value $\tilde{x}_i \in]\hat{x}_i^k, x_i^u[$ is chosen and the FBBT applied to the bounding box $B \cap \{x : x_i \geq \tilde{x}_i\}$ proves that the problem is infeasible, then the tightening $x_i^l \leq x_i \leq \tilde{x}_i$ is valid.

This technique, called *Aggressive Bounds Tightening* (ABT), is analogous to *probing* techniques used in both MILP [72] and MINLP [78]. Similarly to OBBT, ABT is also time consuming as it requires several executions of FBBT for each variable of $\mathbf{P}_k$; also, it may require a call to an NLP solver for computing a new local optimum if one is not available in the bounding box B. It is used in all nodes of the sBB tree up to a certain depth, specified by a parameter L_{abt} that is typically 2 in COUENNE, and with probability $2^{L_{\text{abt}}-\lambda}$ for nodes at depth $\lambda > L_{\text{abt}}$.

4.4. *Reduced-cost bounds tightening*

Reduced-cost bounds tightening, introduced for solving MILP problems [59], has also been developed for MINLP [68, 69]; we briefly describe it here. Assume we are given a lower bound $\bar{z}^k$ for $\mathbf{P}_k$, an upper bound $\hat{z}$ for $\mathbf{P}$, both finite, an optimal solution $\bar{x}^k$ to $\mathbf{LP}_k$, and the vector of (nonnegative) reduced costs $(d_i)_{i=1,2,\ldots,n+q}$. For each i such that $\bar{x}_i^k$ is one of the bounds for x_i, the following tightening is valid:

$$\begin{aligned} \bar{x}_i^k = x_i^l, d_i > 0 \quad &\Rightarrow \quad x_i \leq x_i^l + \tfrac{\hat{z} - \bar{z}^k}{d_i}, \\ \bar{x}_i^k = x_i^u, d_i > 0 \quad &\Rightarrow \quad x_i \geq x_i^u - \tfrac{\hat{z} - \bar{z}^k}{d_i}. \end{aligned}$$

5. Branching techniques

An effective branching strategy aims at minimizing the size of the sBB tree and can strongly affect the performance of an sBB algorithm. As it is difficult to predict the size of the sBB tree for a given strategy, surrogate criteria are used to design branching strategies: an sBB node k is partitioned

- to improve the lower bound of the resulting two (or more) subproblems, or
- to create subproblems of similar difficulty to keep the sBB tree balanced, or
- to eliminate a portion as large as possible of the solution set.

These three goals conflict with one another, and putting emphasis on one or the other leads to alternative branching strategies that appear to be reasonable. Empirical testing of these strategies is then necessary to evaluate their performance.

If the lower bound $\bar{x}^k_{n+q}$ of an sBB node $\mathbf{P}_k$ is smaller than the best known feasible solution value and $\bar{x}^k$ is infeasible for $\mathbf{P}_k$, branching may be required. In this section, we only consider branching on a variable, although other branching strategies are possible. When branching on a variable, one picks a variable x_i, a lower branching point B_l, and a upper branching point B_u with $B_l \leq B_u$. The two subproblems generated by branching are obtained by adding the constraint $x_i \leq B_l$ in the first one and $x_i \geq B_u$ in the second one.

If x_i is an integer variable but the value $\bar{x}^k_i$ is fractional, a usual choice is $B_l = \lfloor \bar{x}^k_i \rfloor$ and $B_u = \lceil \bar{x}^k_i \rceil$. This is the simple branching scheme for MILP described in Section 1. If x_i is a continuous variable, it might be necessary to branch on x_i as illustrated in the following example. Consider the function $x_j = \vartheta_j(x_i) = x_i^2$ and an LP solution $\bar{x}^k$. Assume that $(\bar{x}^k_i, \bar{x}^k_j)$ is inside the linearization of $x_j = x_i^2$ and that $\bar{x}^k_j > (\bar{x}^k_i)^2$ (see Figure 3a). As $(\bar{x}^k_i, \bar{x}^k_j)$ is not separable with a linearization cut, branching is necessary. Here, branching is carried out on x_i with $B_l = B_u = x^b_i$ and it results in two linearizations, both excluding $(\bar{x}^k_i, \bar{x}^k_j)$ (see Figure 3b).

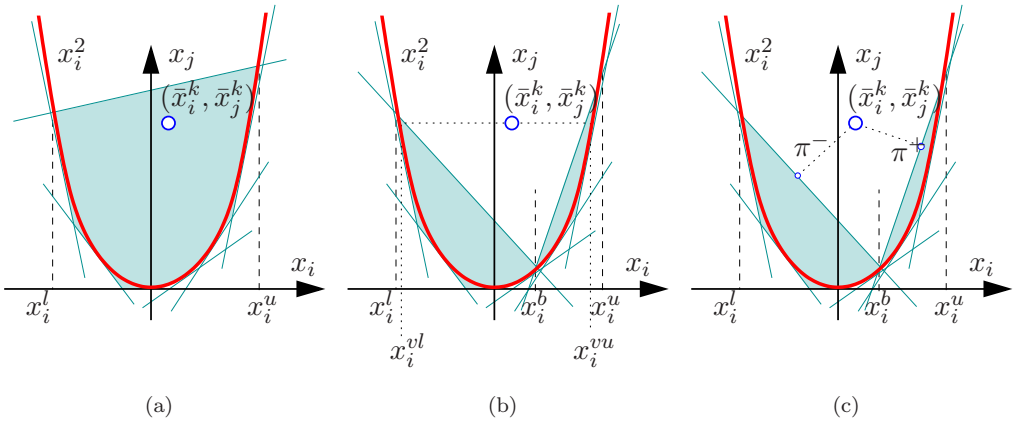


Figure 3. Branching on a nonlinear expression.

The decisions to be made when branching are (i) the selection of the branching variable and (ii) the choice of branching points. In MILP, while the latter is trivial, the former is driven by the aim of reaching a feasible solution as soon as possible: one selects an integer variable x_i whose value $\bar{x}^k_i$ is fractional, i.e. a variable that violates one of the constraints that were relaxed. A similar decison can be taken in the MINLP case. However, it could happen that $\bar{x}^k_i$ is integer for all $i \in N_I$, while $\bar{x}^k$ is not feasible for $\mathbf{P}_k$, as illustrated in Figure 3. It is thus necessary to introduce a measure of infeasibility for a continuous variable x_i.

If $x_i = \vartheta_i(x)$ is a constraint of $\mathbf{P}$, we define the ϑ_i-*infeasibility* of an auxiliary variable x_i at node k as the discrepancy between $\bar{x}_i^k$ and $\vartheta_i(\bar{x}^k)$, scaled with the norm of the gradient of $\vartheta_i(\cdot)$ at $\bar{x}^k$, and denote it with $U_i(\bar{x}^k) = \frac{|\bar{x}_i^k - \vartheta_i(\bar{x}^k)|}{1+||\nabla\vartheta_i(\bar{x}^k)||_2}$. Scaling U_i with $1 + ||\nabla\vartheta_i(\cdot)||_2$ avoids selecting a variable x_i with a small bound interval $[x_i^l, x_i^u]$ (unlikely to improve the linearization if selected for branching) but with a large $|\bar{x}_i^k - \vartheta_i(\bar{x}^k)|$. For example, consider $x_i = e^{x_j}$ with $x_j \in [M, M+\epsilon]$ and $M = 20$, $\epsilon = 10^{-3}$; a linearization of $x_i = e^{x_j}$ contains a point $(\bar{x}_j^k, \bar{x}_i^k)$ such that $\bar{x}_i^k - e^{\bar{x}_j^k} = e^{M+\epsilon/2}(e^{\epsilon/2} - 1) \approx 60$, and which is in general proportional to e^M; the scaled measure is $e^{\epsilon/2}(e^{\epsilon/2} - 1) \approx \epsilon/2$, independent of M.

The concept of ϑ_i-infeasibility has a direct parallel in MILP: given a solution $\bar{x}^k$ to a linear relaxation, an integer variable x_i has infeasibility $U_i(\bar{x}^k) = \min(\bar{x}_i^k - \lfloor\bar{x}_i^k\rfloor, \lceil\bar{x}_i^k\rceil - \bar{x}_i^k)$, and a solution is (integer) feasible when all variables have zero infeasibility. In the MINLP case, $\bar{x}^k$ is feasible if and only if all integer variables have integer values, and the infeasibility of all variables is 0.

5.1. *Branching-variable selection*

Branching variable selection in COUENNE is subject to priorities. Integer variables have higher priority than continuous variables, hence if there are integer variables with a fractional value in $\bar{x}_k$, one of them is chosen for branching. The selection of integer variables for branching is out of the scope of this work; suffice it to say that COUENNE applies standard MILP techniques to select integer branching variables.

From now on, we assume that no integer variable has fractional value and that continuous variables are the only candidates for branching. The key difference with MILP branching is that the infeasibility of a continuous variable is the outcome of a violated (nonlinear) definition of a variable, $x_i \neq \vartheta_i(x)$. Further notation is needed: we define the *dependence set* of x_i, denoted as $D(x_i)$, the set of variables in x on which x_i depends directly, that is, the set of variables appearing in $\vartheta_i(x)$. For instance, if $x_{37} = x_5 x_{11}$, then $D(x_{37}) = \{x_5, x_{11}\}$, regardless of x_5 or x_{11} being auxiliary variables themselves or not. The infeasibility of a continuous variable can hence be associated with x_i itself or with all variables of $D(x_i)$. Because the result of the decision process must be a variable rather than an expression, we have chosen a measure that is associated with each variable x_i, and that is related to all expressions depending on it. To this purpose, consider a variable x_i and the set $E(i) = \{j \in N : x_i \in D(x_j)\}$ of variables depending on x_i; we define the *nonlinear infeasibility* as the following linear combination:

$$\begin{aligned}\Omega_i^N(\bar{x}^k) = \quad & \mu_1 \quad \textstyle\sum_{j\in E(i)} U_j(\bar{x}^k) + \\ & + \mu_2 \max_{j\in E(i)} U_j(\bar{x}^k) + \\ & + \mu_3 \ \min_{j\in E(i)} U_j(\bar{x}^k),\end{aligned}$$

with parameters $\mu_k \geq 0, k = 1, 2, 3$ and $\mu_1 + \mu_2 > 0$ to ensure that $\bar{x}^k$ is infeasible if and only if $\Omega_i^N(\bar{x}^k) > 0$ for at least one variable x_i. COUENNE uses $(\mu_1, \mu_2, \mu_3) = (0.1, 1.3, 0.8)$. A simple branching variable selection procedure, that we call **br-plain**, returns the variable x_i with largest $\Omega_i^N(\bar{x}^k)$. However, as $\Omega_i^N(\bar{x}^k)$ is hardly correlated with the improvement of the lower bound that the two new subproblems would return when branching on x_i, a more sophisticated choice is advised. This is similar to the observation that for MILP problems, branching on the *most fractional* variable is a poor choice [2].

To this purpose, we implemented two strategies: the first one, *Violation Transfer*, described in the next section, has been proposed by Tawarmalani and Sahinidis [79].

The second one, *reliability branching*, is an extension of the reliability branching in MILP and is discussed in Section 5.3.

5.2. *Violation Transfer*

Tawarmalani and Sahinidis [78], [79, §6.2.1] present an algorithm, called *Violation Transfer*, to estimate the impact of a variable on the current status of the problem, when all other variables are fixed.

Violation Transfer works with the Lagrangian function of a relaxation of $\mathbf{P}_k$, defined as $\mathcal{L}^k(x) = \bar{f}(x) + \sum_{j\in H} \pi_j \bar{g}_j(x)$, where π_j's are nonnegative multipliers while $\bar{f}(x)$ and $\bar{g}_j(x), j \in H$ give a convex relaxation $\min\{\bar{f}(x) : \bar{g}_j(x) \leq 0, j \in H\}$ of $\mathbf{P}_k$. It also assumes that a solution $\bar{x}^k$ to that relaxation is given.

The smallest interval $[x_i^{vl}, x_i^{vu}] \subseteq [x_i^l, x_i^u]$ containing $\bar{x}_i^k$ is sought such that, for each $j \in E(i)$, there exists a point $\tilde{x}_{ij}$ in $[x_i^{vl}, x_i^{vu}]$ such that $\bar{x}_j^k = \vartheta_j(\tilde{x}_{ij})$. In other words, given the feasible solution $\bar{x}^k$, all variables x_j defined as functions of x_i are *feasible* for at least one point in the interval. Figure 3b depicts the interval $I_0 = [x_i^{vl}, x_i^{vu}]$ for nonlinear $x_j = (x_i)^2$. If other variables $x_{j_1}, x_{j_2}, \ldots, x_{j_h}$ depend on x_i, the smallest interval containing all corresponding values $\tilde{x}_{ij}$ is chosen.

Then, the maximum and the minimum of $\mathcal{L}^k(x)$ are estimated on the set $X_R^i = \{x : x_j = \bar{x}_j^k \, \forall j \neq i, x_i \in [x_i^{vl}, x_i^{vu}]\}$; notice that here all variables are fixed except x_i. The assumption is that branching on a variable x_i for which these minima and maxima are far apart is likely to improve the lower bound in the two subproblems. If a linear relaxation $\min\{x_{n+q} : a_j x \geq b_j, j \in H\}$ is considered, then $\bar{f}(x) = x_{n+q}$, $\bar{g}_j(x) = b_j - a_j x$. Assume that π is the vector of optimal dual variables of the linear constraints. Denoting $\gamma_i^{\text{vt}} = x_i^{vu} - x_i^{vl}$, an estimate of the change in the lower bound is made through a linear optimization problem on a segment, X_R^i:

$$
\begin{aligned}
\widetilde{\Omega}_i^{\text{vt}} &= \max_{x\in X_R^i} \mathcal{L}^k(x) - \min_{x\in X_R^i} \mathcal{L}^k(x) = \\
&= \max_{x\in X_R^i} \left(\bar{f}(x) + \textstyle\sum_{j\in H} \pi_j \bar{g}_j(x)\right) - \min_{x\in X_R^i} \left(\bar{f}(x) + \textstyle\sum_{j\in H} \pi_j \bar{g}_j(x)\right) = \\
&= \max_{x\in X_R^i} \left(x_{n+q} + \textstyle\sum_{j\in H} \pi_j (b_j - a_j x)\right) - \min_{x\in X_R^i} \left(x_{n+q} + \textstyle\sum_{j\in H} \pi_j (b_j - a_j x)\right) = \\
(\star) &= \max_{x_i\in[x_i^{vl}, x_i^{vu}]} \left(-\textstyle\sum_{j\in H} \pi_j a_{ji}\right) x_i - \min_{x_i\in[x_i^{vl}, x_i^{vu}]} \left(-\textstyle\sum_{j\in H} \pi_j a_{ji}\right) x_i = \\
&= \left|\left(x_i^{vu} - x_i^{vl}\right) \textstyle\sum_{j\in H} \pi_j a_{ji}\right| = \\
&= \gamma_i^{\text{vt}} \left|\textstyle\sum_{j\in H} \pi_j a_{ji}\right|,
\end{aligned}
$$

where $(\star)$ holds because all dropped terms are constant within X_R^i. The term $\left|\sum_{j\in H} \pi_j a_{ji}\right|$ is the absolute value of the reduced cost of variable x_i in $\mathbf{LP}_k$, which has zero coefficient in the objective function. This algorithm is hence similar to reduced cost branching, where the non-zero reduced cost of variable x_i is weighted with γ_i^{vt}. As an alternative to the absolute value of the reduced cost, $\left|\sum_{j\in H} \pi_j a_{ji}\right|$, Tawarmalani and Sahinidis [79] use $\sum_{j\in H} |\pi_j a_{ji}|$. Accordingly, we define

$$\Omega_i^{\text{vt}} = \gamma_i^{\text{vt}} \textstyle\sum_{j\in H} |\pi_j a_{ji}|$$

and select the variable with largest Ω_i^{vt} for branching in COUENNE.

5.3. *Reliability branching*

Strong Branching, introduced by Applegate et al. [11] in the 1990s for solving

Traveling Salesman problems, is a procedure to select a branching rule among a set of candidates. We first give a simplified description of strong branching in the case of MILP.

Consider a MILP problem whose n variables are integer, **ILP** : $\min\{cx : x \in X \cap \mathbb{Z}^n\}$, where $X = \{x \in \mathbb{R}^n : Ax \geq b\}$. Let $\bar{x}$ be an optimal solution of its linear relaxation **LP** : $\min\{cx : x \in X\}$. Assume that the set $F = \{i : \bar{x}_i \notin \mathbb{Z}\}$ is nonempty. For each $i \in F$, we potentially can branch on x_i, creating subproblems with relaxations $LP(i,-) : \min\{cx : x \in X, x_i \leq \lfloor \bar{x}_i \rfloor\}$ and $LP(i,+) : \min\{cx : x \in X, x_i \geq \lceil \bar{x}_i \rceil\}$. The strong branching procedure solves, for all $i \in F$, $LP(i,-)$ and $LP(i,+)$ obtaining their respective optimal values $z(i,-)$ and $z(i,+)$. It then computes a score associated with each potential branching and selects the variable with highest score. The score is computed as follows: Denote φ_i^- and φ_i^+ the nonnegative improvements in the optimal values associated with the two subproblems, i.e., $\varphi_i^- = z(i,-) - c\bar{x}$ and $\varphi_i^+ = z(i,+) - c\bar{x}$. The score for branching on variable x_i is then

$$\alpha \max\{\varphi_i^-, \varphi_i^+\} + (1-\alpha)\min\{\varphi_i^-, \varphi_i^+\} \tag{1}$$

where the weight $\alpha \in [0,1]$ is found empirically [2] and is set to 0.15 in COUENNE.

Strong branching is very efficient in reducing the size of the enumeration tree in MILP, but requires a lot of time. Variants to improve speed while not degrading performances too much, such as solving the subproblems with a limit on the number of dual simplex pivots, can be used. Another possibility is to estimate the effect of branching rules using *pseudocosts* [15], avoiding the solution of several LPs at each node. One of the most commonly used techniques today is *reliability branching*, introduced by Achterberg et al. [2]: strong branching is only applied to the initial nodes of the enumeration tree. Then, statistics are collected to estimate the improvement of the lower bound per unit of change in a branching variable; these estimates are called pseudocosts and are used in subsequent nodes instead of computing φ_i^- and φ_i^+. More precisely, after strong branching is performed at the root, the pseudocosts for variable x_i are initialized to:

$$\psi_i^- = \frac{\varphi_i^-}{\bar{x}_i - \lfloor \bar{x}_i \rfloor}, \qquad \psi_i^+ = \frac{\varphi_i^+}{\lceil \bar{x}_i \rceil - \bar{x}_i}.$$

notice that the change in the lower bound is divided by the change of the branching variable, $\delta_i^- = \bar{x}_i - \lfloor \bar{x}_i \rfloor$ and $\delta_i^+ = \lceil \bar{x}_i \rceil - \bar{x}_i$. This scaling yields a measure that is independent of the distance of $\bar{x}_i$ to the closest integer.

In general, pseudocosts are initialized using the information provided by strong branching at the low-depth sBB nodes, and updated using the optimal values of the relaxations corresponding to actual nodes of the sBB tree as follows. After selecting a branching variable x_i and performing the actual branching, the pseudocosts ψ_i^- and ψ_i^+ are updated by averaging the change in the lower bound per unit of change in the branching variable. At any point of the enumeration, consider the sets K_i^- and K_i^+ of sBB nodes that result from applying the branching rules $x_i \leq x_i^b$ and $x_i \geq x_i^b$, respectively, on variable x_i; consider also the resulting change in the lower bounds, ψ_i^{k-} and ψ_i^{k+}, and the multipliers δ_i^{k-} and δ_i^{k+}, obtained when creating nodes of K_i^- and K_i^+. Then the current pseudocosts for x_i are given by:

$$\psi_i^- = \frac{1}{\left|K_i^-\right|} \sum_{k \in K_i^-} \frac{\varphi_i^{k-}}{\delta_i^{k-}} \qquad \psi_i^+ = \frac{1}{\left|K_i^+\right|} \sum_{k \in K_i^+} \frac{\varphi_i^{k+}}{\delta_i^{k+}}.$$

As soon as $|K_i^-|$ and $|K_i^+|$ are large enough, pseudocosts ψ_i^- and ψ_i^+ for x_i are deemed reliable and are no longer updated.

Pseudocosts are computed at the first nodes of the sBB algorithm and used to estimate the improvement of the objective function for a given branching variable. We provide here a simplified presentation where pseudocosts are only computed at the root node and updated at all other nodes. Suppose that the solution $\bar{x}^k$ to the relaxation $\mathbf{LP}_k$ of a subsequent node $\mathbf{P}_k$ is fractional, and the fractional components of $\bar{x}^k$ are indexed by $F' \subseteq \{1, 2, \ldots, n\}$. Then the branching variable is selected as in (1), where the improvements are not computed by solving an LP, but estimated by multiplying the pseudocosts by $\delta_i^{k-} = (\bar{x}_i^k - \lfloor \bar{x}_i^k \rfloor)$ and $\delta_i^{k+} = (\lceil \bar{x}_i^k \rceil - \bar{x}_i^k)$:

$$\varphi_i^- = \psi_i^- \delta_i^{k-}, \qquad \varphi_i^+ = \psi_i^+ \delta_i^{k+}.$$

5.4. *Pseudocosts in MINLP*

Generalizing pseudocosts for integer variables in MINLP problems is immediate, as the same method described in the previous section can be used. However, the fact that branching on continuous variables might occur requires modification of the use of the pseudocosts, and in particular of the multiplying factors δ_i^{k-} for ψ_i^- and δ_i^{k+} for ψ_i^+ at node $\mathbf{P}_k$, values that cannot be set to $\delta_i^{k-} = (\bar{x}_i^k - \lfloor \bar{x}_i^k \rfloor)$ and $\delta_i^{k+} = (\lceil \bar{x}_i^k \rceil - \bar{x}_i^k)$.

For MILP, we can interpret δ_i^{k-} and δ_i^{k+} as the predicted change of x_i in the LP solutions of the subproblems generated by the branching rule. For MINLP, however, branching rules applied at node $\mathbf{P}_k$ do not imply a nonzero change in the value of x_i. Indeed, when branching on a continuous variable x_i, it might happen that the optimal solution of one of the subproblems (and even both of them when $\bar{x}_i^k$ is the branching point) still has $x_i = \bar{x}_i^k$.

Consider a candidate variable x_i and a branching point $x_i^b \in [x_i^l, x_i^u]$. Also, define $\bar{x}^{k-}$ and $\bar{x}^{k+}$ the solutions of the linear relaxation at the left and the right subproblems, respectively. We study five definitions of δ_i^{k-} for ψ_i^- and δ_i^{k+} for ψ_i^+:

rb-inf: $\delta_i^{k-} = \delta_i^{k+} = \Omega_i(\bar{x}^k)$: the multipliers are both set to the *infeasibility* of a variable, analogously to MILP.

rb-int-br: $\delta_i^{k-} = x_i^b - x_i^l$; $\delta_i^{k+} = x_i^u - x_i^b$. This strategy considers the new range of x_i when estimating the improvement in the lower bound. When x_i^l or x_i^u are infinite, the corresponding multiplier is set to $\Omega_i(\bar{x}^k)$. This measure takes into account the size of the feasible set of each subproblem (when projected on x_i). There are three variants of *rb-int-br*:

- *rb-int-br-rev*: $\delta_i^{k-} = x_i^u - x_i^b$; $\delta_i^{k+} = x_i^b - x_i^l$. This inverts *rb-int-br* and accounts for situations where the half-intervals are unbalanced: if $x_i^b \approx x_i^u$, the right branch is a much tighter problem with a (possibly) much better lower bound, as opposed to the left branch which is almost unchanged. This suggests that the multiplier of the pseudocost should be larger for the right branch and smaller for the left branch.
- *rb-int-lp* and *rb-int-lp-rev*: the multiplier is computed based on the LP point $\bar{x}_i^k$ rather than x_i^b, i.e., $\delta_i^{k-} = \bar{x}_i^k - x_i^l$; $\delta_i^{k+} = x_i^u - \bar{x}_i^k$ and $\delta_i^{k-} = x_i^u - \bar{x}_i^k$; $\delta_i^{k+} = \bar{x}_i^k - x_i^l$, respectively. To avoid null multipliers when $\bar{x}_i^k$ is at one of the bounds, the actual value used is $\max\{\min\{\bar{x}_i^k, x_i^u - \rho\}, x_i^l + \rho\}$ where $\rho = \beta(x_i^u - x_i^l)$ and $0 < \beta < \frac{1}{2}$. In COUENNE, β is set to 0.05.

rb-lpdist: $\delta_i^{k-} = ||\bar{x}^k - \bar{x}^{k-}||_2$; $\delta_i^{k+} = ||\bar{x}^k - \bar{x}^{k+}||_2$, i.e., the Euclidean distance between the LP solution of the parent node and that of the subprob-

lem. This is only available when updating the pseudocosts, after solving the LP of the new subproblem; also, this gives an estimate of the change for the vector x rather than only for x_i.

rb-proj: Consider Figure 3c, where branching on x_i at x_i^b results in two new linearizations. Each contains the projection onto the (x_i, x_j) space of the linearization obtained after the respective branching. A lower bound on the distance between the previous LP point $(\bar{x}_i^k, \bar{x}_j^k)$ and the two new linearizations is the distance between $(\bar{x}_i^k, \bar{x}_j^k)$ and $\pi^- = (x_i^{\pi-}, x_j^{\pi-})$, $\pi^+ = (x_i^{\pi+}, x_j^{\pi+})$, therefore $\delta_i^{k-} = ||(\bar{x}_i^k, \bar{x}_j^k) - (x_i^{\pi-}, x_j^{\pi-})||_2$; $\delta_i^{k+} = ||(\bar{x}_i^k, \bar{x}_j^k) - (x_i^{\pi+}, x_j^{\pi+})||_2$. If other variables than x_j depend on x_i, the maximum distance among all these variables is considered.

rb-vt: Consider the interval $\gamma_i^{\text{vt}} = x_i^{vu} - x_i^{vl}$ used in Violation Transfer (see Section 5.2). The multipliers are set to $\delta_i^{k-} = x_i^b - x_i^{vl}$ and $\delta_i^{k+} = x_i^{vu} - x_i^b$.

5.5. *Branching-point selection*

We tested three branching point selection strategies. For the sake of simplicity, consider a branching variable x_i appearing in one single expression $x_j = \vartheta_j(x_i)$, and assume that x_i is continuous. One seemingly desirable attribute for a branching rule is that the resulting sBB subtree be balanced, hence a reasonable branching point selection is based on the areas of the resulting linearizations (See Fig. 4b).

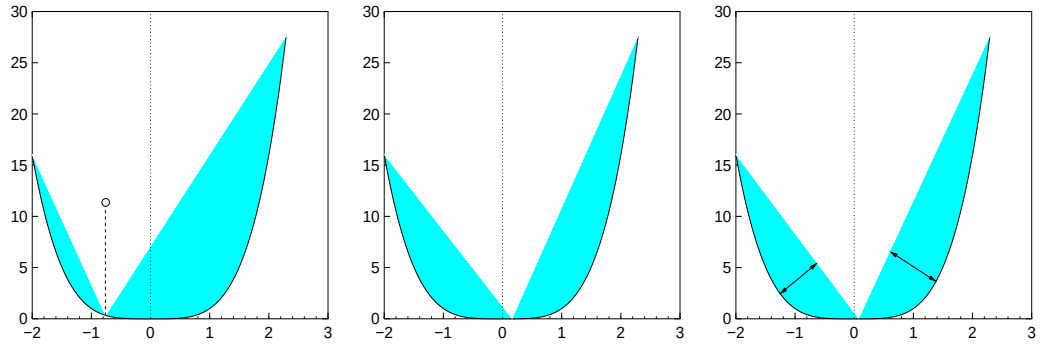
LP-based strategy. An obvious goal of branching is to make sure that $\bar{x}^k$ becomes infeasible in both subproblems. Selecting branching point $\bar{x}_i^k$ for a variable x_i with positive infeasibility will ensure that, as the linearization is exact at the bounds of x_i. However, selecting a branching point too close to the bounds is likely to create a very easy subproblem and a very hard one (see Figure 4a). Thus, for a variable x_i with bounds $[x_i^l, x_i^u]$ it is usual [78] to use as branching point a convex combination of $\bar{x}_i^k$ and the midpoint $x_i^m = \frac{1}{2}(x_i^l + x_i^u)$. Therefore, a selection strategy that guarantees a minimum distance from the variable bounds is as follows:

$$x_i^b = \max\{x_i^l + b, \min\{x_i^u - b, \alpha\bar{x}_i^k + (1-\alpha)x_i^m\}\}, \tag{2}$$

where $0 < \alpha < 1$ and $b = \beta(x_i^u - x_i^l)$ for $0 < \beta < \frac{1}{2}$. COUENNE sets α and β to 0.25 and 0.2, respectively — notice that this default setting makes $x_i^l + b$ and $x_i^u - b$ irrelevant in (2). This strategy aims at balancing the half-intervals of x_i as a way to balance the difficulty of the subproblems.

If a local optimum $\hat{x}^k$ is known, $\hat{x}_i^k$ can be chosen as a branching point for variable x_i. This technique proves to be effective in some cases, as shown by Shectman and Sahinidis [74]. They describe a separable concave minimization problem for which the branching point is chosen as the value of variables at a local minimum. The resulting convexification, which is exact at the new bounds enforced by branching, happens to return a solution $\bar{x}^k$ very close to $\hat{x}^k$ and thus quickly closes the gap.

Expression-based strategies. Consider a convex, univariate function, $x_j = \vartheta_j(x_i)$, and a set of linearization inequalities, $LP = \{(x_i, x_j) : a_h x_i + b_h x_j \leq c_h,\ h = 1, 2 \ldots, H\}$; for brevity, we denote it as $LP = \{(x_i, x_j) : ax_i + bx_j \leq c\}$ with $a = (a_h)_{h=1}^H$, $b = (b_h)_{h=1}^H$, and $c = (c_h)_{h=1}^H$ vectors of order H. For a branching variable x_i, we consider the impact of the branching point x_i^b on the two linearizations that arise from branching on x_i, i.e., $LP' = \{(x_i, x_j) : a'x_i + b'x_j \leq c'\}$ from $x_i \leq x_i^b$



(a) Current LP point (b) LP-midpoint combination (c) A *balanced* strategy

Figure 4. Strategies for selecting a branching point.

and $LP'' = \{(x_i, x_j) : a''x_i + b''x_j \leq c''\}$ from $x_i \geq x_i^b$, with vectors a', b', c' of size H' and vectors a'', b'', c'' of size H''.

Branching is required when $\bar{x}_j^k > \vartheta_j(\bar{x}_i^k)$, as illustrated in Figure 3. The branching point determines the shape and area of LP' and LP''. If one aims at reducing the sum of the areas of the resulting convexification, a *minimum-area* strategy can be applied. Note that this is equivalent to maximizing the area of LP that is cut by the subsequent linearizations. As ϑ_j is convex, this branching point is the unique solution of $\frac{\partial \vartheta_j}{\partial x_i}(x_i) = \dot{\vartheta}_j(x_i) = \frac{\vartheta_j(x_i^l) - \vartheta_j(x_i^l)}{x_i^u - x_i^l}$, namely

$$x_i^b = \dot{\vartheta}_j^{-1}\left(\frac{\vartheta_j(x_i^l) - \vartheta_j(x_i^l)}{x_i^u - x_i^l}\right). \tag{3}$$

This is the point on the curve, $(x_i^b, \vartheta_j(x_i^b))$, with maximum distance from the line defining the upper envelope of $\vartheta_j(x_i)$ in the interval $[x_i^l, x_i^u]$. The implementation for univariate functions is straightforward. For bilinear terms $x_k = x_i x_j$, it can be shown that the minimum volume of the resulting linearization is attained when branching on either x_i or x_j at the middle of the bound interval [14].

We point out that minimizing the sum of the areas of LP' and LP'' may help reduce the search space of the problem, but it does not necessarily make solving the subproblems of similar difficulty. The strategy described below aims at balancing the linearization areas. As discussed in Section 3, one inequality for each of LP' and LP'' is the upper envelope of ϑ_j; let us denote them as $a_0'x_i + b_0'x_j \leq c_0'$ and $a_0''x_i + b_0''x_j \leq c_0''$, respectively — note that the coefficients depend on x_i^b. Suppose that x_i^b is known; for any point $x_i \in [x_i^l, x_i^b]$ (resp. $[x_i^b, x_i^u]$), let us define $u'(x_i)$ (resp. $u''(x_i)$) as the distance between $(x_i, \vartheta_j(x_i))$ and the line $a_0'x_i + b_0'x_j = c_0'$ (resp. $a_0''x_i + b_0''x_j = c_0''$):

$$u'(x_i) = \frac{c_0' - a_0'x_i - b_0'\vartheta_j(x_i)}{\sqrt{(a_0')^2 + (b_0')^2}} \qquad u''(x_i) = \frac{c_0'' - a_0''x_i - b_0''\vartheta_j(x_i)}{\sqrt{(a_0'')^2 + (b_0'')^2}}.$$

The value of x_i that maximizes $u'(x_i)$ is obtained from (3) on the half-interval $[x_i^l, x_i^b]$. Analogously we obtain a value maximizing $u''(x_i)$. A *balanced* strategy finds a branching point through a binary search on the interval $[x_i^l, x_i^u]$ that minimizes the difference between the maximum $u'(x_i)$ and the maximum $u''(x_i)$, i.e.,

$$x_i^b \in \operatorname{argmin}\left|\max_{x_i \in [x_i^l, x_i^b]} u'(x_i) - \max_{x_i \in [x_i^b, x_i^u]} u''(x_i)\right|.$$

This strategy aims at balancing the maximum distance between the points on $\vartheta_j(\cdot)$ and the upper envelope, rather than the areas of the resulting linearizations (see Figure 4c).

6. Upper bounds

If an optimal solution $\bar{x}^k$ to $\mathbf{LP}_k$ is infeasible for $\mathbf{P}_k$, we may try to compute a feasible solution $\hat{x}^k$ by applying an NLP solver with an initial point x^0 to a modified problem in which all integer variables x_i for $i \in N_I$ are fixed to integer values.

We find this initial point by repeated calls to FBBT: if $\bar{x}_i^k$ is integer, then $x_i^0 = \bar{x}_i^k$; otherwise, FBBT is applied to two problems $\mathbf{P}_k(i,-)$ and $\mathbf{P}_k(i,+)$, where x_i^0 is set to $\lfloor \bar{x}_i^k \rfloor$ in $\mathbf{P}_k(i,-)$ and to $\lceil \bar{x}_i^k \rceil$ in $\mathbf{P}_k(i,-)_i'$. If the resulting lower bound in $\mathbf{P}_k(i,-)$ is smaller than in $\mathbf{P}_k(i,+)$ or if $\mathbf{P}_k(i,-)_i'$ is infeasible, then $x_i^0 = \lfloor \bar{x}_i^k \rfloor$, and analogously if the opposite happens. If both $\mathbf{P}_k(i,-)$ and $\mathbf{P}_k(i,+)$ are infeasible, the procedure terminates without result. This procedure is explained in full detail in a forthcoming paper [14]. To avoid excessive calls to the NLP, this procedure is used at all nodes up to a certain depth L_{nlp} of the sBB tree, which is set to 2 by default in COUENNE. At a depth λ beyond L_{nlp}, it is used with probability $2^{\lambda - L_{\text{nlp}}}$.

7. Computational tests

We compare several variants of COUENNE with combinations of bounds tightening and branching techniques. We first compare our sBB with and without all tightening methods described above, to assess the utility of each independently. Then, we test our implementations of reliability branching against the better known Violation Transfer algorithm and against a very plain branching scheme. We also briefly report on the branching point selection strategies described in Section 5.5. Finally, we compare COUENNE with one of the state-of-the-art packages for Global Optimization, BARON [71]; see Section 7.3.

Our testbed consists of 84 instances from publicly available MINLP libraries:

minlplib: a collection of MINLP instances, convex and nonconvex [20];
Misc: various MINLP instances, two of them convex (*CLay** [64]); *barton* is a nonconvex MINLP for production planning in the natural gas industry; *airConduct* is a 2D bin-packing problem for the design of air conducts [34];
conic MINLP: several Mixed-Integer Conic Programming problems from the testbed of a MICP solver [83];
MacMINLP: a collection of MINLP problems, convex and nonconvex [43];
Nonconvex: a collection of nonconvex MINLPs [64];
MIQQP: Mixed-Integer quadratically constrained quadratic programs [55]; model `qpsi.mod` was used;
globallib: a collection of continuous NLP problems [51];
boxQP: continuous nonconvex box-constrained quadratic problems [82];
QCQP: continuous quadratically constrained quadratic programs [56]; among the three available models, `qcqp.mod` was used.

We also include six real-world instances of problems arising in Chemical Engineering: *nConvPl*, a nonconvex MINLP for the design of a water treatment plant [70], and five *Hicks* instances of a non-convex NLP problem of design and control of a production plant [30]. Tables 3 and 4 below describe characteristics of the problems: number of variables (*var*), of integer variables (*int*), of constraints (*con*), and of auxiliary variables created (*aux*), i.e., $|N \setminus N_0|$. A time limit of two hours

for solving any instance is used in all tests.

COUENNE is available on the COIN-OR repository, and it is distributed under the Common Public License [62]. A general description is given at `https://projects.coin-or.org/Bonmin/wiki/Boncouenne`. The release used for the tests is COUENNE-0.9[1]. All tests reported below, except those that compare COUENNE with BARON, have been conducted on a Linux machine equipped with a 2GHz Opteron processor, 3 GB of memory, and `gcc` compiler version 3.3.3.

COUENNE relies on other programs, some of them Open-Source and part of the COIN-OR software framework: continuous relaxations $\mathbf{CP}_k$ are solved by IPOPT [84], while the sBB is implemented using routines of CBC and the LP solver to solve relaxations $\mathbf{LP}_k$ is CLP. The generator of linearization inequalities is a specialization of the CGL library for cut generation. COUENNE also uses software not from the COIN-OR framework: BLAS, LAPACK, routines `ma27` and `mc19` from HSL, and the AMPL Solver Library (ASL).

name	var	con	aux	name	var	con	aux
GlobalLib (pure NLPs) [51]							
qp1	50	2	1277	ex5_2_5	32	19	110
qp2	50	2	1277	ex5_3_3	62	53	113
qp3	100	52	52	foulds3 [32]	168	48	193
QCQP (`qcqp.mod`) [56]							
dualc8	9	16	52	values	203	2	3824
dual4	76	2	2801	qbrandy	206	124	71
dual1	86	2	3560	qshare1b	221	111	91
dual2	97	2	4510	qetamacr	543	334	3012
qadlittl	97	54	129	gouldqp2	700	350	699
cvxqp1_s	101	51	397	primal4	1490	76	1565
cvxqp2_s	101	26	389	stcqp1	3159	1	4685
cvxqp3_s	101	76	406				
Box QPs [82]							
spar030-060-1	30	0	265	spar050-050-1	50	0	636
spar030-060-2	30	0	256	spar060-020-1	60	0	354
spar040-050-1	40	0	399	spar060-020-2	60	0	359
spar040-050-2	40	0	406	spar070-025-1	70	0	618
spar040-060-1	40	0	478	spar080-025-1	80	0	789
spar050-040-1	50	0	498	spar090-025-1	90	0	1012
Hicks [30]							
Hicks_5	83	68	226	Hicks_50	848	698	2251
Hicks_10	168	138	451	Hicks_100	1698	1398	4501
Hicks_20	338	278	901				

Table 3. NLP (continuous) instances used in the tests.

All tables report, for each instance and algorithm, either the running time if it is below the two-hour time limit, or the lower bound in brackets, and the best upper bound. A "–" means that no lower or upper bound is found after two hours, while "NUM" indicates numerical problems.

We have added performance profiles [24] to illustrate the performance of each variant of COUENNE. Performance profiles conveniently depict a comparison of a set S of algorithms on a set Π of instances. For each algorithm $s \in S$ and instance $p \in \Pi$, we define t_{sp} the solution time, or $+\infty$ if s could not solve p within the time limit. To compare running times, consider the minimum time $t_p^{\min} = \min_{s \in S} t_{sp}$;

[1] Available through `svn` at `https://projects.coin-or.org/svn/Bonmin/branches/Couenne-0.9-candidate`

name	var	int	con	aux	name	var	int	con	aux
MinlpLib [20]									
nvs19	8	8	8	45	fo7	112	42	211	226
nvs23	9	9	9	55	no7_ar2_1	112	42	269	283
du-opt5	18	11	6	221	no7_ar3_1	112	42	269	283
du-opt	20	13	8	222	no7_ar4_1	112	42	269	283
tln5	35	35	30	56	o7_2	112	42	211	226
ex1252	39	15	43	68	enpro56pb	128	73	192	188
ex1233	48	12	52	121	eniplac	140	24	149	112
nous1	48	2	41	78	enpro48pb	154	92	215	206
nous2	48	2	41	78	tls5	161	136	90	111
tln6	48	48	36	73	tls6	213	177	120	151
ex1243	57	15	75	98	csched2	401	308	138	217
csched1	74	60	17	25	stockcycle	480	432	97	98
m6	84	30	157	170	cecil_13	733	117	686	670
ex1244	86	21	110	160	lop97icx	986	899	87	407
Nonconvex [64]									
Synheat	53	12	65	150	par72	568	56	240	805
SynheatMod	53	12	61	148	c-sched-4-7	233	140	138	217
Multistage	185	18	265	226					
MIQQP [55]									
ibell3a	122	60	104	284	imisc07	260	259	212	957
ibienst1	505	28	576	541	iran8x32	512	256	296	1025
MICP [83] (conic MINLP)									
robust_30_0	123	31	96	96	robust_30_1	123	31	96	96
shortfall_30_0	124	31	97	97	shortfall_30_1	124	31	97	97
classical_40_0	120	40	83	83	classical_40_1	120	40	83	83
MacMINLP, Nonconvex [43]									
space-25	893	750	235	136	space-25-r	818	750	160	136
trimlon7	63	63	42	92	trimlon12	168	168	72	217
Misc									
airConduct	102	80	156	157	nConvPl [70]	948	148	920	749
CLay0203H	90	18	132	211	barton [12]	818	23	987	823
CLay0204H	164	32	234	329					

Table 4. MINLP instances used in the tests.

for each algorithm $s \in S$, the nondecreasing step-wise function

$$\tau_{s\Pi}(\lambda) = \left|\{p \in \Pi : t_{sp} \leq \lambda t_p^{\min}\}\right|$$

is the number of instances of Π solved by s in at most λ times the best time performance on these instances by any algorithm in S. If the performance profile of an algorithm s_1 is higher than that of another algorithm s_2 then s_1 is preferable.

As many instances in our testbed could not be solved within the time limit, we use performance profiles on a similar measure for the lower bound. Further notation is needed; for an instance $p \in \Pi$:

r_{sp}: the root-node lower bound, i.e. the solution of $\mathbf{LP}_0$ after one round of linearization inequalities are generated;
$\bar{z}_{sp}$: the lower bound;
$\hat{z}_{sp}$: the best upper bound found by s;
$\bar{z}_p = \min_{s\in S:\bar{z}_{sp}>-\infty} \bar{z}_{sp}$: the overall smallest finite lower bound;
$\hat{z}_p = \min_{s\in S} \hat{z}_{sp}$: the overall best upper bound;
$r_p = \min_{s\in S:r_{sp}>-\infty} r_{sp}$: the overall smallest finite root-node lower bound.

If all r_{sp} are infinite, we set r_p to $\bar{z}_p$. Instances where all $\bar{z}_{sp}$ are infinite are ignored. If no upper bound is known, we set $\hat{z}_p = \max_{s\in S} \bar{z}_{sp}$. A normalized measure of the

lower bound, $l_{sp} = \frac{\hat{z}_p - \bar{z}_{sp}}{\hat{z}_p - r_p} \in [0, 1]$, describes the *remaining gap* for algorithm s on instance p.

This measure aims at comparing the lower bounds rather than the overall performance of a set of algorithms, thus ignoring the upper bound and the initial lower bound found by each algorithm. Although these affect the performance — for instance, the upper bound helps eliminate part of the solution set through bounds tightening and pruning by bound — they are neglected as the purpose of this paper is to study the effectiveness of branching rules and bounds tightening techniques.

In order to provide a fair comparison on the lower bound, we create performance profiles considering the subset of instances of Π that no algorithm in S could solve before the time limit, defined as $\Gamma(\Pi) = \{p \in \Pi : t_p^{\min} = +\infty\}$. Analogously to performance profiles for CPU time, the nondecreasing step-wise function

$$\mu_{s\Pi}(\lambda) = |\{p \in \Gamma(\Pi) : l_{sp} \leq \lambda\}|$$

is the number of instances of $\Gamma(\Pi)$ for which algorithm s has a remaining gap at most λ. For these performance profiles as well, the higher $\mu_{s\Pi}(\lambda)$, the better. For all classes of instances and of algorithms, we show performance profiles with both $\tau_{s\Pi}(\lambda)$ and $\mu_{s\Pi}(\lambda)$. Tables and graphs are presented for MINLP instances and for continuous NLP instances separately. The best performances (running time or remaining gap) are highlighted in bold in the tables, and so are those within the 10% of the best in order to neglect small differences. The time performance profiles also report, in brackets, the number of instances that at least one algorithm could solve within two hours, i.e., $|\Pi \setminus \Gamma(\Pi)|$, whereas the number of instances that no algorithm could solve in two hours, i.e., $|\Gamma(\Pi)|$, is the highest point in the y-axis of the remaining gap performance profile.

7.1. *Bounds tightening tests*

Tables 11, 12, and 13 report the comparison of the following variants of COUENNE:

no_bt: no bounds tightening;
fbbt: FBBT only;
obbt: FBBT and Optimality-based bounds tightening (OBBT) with parameter $L_{\text{obbt}} = 3$ (see Section 4.1);
abt: FBBT and Aggressive Bounds Tightening (ABT) with parameter $L_{\text{abt}} = 3$ (see Section 4.3);
tuned: a standard setting with FBBT and where both OBBT and ABT are active with $L_{\text{abt}} = 1$ and $L_{\text{obbt}} = 0$.

Notice that FBBT is performed at all sBB nodes in all variants except the first. The purpose of these tests is to show how different bounds tightening techniques help improve the performance of COUENNE. The branching scheme used in all these tests is the **br-plain** described at the end of Section 5.1.

From Tables 11 and 12, none of the variants seems to have an overall performance significantly better or worse than the others in MINLP instances. Although the **no_bt** variant achieves the best performance in some instance, (e.g. *m6*, *ibell3a* and *ibienst1*), it fails to find lower bounds in *Multistage*, *csched1*, and *csched2*, shows numerical problems in *ex1233*, and has poor performance in *c-sched-4-7* and *par72*. The results for instance *csched2* are in some contradiction to those for other problems: the best bound is obtained with FBBT alone, while using ABT is of no use and OBBT degrades the performance substantially, and consequently **tuned**, which uses all of them, does not perform well either. Notice, however, that *no_bt*

fails to provide a lower bound.

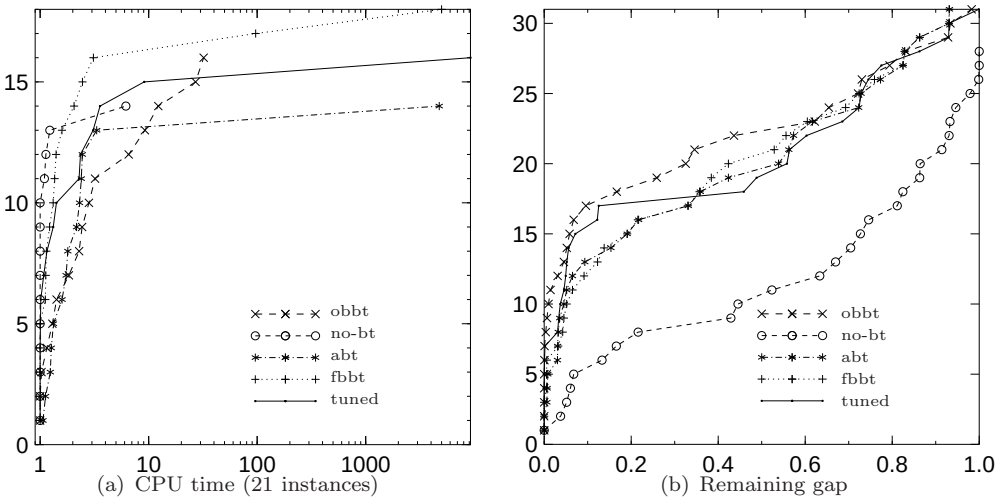
In Table 13, it appears that the **tuned** variant, using a combination of OBBT, ABT, and FBBT, performs better especially for the *Hicks* instances. Again, **no_bt** shows good performance in some of the instances but cannot find a lower bound in the *qp** instances.

The continuous *Hicks* instances suggest that each bounds tightening technique can help improve the lower bound: although the contribution of each technique is hard to quantify, OBBT appears to dramatically improve the lower bound when paired with FBBT, but only when using all three of them (see column **tuned**), all instances are solved before the time limit.

We ran extra tests on the same set of instances to compare the **obbt** variant with one, called **obbt-only**, which never uses FBBT. The results can be summarized as follows:

- three instances were solved by **obbt-only** and not by **obbt** before the time limit; three other instances were solved by **obbt** and not by **obbt-only** before the time limit;
- for three instances (*Multistage*, *csched1*, and *csched2*), **obbt-only** could not find a finite lower bound, while **obbt** finds lower bounds for all instances;
- bounds and solution times were substantially better in **obbt-only** than in **obbt** in nine cases, and viceversa in eleven;
- for all other instances, the results are similar.

Hence, the additional tightening given by FBBT has a limited impact on the solution time, but it helps find a finite lower bound.



(a) CPU time (21 instances) (b) Remaining gap

Figure 5. Performance profiles for bounds tightening on MINLP instances. The number of instances that at least one algorithm could solve within two hours, i.e., $|\Pi \setminus \Gamma(\Pi)|$, is given in brackets in (a); the number of instances that no algorithm solved in two hours, i.e., $|\Gamma(\Pi)|$, is the highest point on the y-axis in (b).

Figure 5a shows that **no-bt** solves most quickly some of the instances, but not as many as **fbbt**, **tuned**, and **obbt** within the time limit; all variants for "easy" instances (those solved in less than two hours) solve about the same number of instances. In the performance profile for remaining gap (Fig. 5b), however, the losing strategy is **no-bt**, which is consistently below all other graph and has a tail of about 5 instances for which virtually no gap is covered. **Obbt** and **tuned** obtain a gap below 5% for 15 instances out of 30, but then their performances are similar to that of **abt** and **fbbt**.

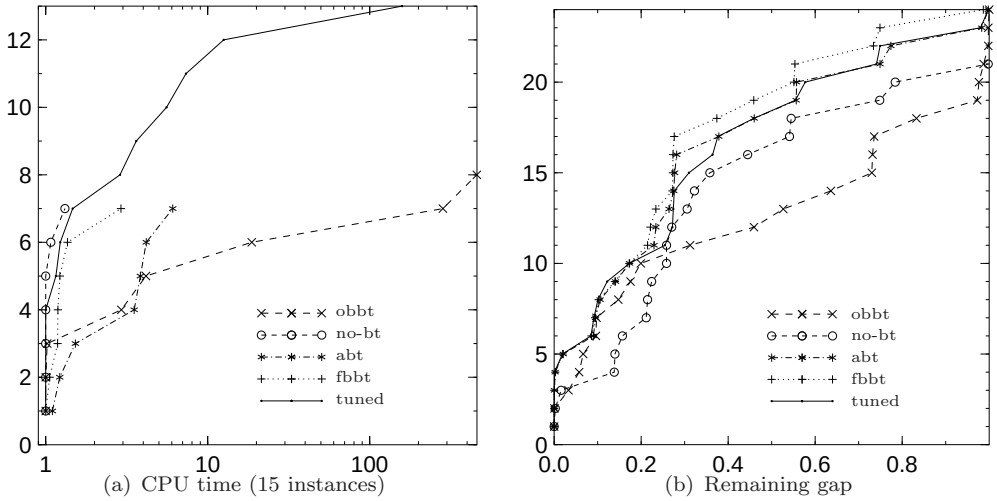


Figure 6. Performance profiles for bounds tightening on continuous NLP instances.

For continuous nonlinear instances (see Figure 6), the best strategy is again **tuned** for instances solved within the time limit, although it takes ten and 100 times longer than **no-bt** and **fbbt** to solve two instances: *spar-030-060-2* and *primal4*, respectively. It dominates **obbt** and **abt**, which have the worst time performance. For the instances that are not solved in two hours, **no-bt** turns out to be a poor choice as it is dominated by most strategies, but **obbt** also performs poorly, with a remaining gap of more than 50% for almost half of the instances. There is no strictly dominating variant, although **fbbt** is better or comparable to the others, thus suggesting that, at least for continuous NLP instances, ABT and OBBT should be limited, for instance by running them only at the root node.

Table 5 briefly reports on the number of sBB nodes and the maximum depth of the sBB tree obtained by bounds tightening variants. These results are only included for the 21 instances that at least two of the variants in the comparison could solve before the time limit. For each variant, we report the number of instances solved (*#solved*), the number of instances on which the variant obtained the best time (*#best time*), the number of instances with lowest nodes count (*#best nodes*), and the number of instances with minimum sBB tree depth (*#best depth*) — these numbers are taken as "best" if they are within 10% of the minimum among all variants. It is apparent that **tuned** gets the best node and depth performance far more often than the other variants. We do not show the detailed results on each instance; however, we point out that the instances where **no_bt** obtains the best number of nodes are mostly the box-QP *spar-** instances.

Variant	#solved	#best time	#best nodes	#best depth
tuned	19	12	14	16
fbbt	16	2	4	6
abt	14	1	3	6
no-bt	16	10	7	10
obbt	16	2	8	9

Table 5. Comparison of the performance on number of sBB nodes and maximum depth.

7.2. *Branching-techniques tests*

The purpose of this set of tests is to assess the quality of the branching techniques implemented in COUENNE. It also compares Violation Transfer with two methods that proved successful in MILP, namely strong and reliability branching. For all tests below, the bounds tighening parameters are as in the **tuned** variant described above.

Branching-variable selection

We performed an initial test on a small set of instances with several branching options, to narrow the choice to the most promising ones. Many of them are variants of the reliability branching scheme discussed in Section 5.3, differing in the choice of the multipliers δ^{k-} and δ^{k+} for the pseudocosts. Recall that these multipliers are used in two steps: (i) when estimating the change in the objective function and (ii) when updating the pseudocosts. We tested options *rb-inf*, *rb-int-br*, *rb-int-lp-rev*, *rb-proj*, *rb-vt* (see Section 5.4), as well as:

str-br: strong branching at all nodes; although computationally expensive, this allows for the comparison of the overall quality of pseudocost multipliers — i.e. of their ability to estimate the per-unit of change in the branching variable;

rb-proj+lpdist: *rb-proj* when estimating the change in the lower bound, and *rb-lpdist* for updating the pseudocosts;

rb-vt+lpdist: *rb-vt* when estimating the change in the lower bound, and *rb-lpdist* for updating the pseudocosts;

pure-vt: the Violation Transfer algorithm;

br-plain: the infeasibility-based branching, where the variable x_i with largest $\Omega_i(\bar{x}^k)$ is selected for branching (see Section 5.1).

These initial tests are summarized in Table 6. For each strategy we report the number of instances that were solved before the two-hour time limit (*#<2h*), the number of times when its solution time was minimum[1] (*#best*), the geometric average of the running time (t_{avg}), the number of times when the remaining gap was minimum[1] (*#best_gap*) and the geometric average of the remaining gap (g_{avg}).

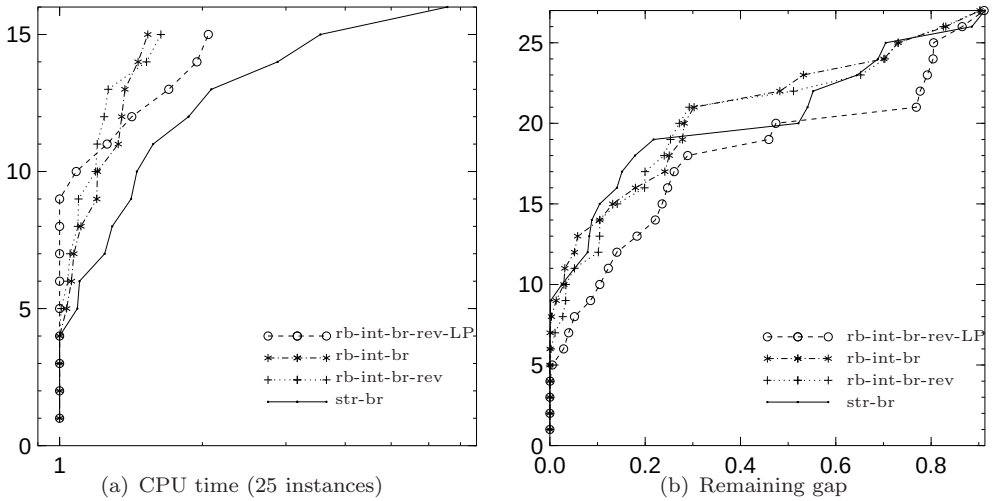
The average of both the running time and of the remaining gap are computed on different subsets for each variant (those instances that are solved in less than two hours and those that are not solved, respectively), hence columns t_{avg} and g_{avg} give somewhat inconsistent information. However, for the strategies that we have later chosen for the detailed tests, these averages are made on sets of instances that share a fairly large common subset, of 13 and 28 instances respectively.

Name	#<2h	#best	t_{avg}	#best_gap	g_{avg}
br-plain	**19**	**8**	545.4	**8**	0.1968
pure-vt	**22**	**7**	435.6	4	0.1466
str-br	**23**	6	400.6	2	0.1012
rb-proj	13	6	764.5	0	0.1400
rb-proj+lpdist	13	3	558.0	2	0.1471
rb-inf	**20**	**8**	379.8	3	0.1344
rb-int-br	**24**	6	384.1	**11**	0.1233
rb-int-lp-rev	16	5	716.6	2	0.1831
rb-vt	14	5	654.0	1	0.1739
rb-vt+lpdist	16	3	413.5	6	0.1638

Table 6. Initial comparison of branching techniques.

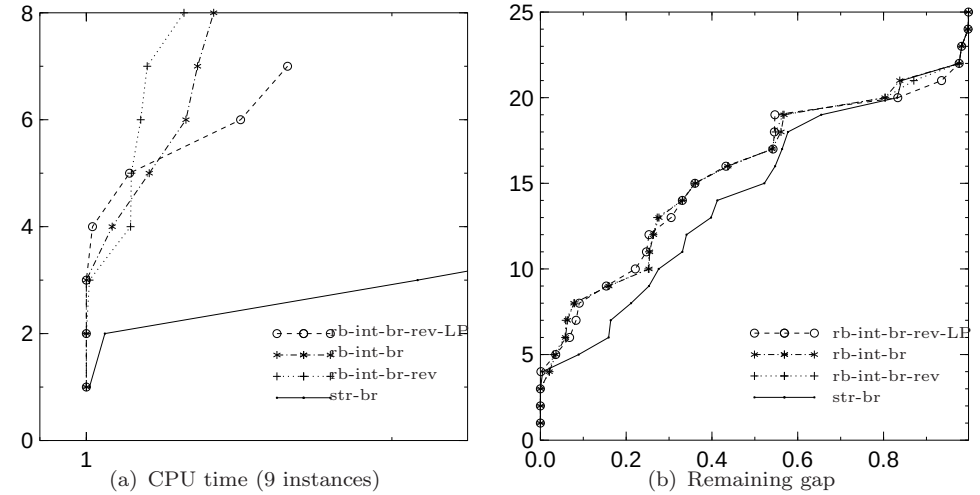
[1]Running times and remaining gaps are taken as "equal" if they do not differ by more than 10%.

From Table 6 it appears that the most promising branching techniques are **br-plain**, **pure-vt**, **str-br**, **rb-inf**, and **rb-int-br**. We also tested two variants of **rb-int-br**: **rb-int-br-rev** and **rb-int-br-rev-LP**, which is similar to **rb-int-br-rev** except that the update of the pseudocost multiplier is done with **rb-lpdist**. These three variants are compared with one another to show that they perform similarly, and we include only one of them, **rb-int-br-rev**, in the comparison with the other branching techniques. Performance profiles for these three variants are given in Figures 7 and 8. On MINLP instances, **rb-int-br** and **rb-int-br-rev** have good performances both in terms of running time and remaining gap. While **str-br** performs worse on those instances that are solved within the time limit (Fig. 7a), it does a good job on those that are not solved by any of these branching techniques, where **rb-int-br-rev-LP** is dominated (Fig. 7b). Results on continuous NLP instances favor all of the **rb-int-*** strategies against **str-br** on all instances.



(a) CPU time (25 instances) (b) Remaining gap

Figure 7. Performance profiles for **rb-int-*** branching techniques on MINLP instances.



(a) CPU time (9 instances) (b) Remaining gap

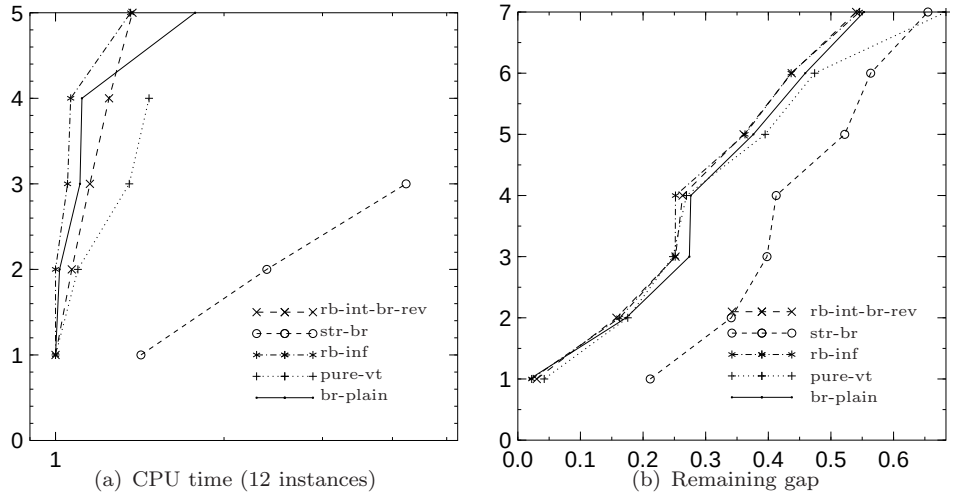
Figure 8. Performance profiles for **rb-int-*** branching techniques on continuous NLP instances.

The tests on branching strategies **br-plain**, **pure-vt**, **rb-inf**, **str-br**, and **rb-int-br-rev** are shown in Tables 14 and 15 for MINLP instances, and in Table 16

for continuous NLP instances. For all variants tested, all five *Hicks* instances were solved at the root node by the standard bounds tightening setting, and are thus excluded from these tables.

On *spar-** instances, **str-br** instances performs consistently worse, while all other variants give similar results. These are box QP problems: $\max\{cx + x^{\mathrm{T}}Qx : x \in [0,1]^n\}$. The similarity of the performances of simple strategies **br-plain** and of more elaborate ones such as **pure-vt** and **rb-*** is highlighted in Figure 9; **pure-vt** has a similar performance also in terms of sBB nodes (not displayed in the tables). This suggests that the infeasibility Ω_i^N used in **br-plain** is as reliable for unconstrained problems as more sophisticated branching strategies such as **pure-vt** or **rb-***. A plausible explanation for the poor performance of **str-br** is that it spends most of the CPU time in solving the LP problems necessary to select the branching variable: between 90% and 95% in all instances that are not solved within the time limit, and 60%, 96%, and 34% in the remaining three.

Variants **br-plain** and **pure-vt** perform better than strong and reliability branching variants on other quadratic instances. This holds for instances *dual1*, *dual4*, and *cvxqp3_s*, whose size does not allow **str-br** or **rb-*** variants to develop a sBB tree larger than a few nodes, while **br-plain** and **pure-vt** create far more sBB nodes and thus improve the lower bound.



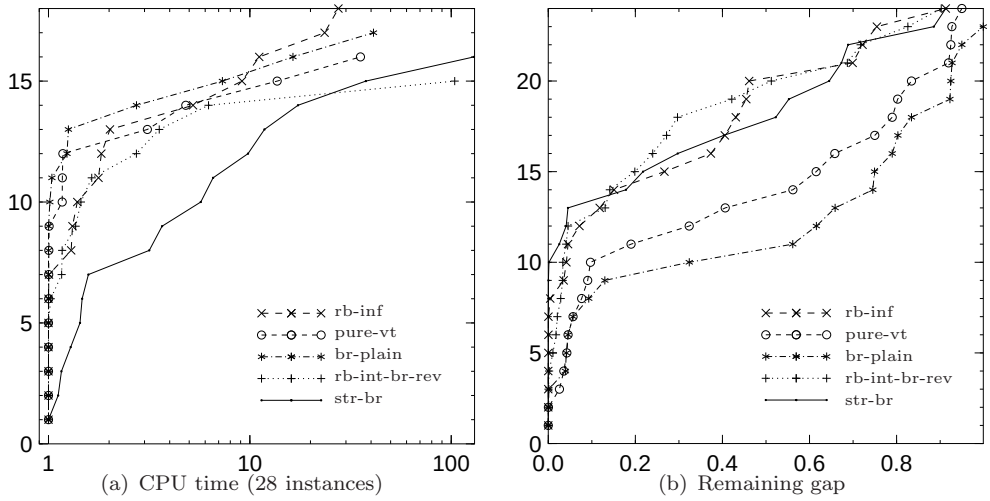
(a) CPU time (12 instances) (b) Remaining gap

Figure 9. Performance profiles for comparing all branching rules on the *spar-** instances.

The comparison between branching rules on MINLP instances does not show a clear winner. The difference is highly dependent on the single instances, but the tables suggest that **br-plain** and **pure-vt** often have similar performances to each other, while **rb-inf** and **rb-int-br-rev** and, in some cases, **str-br**, seem to perform equally well — this is apparent in MINLP instances *nvs19*, *ex1233*, *nvs23*, *csched2*, *cecil_13*, *lop97icx* (see Table 14), *trimlon12*, *foulds3*, *barton*, *space-25* (see Table 15) and in NLP instances *ex5_3_3*, *dual4*, and *values* (Table 16).

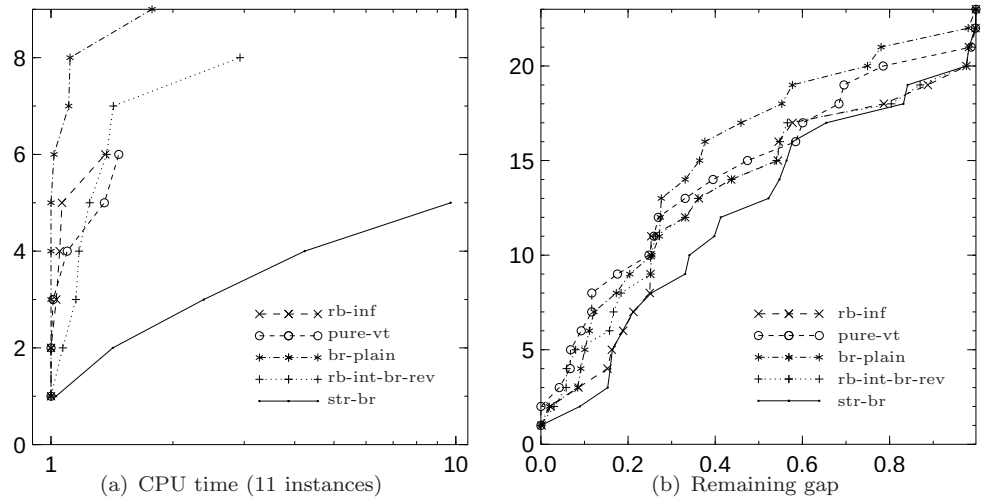
The performance profiles for MINLP instances (Figure 10) provide a clearer perspective: for relatively easy instances, the time profile shows that **pure-vt** and **br-plain** are good branching strategies as opposed to **str-br** and **rb-int-br-rev**. With more difficult instances, **pure-vt** and **br-plain** perform much worse and are dominated by the strong and reliability branching variants. This observation holds for easy NLP instances (Figure 11a) but not for the more difficult ones: the gap performance profile (Figure 11b) suggests that **br-plain** and **pure-vt** give a better bound, although not by a large amount, while **str-br** is dominated by all other

strategies.



(a) CPU time (28 instances) (b) Remaining gap

Figure 10. Performance profiles for reliability branching, VT, and *infeasibility*-based branching on MINLP instances.



(a) CPU time (11 instances) (b) Remaining gap

Figure 11. Performance profiles for reliability branching, VT and *infeasibility*-based branching on continuous NLP instances.

Branching-point selection

We briefly report here on our tests on the branching point selection strategies described in Section 5.5. As with variable selection, there is no single strategy that obtains better results, but a combination of them appears to be the best choice.

We tested the LP-based strategies on general MINLP instances and the expression-based strategies (*balanced* and *min-area*) on a set of instances that use a comprehensive subset of operators of Θ. We used a set of parametric instances from the *Cops* collection [25], comprising most of the operators that are handled by COUENNE and with parameters that allow to create instances of any size.

We performed extensive tests on both LP- and expression-based strategies, which we do not report here, and summarize the results below:

- the LP point $\bar{x}^k$ should always be taken into account for the two resulting nodes to be of similar difficulty; this also ensures that $\bar{x}^k$ is cut;
- in further support to the previous point, the expression-based strategies do not have an influence on the performance of COUENNE, even when compared on a per-operator basis, suggesting that minimizing or balancing the size of the resulting linearizations is not as important as improving the bounds;
- x_i^b should not be close to the bounds, but it should be kept at a minimum distance from them;
- some operators (product, quotient) benefit from having operands that are either nonnegative or nonpositive, thus if the branching point is relatively close to zero it should be set to zero. For instance, if there is an auxiliary variable $x_3 = x_1 x_2$ with $x_1 \in [0, +\infty)$, $x_2 \in [-1, 1]$ and x_2 has been selected as branching variable with $x_2^b = \epsilon$ with a small ϵ, we restrict the sign of x_3 in both resulting nodes by setting $x_2^b = 0$, so that $x_3 \leq 0$ in one node and $x_3 \geq 0$ in the other; this is useful in situations such as that pointed out in footnote 1 at page 7.

Therefore, the default strategy implemented in COUENNE, which is used in testing both the bounds tightening techniques and the branching variable selection techniques, sets x_i^b as in (2), or to zero if sufficiently close to zero with respect to its bounds. This strategy is similar to the one described by Tawarmalani and Sahinidis [79, page 194].

Number of sBB nodes and maximum depth

We discuss here the performance of the branching strategies in terms of number of nodes and of maximum depth of the sBB tree. We only provide these results on the 33 instances that at least two of the variants in the comparison could solve before the time limit. Table 17 reports the running time, number of sBB nodes, and maximum depth of the sBB tree for variants **str-br**, **pure-vt**, **rb-int-br-rev**, **rb-inf**, and **br-plain**. Empty entries in the table correspond to instances that could not be solved in two hours.

Table 7 below summarizes the results on these 33 instances, providing, for each variant, the number of instances solved (*#solved*), the number of instances the variant obtained the best time (*#best time*), number of nodes (*#best nodes*), and maximum sBB tree depth (*#best depth*) — again, the number of nodes and the maximum depth are taken as "best" if they are within 10% of the best.

Variant	#solved	#best time	#best nodes	#best depth
str-br	25	1	17	11
pure-vt	26	13	4	20
br-plain	23	14	7	15
rb-inf	23	11	4	8
rb-int-br-rev	25	9	5	8

Table 7. Comparison of the performance on number of sBB nodes and maximum depth.

As expected, **str-br** is the best strategy 17 times out of 33 in terms of number of nodes created, far above all other strategies. However, it runs in shortest time only once, much worse than **pure-vt** and **br-plain**. These two strategies, although not as elaborate as strong or reliability branching, perform relatively well in terms of nodes and depth. They obtain the best sBB tree depth 20 and 15 times, respectively, and hence are better than the others in keeping the sBB tree balanced. There is no substantial difference between **rb-int-br-rev**, **rb-int-br**, and **rb-int-br-rev-LP**, not shown in the tables.

7.3. *Comparison with* BARON

BARON [71] is a state-of-the-art solver for MINLP problems. It implements several bounds tightening and linearization techniques, and the Violation Transfer, described in Section 5.2, as branching technique. It is therefore the ideal candidate for a comparison with COUENNE.

The comparison is conducted on all the instances of the testbed. From the result of the computational tests shown above, we have chosen the variant of COUENNE that has reliability branching turned on with **rb-int-br-rev** as pseudocosts multiplier strategy; FBBT, ABT, and OBBT are turned on with $L_{\text{abt}} = 1$ and $L_{\text{obbt}} = 0$. Version 7.5 of BARON has been used, with options `workfactor=2000` and `epsr=1e-09`. Both algorithms were run on a machine equipped with a 3.2GHz processor, 2.5GB of RAM memory, `gcc` version 4.1.2, and Linux kernel 2.6.23. While COUENNE reads input `.nl` files generated by AMPL [33], BARON has been run from GAMS [19]. Instances from **minlplib** and **globallib** were originally only in GAMS format, and their AMPL version has been created with the *convert* feature in GAMS. Some other instances are available in AMPL format only, and their GAMS version has been created with a conversion program from the COCONUT software package [73] or, when COCONUT returns an error, with a conversion routine available in COUENNE.

The performances of the two algorithms are reported in Tables 8, 9, and 10 and are graphically summarized in Figure 12, which is divided in four parts: Fig. 12(a,b) compares the CPU times of all instances that BARON or COUENNE can solve before the time limit, while Fig. 12(c,d) depicts a comparison between the *remaining gaps* of the two algorithms — see definition in the first part of Section 7.

The graphs in Figures 12(a,c) contain one point for each instance: BARON is on the x-axis and COUENNE on the y-axis, therefore a point in the lower right part favors COUENNE. Figures 12(b,d) contain time and gap performance profiles.

BARON uses *probing* extensively. On the instances that required more than ten seconds of CPU time, a percentage between 41% and 99% of the CPU time was spent in probing, with a geometric average of 77%. This results in BARON generating fewer sBB nodes than COUENNE in all the instances. We do not report the sBB nodes of the two algorithms in detail, but the geometric mean of their ratio is 1:5; this average is computed on the 64 instances that required at least two nodes for both algorithms.

The tables and graphs suggest that the performances of BARON and COUENNE are comparable, although it is clear that BARON is on average better both in CPU time and in lower bound. In the large, nonconvex instances *Multistage*, *Synheat*, *SynheatMod*, *c-sched-4-7*, and *par72*, BARON provides both better lower and upper bounds, while COUENNE performs better with *ibienst1*, *imisc07*, *ibell3a*, and *iran8x32* (the last two are convex).

COUENNE dominates in the *spar-** box-QP instances. Let us take as example instance *spar030-060-1*. BARON spends 96% of the allotted time in probing (which is done at all sBB nodes), while COUENNE spends less than 10% of the time in separation of linearization cuts (adding in total 4,229,270 of them) and bounds tightening. The difference here seems to be in the number of nodes, 22,567 for BARON and 100,726 for COUENNE, in accordance with the average ratio. BARON seems to have numerical problems in the continuous *Hicks* instances, as it cuts all optimal solutions and claims optimality of a local minimum instead.

COUENNE seems to perform slightly better than BARON in continuous NLPs than in general MINLPs. There are various possible explanations for this; for example, COUENNE has only a simple heuristic for finding upper bounds, which fails to obtain an integer feasible solution for some of the instances. A more sophisticated

Name	BARON time (lb)	BARON ub	COUENNE time (lb)	COUENNE ub
GlobalLib (pure NLPs) [51]				
ex5_2_5	**(-5054.59)**	-3500	(-6831.8)	-3500
ex5_3_3	**(2.173)**	3.234	(1.869)	3.325
qp1	(-0.3047)	8.093e-4	**(-0.0861)**	8.093e-4
qp2	(-0.3056)	8.093e-4	**(-0.0817)**	8.093e-4
qp3	**(-0.0938)**	8.093e-4	(-0.3017)	8.093e-4
foulds3	(-69.804)	-8	(-71.311)	-8
QCQP [56]				
cvxqp1_s	(9622.57)	**11590.7**	**(10775.1)**	–
cvxqp2_s	(6919.6)	**8120.94**	**(7291.6)**	–
cvxqp3_s	**180.14**	**11943.4**	(11943.1)	–
dual1	**(-178.1)**	0.0350	(-214.431)	0.0350
dual2	**(-184.408)**	0.0337	(-221.265)	0.0337
dual4	**(-202.002)**	0.746	(-304.143)	0.746
dualc8	(18306.4)	18309.3	(18309.1)	18309.2
gouldqp2	(-0.185)	1.842e-4	(-0.166)	1.842e-4
primal4	(-0.779)	0	**1395.74**	**-0.746**
qadlittl	**200.39**	480319	(480319)	480995
qbrandy	**8.68**	28375.1	58.88	28375.1
qetamacr	(61835.2)	86760.4	(63739.6)	86760.4
qshare1b	**1042.44**	**720078**	(720005)	–
stcqp1	(148327)	157759	(148329)	155144
values	(-12.914)	-1.396	(-13.621)	-1.396
BoxQPs [82]				
spar030-060-1	(-830.163)	-706	**2028.56**	-706
spar030-060-2	**3.55**	-1377.17	27.00	-1377.17
spar040-050-1	(-1403.03)	-1154.5	**(-1195.63)**	-1154.5
spar040-050-2	(-1635.81)	-1430.98	**3581.21**	-1430.98
spar040-060-1	(-2008.91)	-1322.67	**(-1729.59)**	-1311.06
spar050-040-1	(-1900.34)	-1411	**(-1608.66)**	-1411
spar050-050-1	(-2685.75)	-1198.41	**(-2220.19)**	-1193
spar060-020-1	3622.04	-1212	**522.99**	-1212
spar060-020-2	54.94	-1925.5	54.67	-1925.5
spar070-025-1	(-3168.62)	-2538.91	**(-2865.74)**	-2538.91
spar080-025-1	(-4173.78)	-3157	**(-3775.16)**	-3157
spar090-025-1	(-5466.5)	-3372.5	**(-4882.19)**	-3361.5
Hicks [30]				
Hicks_5	**21.89**	227.542	31.82	227.257
Hicks_10	43.81	229.13	**31.68**	227.257
Hicks_20	334.37	229.733	**129.42**	227.257
Hicks_50	3967.71	227.406	**820.14**	227.257
Hicks_100	(0)	234.158	**2615.85**	227.257

Table 8. Comparison between BARON and COUENNE on continuous NLP problems.

heuristic would help eliminate portions of the solution space in two ways: bounds tightening and pruning by bound. Also, in the above experiments COUENNE did not include MILP cutting planes available in COIN-OR (e.g. lift-and-project cuts, mixed-integer Gomory cuts, etc.); some preliminary tests have shown that those cuts improve the performance for some of these instances: *Synheat*, *SynheatMod*, *ex1233*, and *ex1244*.

8. Concluding remarks

This paper presents COUENNE, an Open-Source solver for MINLP problems. It is meant to provide the Optimization Community with a flexible tool of immediate use

Name	BARON time (lb)	BARON ub	COUENNE time (lb)	COUENNE ub
Nonconvex [64]				
Multistage	**70.88**	**-7581.03**	(-36943.6)	–
Synheat	**4680.42**	**154997**	(131034.7)	160435.5
SynheatMod	**3815.93**	**154997**	(81950.5)	161010
c-sched-4-7	**(-193716)**	**-133402**	(-257752)	–
par72	**(-1930.9)**	**2456.07**	(-8057.684)	–
MICP [83] (conic MINLP)				
robust_30_0	57.21	-0.0455	**2.29**	-0.0455
robust_30_1	**189.92**	-0.0477	5134.01	-0.0477
shortfall_30_0	**27.09**	-1.080	1427.64	-1.080
shortfall_30_1	**215.28**	-1.085	(-1.088)	-1.078
classical_40_0	**216.98**	-0.0815	1797.52	-0.0815
classical_40_1	**20.97**	-0.0847	200.90	-0.0847
MacMINLP, Nonconvex [43]				
space-25	**(156.669)**	**493.914**	(93.9194)	–
space-25-r	**(160.45)**	**786.343**	(68.7403)	–
trimlon12	(26.710)	**583.8**	(29.342)	–
trimlon7	(13.174)	15.1	**3840.18**	15
Misc				
CLay0203H	0.69	NUM	(40377.4)	–
CLay0204H	**1.37**	**7885**	(4650)	10510
nConvPl	(8944.50)	**7529.30**	**(-8793.307)**	-4362.174
airConduct	(24.5113)	25.6905	(24.5108)	26.8327
barton	(-103.315)	**-81.8659**	(-103.007)	–
MIQQP [55]				
ibell3a	(-3.370e+09)	2.966e+06	**896.17**	**878785**
ibienst1	(-2.424e+09)	48.7377	**5099.96**	48.7377
imisc07	(0)	–	**(2447.98)**	**2815.82**
iran8x32	141.21	NUM	**4972.72**	**5255.45**

Table 9. Comparison between BARON and COUENNE on MINLP problems.

that can be enhanced or suited to specific classes of nonconvex MINLP problems.

The main components of COUENNE, linearization, branching and bounds tightening, are well-known to the MINLP community, however they lacked, up to now, a computational basis to develop new MINLP solution approaches that specialize the standard Branch-and-Bound procedure.

We have proposed a computational study that focuses on the effectiveness of bounds tightening and branching rules. The former prove, on average, to be important for speeding up the convergence of the Branch-and-Bound. For branching rules, we have adapted a well known approach of MILP, reliability branching, that has been thoroughly studied for integer variables, to a more general class of problems that can require branching on continuous variables. We have shown its utility in the MINLP framework, although simpler techniques such as Violation Transfer sometimes give better results.

COUENNE is also a fair competitor of state-of-the-art solvers such as BARON, although it would benefit from a good heuristic for finding feasible solutions. A possibility is to use a convex MINLP solver such as Bonmin [17] or a heuristic developed for that purpose, the Enhanced Feasibility Pump [18]. Current developments comprise a class of cuts handling disjunctions arising from nonconvex MINLP problems, SOS branching [13], and different operators for handling quadratic expressions and other functions.

Name	BARON time (lb)	BARON ub	COUENNE time (lb)	COUENNE ub
MinlpLib [20]				
cecil_13	(-115895)	**-115657**	(-116057)	–
csched1	(-30639.3)	-30639.3	**141.42**	-30639.3
csched2	(-247203)	**-148733**	(-262709)	–
du-opt	**72.11**	3.556	248.16	3.556
du-opt5	**28.34**	**8.073**	(7.965)	70.088
eniplac	(-140736)	-131855	**1582.04**	-132117
enpro48pb	**20.14**	187277	168.11	187277
enpro56pb	66.57	263428	56.09	263428
ex1233	**169.79**	155011	(83587.12)	155522.5
ex1243	1.33	83402.5	6.44	83402.5
ex1244	**25.01**	82042.9	(72342.1)	84035.6
ex1252	0.23	143555	102.03	128894
fo7	**(12.643)**	**22.392**	(0)	33.5943
lop97icx	(2722.78)	**4373.74**	**(3086.75)**	–
m6	**283.19**	**82.256**	(60.485)	123.91
no7_ar2_1	**4719.64**	107.815	(94.325)	131.985
no7_ar3_1	(71.8686)	111.247	**(86.873)**	119.587
no7_ar4_1	(72.5545)	100.419	(68.601)	109.943
nous1	**169.23**	1.567	(0.745)	1.567
nous2	1.22	0.625	42.41	0.625
nvs19	32.46	-1098.4	40.61	-1098.4
nvs23	117.7	-1125.2	182.54	-1125.2
o7_2	(66.32)	128.054	(63.7769)	130.553
stockcycle	**668.23**	**119949**	(119872)	145727
tln5	155.59	10.3	76.91	10.3
tln6	(14.6675)	15.3	**7112.72**	15.3
tls5	(5.634)	**11.2**	(3.633)	–
tls6	(5.743)	**15.6**	(5.010)	–

Table 10. Comparison between BARON and COUENNE on MINLP problems. "NUM" indicates numerical problems.

Acknowledgments

This research was conducted under an Open Collaboration Research Agreement between the IBM Corporation and Carnegie Mellon University, a grant from the IBM Corporation, and NSF Grant OCI07500826.

References

[1] K. Abhishek, S. Leyffer, and J.T. Linderoth. FILMINT: An outer-approximation-based solver for nonlinear mixed integer programs. Preprint ANL/MCS-P1374-0906, 2006.
[2] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *OR Letters*, 33(1):42–54, 2005.
[3] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. A global optimization method, αBB, for general twice-differentiable constrained NLPs: II. Implementation and computational results. *Computers & Chemical Engineering*, 22(9):1159–1179, 1998.
[4] C.S. Adjiman, I.P. Androulakis, and C.A. Floudas. Global optimization of MINLP problems in process synthesis and design. *Computers & Chemical Engineering*, 21:S445–S450, 1997.
[5] C.S. Adjiman, I.P. Androulakis, C.D. Maranas, and C.A. Floudas. A global optimization method, αBB, for process design. *Computers & Chemical Engineering*, 20:S419–S424, 1996.
[6] C.S. Adjiman, S. Dallwig, C.A. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs: I. Theoretical advances. *Computers & Chemical Engineering*, 22(9):1137–1158, 1998.
[7] C.S. Adjiman and C.A. Floudas. Rigorous convex underestimators for general twice-differentiable problems. *Journal of Global Optimization*, 9(1):23–40, July 1996.
[8] C.S. Adjiman, C.A. Schweiger, and C.A. Floudas. Mixed-integer nonlinear optimization in process synthesis. In D.-Z. Du and P.M. (Eds.) Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1, pages 1–76. Kluwer Academic Publishers, Dordrecht, 1998.
[9] F.A. Al-Khayyal and J.E. Falk. Jointly constrained biconvex programming. *Math. Oper. Res.*, 8:273–286, 1983.
[10] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB: A global optimization method for general

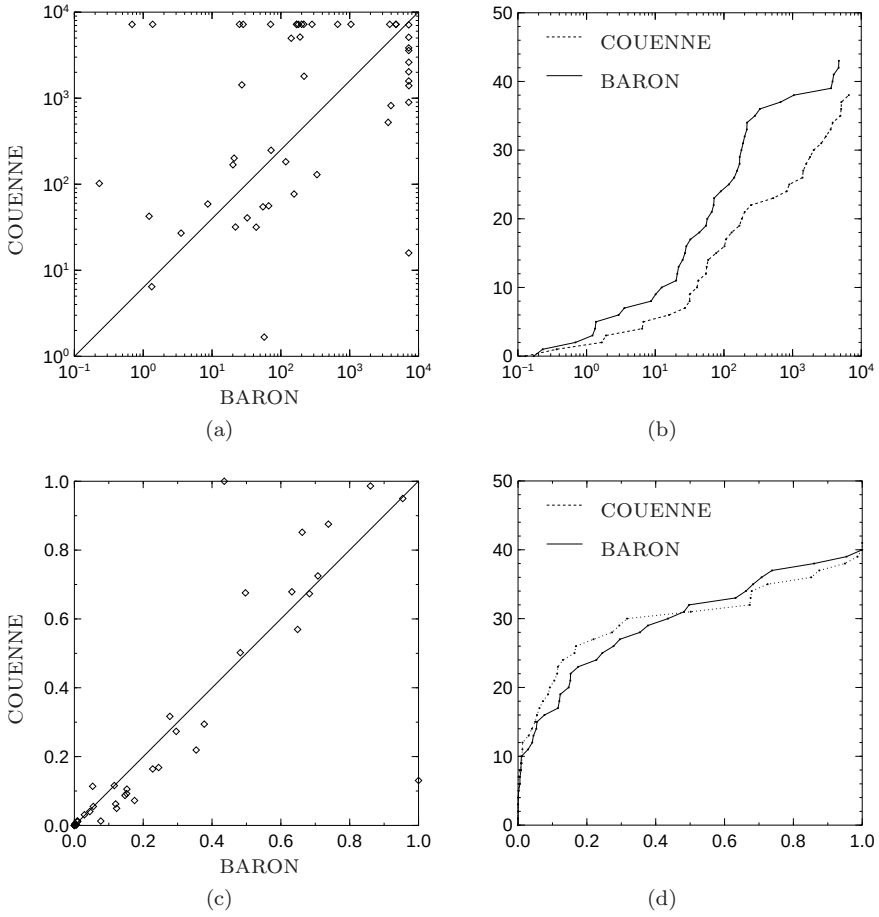


Figure 12. Comparison between BARON and COUENNE.

constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, December 1995.

[11] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem, A Computational Study*. Princeton, 2006.

[12] M. Barton and A. Selot. A production allocation framework for natural gas production systems. In V. Plesu and P.S. Agachi, editors, *17th European Symposium on Computer Aided Process Engineering - ESCAPE17*, 2007.

[13] E. M. L. Beale and J. J. H. Forrest. Global optimization using special ordered sets. *Mathematical Programming*, 10(1):52–69, 1976.

[14] P. Belotti. COUENNE, an Open-Source solver for Mixed-Integer non-convex problems. In preparation.

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

[16] L.T. Biegler, I.E. Grossmann, and A.W. Westerberg. *Systematic Methods of Chemical Process Design*. Prentice Hall, Upper Saddle River (NJ), 1997.

[17] P. Bonami, L. Biegler, A. Conn, G. Cornuéjols, I. Grossmann, C. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5:186–204, 2008.

[18] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A Feasibility Pump for Mixed Integer Nonlinear Programs. Technical Report RC23862, IBM, 2008. To appear on Mathematical Programming.

[19] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide, Release 2.25*. The Scientific Press, 1992. `http://citeseer.ist.psu.edu/brooke92gams.html`.

[20] 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(1), 2003. `http://www.gamsworld.org/minlp/minlplib/minlpstat.htm`.

[21] E. Carrizosa, P. Hansen, and F. Messine. Improving interval analysis bounds by translations. *Journal of Global Optimization*, 29(2):157–172, 2004.

[22] COIN-OR project. See `http://www.coin-or.org`.

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

[24] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[25] E.D. Dolan, J.J. Moré, and T.S. Munson. Benchmarking optimization software with COPS 3.0. Technical Report ANL/MCS-273, Mathematics and Computer Science Division, Argonne National Laboratory, 2004. `http://www.mcs.anl.gov/~more/cops/cgilog.cgi?+/cops/cops3.pdf`.

[26] T.G.W. Epperly and E.N. Pistikopoulos. A reduced space branch and bound algorithm for global optimization. *Journal of Global Optimization*, 11:287:311, 1997.
[27] J.E. Falk and R.M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15:550–569, 1969.
[28] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.
[29] R. Fletcher and S. Leyffer. Numerical experience with lower bounds for MIQP branch-and-bound. *SIAM Journal of Optimization*, 8(2):604–616, 1998.
[30] A. Flores-Tlacuahuac and L. T. Biegler. A robust and efficient mixed-integer non-linear dynamic optimization approach for simultaneous design and control. *Computers and Chemical Engineering*, 31:588–600, 2007.
[31] C.A. Floudas. Global optimization in design and control of chemical process systems. *Journal of Process Control*, 10:125–134, 2001.
[32] L.R. Foulds, D. Haugland, and K.Jörnsten. A bilinear approach to the pooling problem. *Optimization*, 24:165–180, 1992.
[33] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: a modeling language for mathematical programming*. Boyd and Fraser Publishing Company, 1993.
[34] A. Fügenschuh and L. Schewe. Solving a nonlinear mixed-integer sheet metal design problem with linear approximations, work in progress.
[35] GAMS Development Corp. SBB – `http://www.gams.com/dd/docs/solvers/sbb.pdf`.
[36] I. Grossmann. Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, 3(3):227–252, 2002.
[37] E. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, Inc., New York, 1992.
[38] R. Horst. A general-class of branch-and-bound methods in global optimization with some new approaches for concave minimization. *Journal of Optimization Theory and Applications*, 51(2):271–291, November 1986.
[39] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer Verlag, Berlin, 1996.
[40] M. Jünger and D. Naddef, editors. *Computational Combinatorial Optimization*, volume 2241 of *Lecture Notes in Computer Science*. Springer, Berlin/Heidelberg, 2001.
[41] J. Kallrath. Solving planning and design problems in the process industry using mixed integer and global optimization. *Annals of Operations Research*, 140(1):339–373, 2005.
[42] P. Kesavan and P.I. Barton. Generalized branch-and-cut framework for mixed-integer nonlinear optimization problems. *Computers & Chemical Engineering*, 24:1361–1366, 2000.
[43] S. Leyffer. `MacMINLP`: AMPL collection of MINLPs. `http://www-unix.mcs.anl.gov/~leyffer/MacMINLP`.
[44] S. Leyffer. User manual for MINLP_BB. Technical report, University of Dundee, UK, March 1999.
[45] L. Liberti. Writing global optimization software. In L. Liberti and N. Maculan, editors, *Global Optimization: from Theory to Implementation*, pages 211–262. Springer, Berlin, 2006.
[46] L. Liberti. Reformulation techniques in mathematical programming, November 2007. Thèse d'Habilitation à Diriger des Recherches.
[47] L. Liberti. Reformulations in mathematical programming: Definitions. In R. Aringhieri, R. Cordone, and G. Righini, editors, *Proceedings of the 7th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, Crema, 2008. Università degli Studi di Milano.
[48] L. Liberti, C. Lavor, and N. Maculan. A branch-and-prune algorithm for the molecular distance geometry problem. *International Transactions in Operational Research*, 15:1–17, 2008.
[49] L. Liberti, C. Lavor, M.A. Chaer Nascimento, and N. Maculan. Reformulation in mathematical programming: an application to quantum chemistry. *Discrete Applied Mathematics*, accepted for publication.
[50] L. Liberti and C.C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25(2):157–168, 2003.
[51] GamsWorld Global Optimization library. `http://www.gamsworld.org/global/globallib/globalstat.htm`.
[52] Lindo Systems. Lindo solver suite. See `http://www.gams.com/solvers/lindoglobal.pdf`.
[53] G.P. McCormick. *Nonlinear programming: theory, algorithms and applications*. John Wiley & sons, 1983.
[54] F. Messine. Deterministic global optimization using interval constraint propagation techniques. *RAIRO-RO*, 38(4):277–294, 2004.
[55] H. Mittelmann. A collection of mixed integer quadratically constrained quadratic programs. `http://plato.asu.edu/ftp/ampl_files/miqp_ampl`.
[56] H. Mittelmann. A collection of quadratically constrained quadratic programs. `http://plato.asu.edu/ftp/ampl_files/qp_ampl`.
[57] R.E. Moore. *Methods and Applications of Interval Analysis*. Siam, Philadelphia, 1979.
[58] A.S. Moura, J. MacGregor Smith, and R.H.C. Takahashi. An ellipsoidal branch-and-cut method for solving mixed-integer quasi-convex problems, 2007.
[59] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988.
[60] I. Nowak, H. Alperin, and S. Vigerske. `LaGO` – an object oriented library for solving MINLPs. In *Global Optimization and Constraint Satisfaction*, number 2861 in Lecture Notes in Computer Science, pages 32–42. Springer, Berlin/Heidelberg, 2003.
[61] A.R.F. O'Grady, I.D.L. Bogle, and E.S. Fraga. Interval analysis in automated design for bounded solutions. *Chemicke Zvesti*, 55(6):376–381, 2001.
[62] Open Source Initiative OSI. Common public license version 1.0. `http://www.opensource.org/licenses/cpl1.0.php`.
[63] M.W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.
[64] Web page for the IBM/CMU MINLP project. `http://egon.cheme.cmu.edu/ibm/page.htm`.
[65] A.T. Phillips and J.B. Rosen. A quadratic assignment formulation of the molecular conformation

problem. Technical report, CSD, University of Minnesota, 1998.
[66] I. Quesada and I.E. Grossmann. Global optimization of bilinear process networks and multicomponent flows. *Computers & Chemical Engineering*, 19(12):1219–1242, 1995.
[67] H. Ratschek and J. Rokne. Interval methods. In R. Horst and P.M. Pardalos, editors, *Handbook of Global Optimization*, volume 1, pages 751–828. Kluwer Academic Publishers, Dordrecht, 1995.
[68] H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, March 1996.
[69] H.S. Ryoo and N.V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19(5):551–566, 1995.
[70] N. V. Sahinidis, I.E. Grossmann, R. E. Fornari, and M. Chathrathi. Optimization model for long range planning in the chemical industry, 1989.
[71] N.V. Sahinidis. BARON: a general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.
[72] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.
[73] H. Schichl. Global optimization in the COCONUT project. In *Numerical Software with Result Verification*, number 2991 in Lecture Notes in Computer Science, pages 243–249. Springer, Berlin/Heidelberg, 2004.
[74] J.P. Shectman and N.V. Sahinidis. A finite algorithm for global minimization of separable concave programs. *Journal of Global Optimization*, 12:1–36, 1998.
[75] E.M.B. Smith. *On the Optimal Design of Continuous Processes*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, October 1996.
[76] E.M.B. Smith and C.C. Pantelides. Global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 21:S791–S796, 1997.
[77] E.M.B. Smith and C.C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 23:457–478, 1999.
[78] M. Tawarmalani and N.V. Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming: Theory, algorithms, software and applications*, volume 65 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dordrecht, 2002.
[79] M. Tawarmalani and N.V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004.
[80] R. Vaidyanathan and M. El-Halwagi. Global optimization of nonconvex MINLPs by interval analysis. In I.E. Grossmann, editor, *Global Optimization in Engineering Design*, pages 175–193. Kluwer Academic Publishers, Dordrecht, 1996.
[81] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica, a Modeling Language for Global Optimization*. MIT Press, Cambridge, MA, 1997.
[82] D. Vandenbussche and G.L. Nemhauser. A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. *Math. Prog.*, 102(3), 2005.
[83] J.P. Vielma, S. Ahmed, and G.L. Nemhauser. A lifted linear programming branch-and-bound algorithm for mixed integer conic quadratic programs. *INFORMS Journal on Computing*, to appear.
[84] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.
[85] L.A. Wolsey. *Integer Programming*. Wiley, 1998.
[86] J. M. Zamora and I. E. Grossmann. A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. *Journal of Global Optimization*, 14:217:249, 1999.
[87] J. Žilinskas and I.D.L. Bogle. Evaluation ranges of functions using balanced random interval arithmetic. *Informatica*, 14(3):403–416, 2003.

Name	tuned		fbbt		abt		no-bt		obbt	
	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub
nvs19	**(-1463.84)**	-971.4	**(-1105.05)**	-1094	**(-1110.59)**	-1094.8	(-42192)	–	**(-1202.95)**	-1073.6
nvs23	**(-1310.02)**	-1091.2	(-233524)	–	(-233558)	–	(-233484)	–	**(-1360.12)**	-1089.4
du-opt5	47.2	8.07366	17.5	8.07366	50.9	8.07366	**15.3**	8.07366	414.4	8.07366
du-opt	57.7	3.55634	**16.2**	3.55634	39.6	3.55634	19.9	3.55634	519.5	3.55634
tln5	(6.74481)	10.6	(6.45067)	10.8	(6.3782)	10.8	(5.68835)	10.6	(7.12437)	10.3
ex1252	61.3	128894	60.8	128894	69.0	128894	(0)	–	**43.4**	128894
ex1233	1144.9	155011	**620.0**	155011	**590.5**	155011	NUM	–	922.5	155011
nous1	(-0.232433)	1.56707	(-0.432783)	1.62254	(-0.433816)	1.56707	(-0.519321)	1.56707	(-0.174955)	1.56707
nous2	**4.5**	0.625967	431.5	0.625967	(-9.501e-02)	1.38432	(-0.467679)	0.625967	10.8	0.625967
tln6	(7.96525)	–	(7.66781)	–	(7.65883)	–	(7.78365)	–	(8.11623)	–
ex1243	**47.8**	83402.5	**47.2**	83402.5	**50.1**	83402.5	(75688.2)	–	54.4	83402.5
csched1	(-41999.5)	-29397.7	(-45987)	–	(-45987)	–	–	–	(-44263.6)	–
m6	141.1	82.2569	145.5	82.2569	231.5	82.2569	**106.9**	82.2569	700.4	82.2569
ex1244	7.8	82042.9	3.8	82042.9	8.2	82042.9	**3.4**	82042.9	41.8	82042.9
fo7	(1.95463)	28.2364	(1.95463)	28.2364	(1.95463)	29.1573	(1.95463)	33.9566	(2.01885)	36.1932
no7_ar2_1	(72.9292)	–	(71.8953)	–	(70.583)	–	**(83.387)**	–	(69.4742)	–
no7_ar3_1	(51.0052)	–	(50.7747)	–	(49.2448)	128.495	(61.2455)	130.911	(44.4205)	–
no7_ar4_1	(46.4066)	127.966	(47.2538)	–	(45.4067)	122.135	(58.9944)	106.393	(40.2347)	–
o7_2	(9.99795)	153.94	(9.8579)	142.047	(9.71642)	154.248	(7.65306)	153.688	(9.39165)	147.859
enpro56pb	772.0	263428	714.2	263428	765.1	263428	**449.2**	263428	(260898)	–
eniplac	**1280.1**	-132117	1442.5	-132117	1622.8	-132117	(-170806)	–	1815.6	-132117
enpro48pb	**49.8**	187277	65.6	187277	84.7	187277	88.2	187277	835.3	187277
tls5	(0.281252)	–	(0.692908)	–	(0.691527)	–	(0)	–	**(1.03417)**	–
tls6	(0.472321)	–	(0.687032)	–	(0.685201)	–	(9.176e-02)	–	**(1.06872)**	–
csched2	(-3.907e+07)	–	(-689735)	–	(-690558)	–	–	–	(-3138600)	–
stockcycle	133.8	119949	115.8	119949	129.9	119949	130.9	119949	150.1	119949
cecil_13	(-124286)	–	**(-115771)**	-115656	**(-115774)**	-115656	**(-116363)**	–	**(-115843)**	-115656
lop97icx	**(2654.42)**	–	**(2662.7)**	–	**(2658.59)**	–	(2613.1)	–	**(2654.85)**	–

Table 11. Tests on bounds tightening variants for MINLP instances. The "NUM" for **no_bt** on instance *ex1233* is caused by a numerically unstable initial linearization $\mathbf{LP}_0$.

	tuned		fbbt		abt		no-bt		obbt	
Name	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub
Synheat	(140732)	155174	(140667)	155376	(151425)	154997	(139465)	155510	(143798)	154997
SynheatMod	(150668)	154997	(150030)	154997	(151379)	154997	(135633)	160497	(151396)	154997
c-sched-4-7	**(-433662)**	–	(-6.222e+07)	–	(-9.089e+07)	–	(-1.579e+08)	–	(-5.602e+07)	–
par72	**(-21696.5)**	–	**(-337822)**	2423.41	**(-411453)**	2423.41	(-6.053e+07)	–	(-1.971e+07)	2448.61
Multistage	(-57499.5)	–	(-56658.2)	–	(-56658.2)	–	–	–	(-29200.4)	–
space-25	**(122.365)**	–	(103.979)	–	(103.825)	–	(42.6826)	–	(100.266)	–
space-25-r	**(69.7149)**	–	**(69.6589)**	–	**(69.6244)**	–	(61.7838)	–	**(69.6452)**	–
trimlon7	(5.06921)	–	(5.05375)	21.1	(5.06725)	–	(5.0581)	–	(5.64155)	–
trimlon12	**(16.1689)**	–	(13.3971)	–	(13.3971)	–	(11.5577)	–	**(16.0731)**	–
iran8x32	(5087.56)	5393.52	(5084.18)	5554.62	(5070.81)	5539.45	(5094.53)	5591.68	(5039.13)	5575.38
ibell3a	975.4	878785	1520.5	878785	1483.0	878785	**618.9**	878785	1414.7	878785
ibienst1	4536.7	48.7377	5143.1	48.7377	5580.1	48.7377	**4207.4**	48.7377	(24.923)	54.6463
imisc07	(1765.82)	2975.27	(1790.73)	3255.2	(1769.01)	3094.7	(1811.63)	–	(1442.17)	–
nConvPl	(-8268.21)	-4362.17	(-8267.89)	-4362.17	(-8292.82)	-4362.17	(-8741.39)	–	(-8302.14)	-4362.17
foulds3	(-76.1689)	-8	(-75.5308)	-8	(-79.2133)	-8	(-80.1292)	-8	(-75.1851)	-8
barton	(-113.257)	–	(-105.795)	–	(-105.795)	–	(-113.267)	–	**(-104.034)**	–
CLay0203H	(0)	–	(41573.1)	41573.3	(41573.2)	41573.3	(0)	–	**3750.3**	41573.2
CLay0204H	(0)	–	**68.0**	6545	122.3	6545	(0)	7300	627.8	6545
airConduct	(24.5116)	26.8327	(24.512)	25.4557	(24.512)	26.8327	(24.2041)	–	(24.512)	26.8327
shortfall_30_0	**104.9**	-1.08072	(-1.08075)	–	(-1.08075)	-1.07975	646.6	-1.08072	193.0	-1.08072
shortfall_30_1	(-1.08838)	-1.07865	(-1.08834)	-1.07207	(-1.08834)	-1.07865	(-1.08832)	-1.07489	(-1.0884)	-1.07865
classical_40_0	(-8.289e-02)	-7.861e-02	1187.7	-8.152e-02	1003.1	-8.152e-02	**578.4**	-8.152e-02	1630.3	-8.152e-02
classical_40_1	6031.5	-8.475e-02	2054.9	-8.475e-02	1530.1	-8.475e-02	**663.3**	-8.475e-02	2126.2	-8.475e-02
robust_30_0	1.3	-4.545e-02	1.3	-4.545e-02	1.6	-4.545e-02	1.4	-4.545e-02	1.4	-4.545e-02
robust_30_1	135.7	-4.775e-02	(-9.091e-02)	-4.726e-02	(-9.108e-02)	-4.580e-02	**57.7**	-4.775e-02	(-7.049e-02)	-4.739e-02

Table 12. Tests on bounds tightening variants for MINLP instances (continued).

	tuned		fbbt		abt		no-bt		obbt	
Name	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub
ex5_2_5	(-5186.13)	-3500	(-5185.4)	-3500	(-5245.9)	-3500	(-5395.08)	-3500	(-4588.03)	-3500
ex5_3_3	(1.66301)	3.39097	(1.65313)	3.30464	(1.65883)	3.23402	(1.63485)	3.23402	(1.67527)	3.23402
foulds3	(-76.1689)	-8	(-75.5308)	-8	(-79.2133)	-8	(-80.1292)	-8	(-75.1851)	-8
qp1	(-0.130191)	8.093e-04	(-0.13365)	8.093e-04	(-0.128582)	8.093e-04	–	8.093e-04	(-0.13861)	8.093e-04
qp2	(-0.143293)	8.093e-04	(-0.121931)	8.093e-04	(-0.13552)	8.093e-04	–	8.093e-04	(-0.136783)	8.093e-04
qp3	(-0.303239)	8.093e-04	(-0.367258)	8.093e-04	(-0.379187)	8.093e-04	–	8.093e-04	(-0.23947)	8.093e-04
Hicks_5	**7.5**	227.257	(2.630e-02)	227.257	(1.6942)	227.257	(1.18405)	227.257	(141.096)	227.257
Hicks_10	**29.0**	227.257	(2.914e-03)	227.257	(0.868933)	227.257	(1.833e-02)	227.257	(0.261244)	227.257
Hicks_20	111.2	227.257	(2.975e-03)	227.257	(0.515046)	227.257	(5.346e-02)	227.257	**30.7**	227.257
Hicks_50	**735.4**	227.257	(0.197572)	227.257	(0.284617)	227.257	(6.934e-03)	227.257	(0.443255)	227.257
Hicks_100	2684.3	227.257	(0.446834)	227.257	(0.334031)	227.257	(5.032e-03)	227.257	**481.8**	227.257
spar030-060-1	3226.8	-706	**2958.3**	-706	**3071.1**	-706	**2794.3**	-706	(-707.375)	-696.5
spar030-060-2	20.7	-1377.17	2.0	-1377.17	6.9	-1377.17	**1.7**	-1377.17	468.2	-1377.17
spar040-050-1	**(-1172.88)**	-1154.5	**(-1173.29)**	-1154.5	**(-1174.81)**	-1154.5	**(-1169.87)**	-1154.5	(-1347.16)	-1154.5
spar040-050-2	3238.4	-1430.98	3223.2	-1430.98	3215.8	-1430.98	**2635.0**	-1430.98	(-1593.19)	-1430.98
spar040-060-1	**(-1735.62)**	-1322.67	**(-1735.5)**	-1322.67	**(-1737.06)**	-1322.67	**(-1708.97)**	-1311.06	(-2110.22)	-1311.06
spar050-040-1	**(-1613.34)**	-1411	**(-1612.84)**	-1411	**(-1613.78)**	-1411	**(-1594.53)**	-1411	(-1948.22)	-1411
spar050-050-1	**(-2254.94)**	-1193	**(-2254.25)**	-1193	**(-2257)**	-1193	**(-2221.94)**	-1193	(-2886.12)	-1193
spar060-020-1	405.2	-1212	376.5	-1212	421.1	-1212	**276.2**	-1212	5167.5	-1212
spar060-020-2	33.1	-1925.5	5.3	-1925.5	17.3	-1925.5	**4.5**	-1925.5	2072.6	-1925.5
spar070-025-1	**(-2881.28)**	-2538.91	**(-2879.97)**	-2538.91	**(-2882.72)**	-2538.91	**(-2861.72)**	-2538.91	(-3333.94)	-2538.91
spar080-025-1	**(-3786.06)**	-3157	**(-3782.25)**	-3157	**(-3790)**	-3157	**(-3755.56)**	-3157	(-4386.94)	-3157
spar090-025-1	**(-4899.62)**	-3361.5	**(-4896.69)**	-3361.5	**(-4905)**	-3361.5	**(-4862.06)**	-3361.5	(-5671.62)	-3361.5
dualc8	(18309.1)	18309.2	(18309.2)	18309.2	(18309.1)	18309.2	(18309.2)	18309.2	(18309.2)	18309.2
dual4	**(-78.3223)**	0.746091	**(-65.9165)**	0.746091	**(-70.537)**	0.746091	**(-68.8498)**	0.746091	(-305.806)	0.746091
dual1	(-78.21)	3.501e-02	**(-50.2085)**	3.501e-02	**(-56.7133)**	3.501e-02	**(-58.0487)**	3.501e-02	(-214.391)	3.501e-02
dual2	(-221.494)	3.373e-02	**(-60.5728)**	3.373e-02	(-221.494)	3.373e-02	**(-71.5777)**	3.373e-02	(-221.494)	3.373e-02
qadlittl	2186.2	480319	2215.9	480319	2674.6	480319	**760.6**	480319	2227.1	480319
cvxqp1_s	**(10680.2)**	–	**(10681)**	–	**(10680.2)**	–	(9577.69)	–	**(10381.7)**	–
cvxqp2_s	(7424.12)	–	(7422.84)	–	(7422.35)	–	(6906.99)	–	(7301.97)	–
cvxqp3_s	(11943.4)	13727.9	(11932.5)	12024.6	(11934)	12103.9	(11932.7)	12035	(11943)	11947.2
values	(-7.28079)	-1.39662	(-8.13076)	-1.39662	(-8.21165)	-1.39662	(-8.06712)	-1.39662	(-8.51661)	-1.39662
qbrandy	**32.7**	28375.1	85.2	28375.1	199.1	28375.1	43.0	28375.1	136.1	28375.1
qshare1b	6571.0	720078	(720078)	720078	(720078)	720078	(719944)	–	6707.9	720078
qetamacr	(63739.6)	86760.4	(70321.8)	86760.4	(69397.5)	86760.4	(70810.4)	86760.4	(63555.4)	86760.4
gouldqp2	(-0.15444)	1.843e-04	(-0.154362)	1.843e-04	(-0.154406)	1.843e-04	(-0.154254)	1.843e-04	(-0.206172)	1.843e-04
primal4	1180.6	-0.746091	**7.4**	-0.746091	(-0.746759)	-0.746091	**8.0**	-0.746091	800.4	-0.746091
stcqp1	**(151116)**	155144	**(151296)**	155144	**(151254)**	155144	**(151341)**	155144	(148329)	155144

Table 13. Tests on bounds tightening variants for continuous NLP instances.

	str-br		rb-int-br-rev		br-plain		pure-vt		rb-inf	
Name	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub
nvs19	208.7	-1098.4	**31.8**	-1098.4	(-1463.84)	-971.4	(-1230.4)	-1078.8	58.2	-1098.4
nvs23	927.3	-1125.2	**153.5**	-1125.2	(-1310.02)	-1091.2	(-1283.64)	-1083.2	310.1	-1125.2
du-opt5	1777.6	8.07366	(7.96564)	157.292	**47.2**	8.07366	**47.3**	8.07366	525.6	8.07366
du-opt	564.6	3.55634	359.6	3.55634	**57.7**	3.55634	**57.9**	3.55634	301.2	3.55634
tln5	132.3	10.3	57.0	10.3	(6.74481)	10.6	95.6	10.3	**41.8**	10.3
ex1252	89.8	128894	91.7	128894	61.3	128894	**22.4**	128894	(96158.2)	140049
ex1233	(98260.7)	313541	(85451.1)	313541	1144.9	155011	**911.1**	155011	(90049.1)	313541
nous1	(1.13599)	1.56707	**(1.54113)**	1.56707	(-0.232433)	1.56707	(0.585914)	1.56707	(0.469265)	1.56707
nous2	577.1	0.625967	463.6	0.625967	**4.5**	0.625967	5.2	0.625967	104.7	0.625967
tln6	2133.0	15.3	(15.1)	15.6	(7.96525)	–	(7.98961)	–	**1451.6**	15.3
ex1243	9.4	83402.5	4.8	83402.5	47.8	83402.5	103.5	83402.5	**2.9**	83402.5
csched1	52.1	-30639.3	24.9	-30639.3	(-41999.5)	-29397.7	**9.1**	-30639.3	(-30639.3)	-30639.3
m6	(82.193)	82.2855	(61.4593)	82.2855	**141.1**	82.2569	**138.9**	82.2569	(78.8575)	111.297
ex1244	135.0	82042.9	(73629.7)	85275.1	**7.8**	82042.9	8.9	82042.9	744.9	82042.9
fo7	(9.17862)	26.8476	(4.50536)	25.8579	(1.95463)	28.2364	(1.95463)	28.2364	(6.34519)	28.1697
no7_ar2_1	**(105.138)**	107.863	(92.6172)	117.579	(72.9292)	–	(72.9557)	–	**(95.1179)**	–
no7_ar3_1	**(90.9676)**	116.287	**(88.3894)**	137.694	(51.0052)	–	(50.9655)	–	**(85.2688)**	134.893
no7_ar4_1	**(84.8417)**	125.531	(69.8584)	120.749	(46.4066)	127.966	(46.4613)	127.966	(71.7896)	–
o7_2	(65.6153)	158.178	**(67.0205)**	140.876	(9.99795)	153.94	(10.0484)	153.94	**(74.0129)**	137.399
enpro56pb	57.3	263428	64.4	263428	772.0	263428	751.3	263428	**44.4**	263428
eniplac	(-134499)	–	**1314.5**	-132117	**1280.1**	-132117	**1290.2**	-132117	2267.8	-132117
enpro48pb	83.2	187277	84.8	187277	**49.8**	187277	**49.7**	187277	454.5	187277
tls5	**(5.6)**	–	(4.07901)	–	(0.281252)	–	(0.281717)	–	(4.75417)	–
tls6	**(5.66333)**	–	**(5.54602)**	–	(0.472321)	–	(0.490006)	–	**(5.25982)**	–
csched2	(-305830)	–	(-255056)	–	(-3.90708e+07)	–	(-3.90219e+07)	–	(-305830)	–
stockcycle	1581.6	119949	(119933)	145727	**133.8**	119949	**134.2**	119949	(119841)	120199
cecil_13	**(-116000)**	–	**(-116025)**	–	(-124286)	–	(-124277)	–	**(-116004)**	–
lop97icx	**(3091.46)**	–	**(3073.41)**	–	(2654.42)	–	(2654.73)	–	**(3088.98)**	–

Table 14. Tests for branching techniques on MINLP instances.

Name	str-br		rb-int-br-rev		br-plain		pure-vt		rb-inf	
	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub
Synheat	(81061.9)	161010	(76212.1)	161010	**(140732)**	155174	**(144347)**	154997	(75746.4)	161010
SynheatMod	(79384.7)	161010	(79573.3)	161010	**(150668)**	154997	**(150311)**	154997	(78234)	161010
c-sched-4-7	(-261947)	–	(-254372)	–	(-433662)	–	(-223972)	–	(-263059)	–
par72	(-3234.77)	19057.7	(-8535.91)	–	(-21696.5)	–	(374.956)	–	(-3234.77)	–
Multistage	**533.7**	-7582.62	(-36772.2)	–	(-57499.5)	–	2565.2	-7582.62	(-94089.1)	–
space-25	(87.9277)	–	(97.0191)	–	**(122.365)**	–	**(120.165)**	–	(85.5853)	–
space-25-r	(70.8325)	–	(69.2622)	–	(69.7149)	–	(69.7442)	–	(70.8325)	–
trimlon7	(14.7173)	–	3593.8	15	(5.06921)	–	(5.02803)	–	**1012.6**	15
trimlon12	**(30.2292)**	–	**(29.7712)**	–	(16.1689)	–	(16.1688)	–	**(30.2558)**	–
iran8x32	(5229.01)	5649.76	5095.2	5255.45	(5087.56)	5393.52	(5089.34)	5529.02	5125.9	5255.45
ibell3a	3381.4	878785	**953.3**	878785	**975.4**	878785	**922.1**	878785	1213.8	878785
ibienst1	5265.8	48.7377	4292.4	48.7377	4536.7	48.7377	4328.2	48.7377	3671.0	48.7377
imisc07	(2041.18)	2815.47	**(2536.14)**	2814.37	(1765.82)	2975.27	(1765.25)	2975.27	(2291.95)	2814.54
nConvPl	(-8842.22)	-4579.99	(-8793.61)	-4579.99	(-8268.21)	-4362.17	(-8267.71)	-4362.17	(-8842.22)	-4579.99
foulds3	(-55.7946)	-8	(-57.3748)	-8	(-76.1689)	-8	(-68.7305)	-8	(-55.4884)	-8
barton	**(-102.736)**	–	**(-102.924)**	–	(-113.257)	–	**(-103.544)**	–	**(-102.736)**	–
CLay0203H	(4761.22)	41907.5	**(40179)**	41573.3	(0)	–	(33667.7)	–	**(40104.9)**	41737.5
CLay0204H	101.8	6545	(3025)	7025	(0)	–	**64.5**	6545	(3025)	6775
airConduct	(24.5134)	26.8327	(24.5108)	26.8327	(24.5116)	26.8327	(24.5106)	25.4557	(24.5117)	26.8327
shortfall_30_0	791.0	-1.08072	867.5	-1.08072	**104.9**	-1.08072	324.9	-1.08072	2888.3	-1.08072
shortfall_30_1	(-1.08833)	-1.07865	(-1.08833)	-1.07865	(-1.08838)	-1.07865	(-1.08731)	-1.08069	(-1.08841)	-1.07865
classical_40_0	1195.9	-8.152e-02	**1070.1**	-8.152e-02	(-8.289e-02)	-7.861e-02	1254.9	-8.152e-02	(-8.168e-02)	-7.833e-02
classical_40_1	882.6	-8.475e-02	**146.8**	-8.475e-02	6031.5	-8.475e-02	2006.4	-8.475e-02	190.3	-8.475e-02
robust_30_0	1.4	-4.545e-02	1.4	-4.545e-02	1.3	-4.545e-02	1.2	-4.545e-02	1.7	-4.545e-02
robust_30_1	(-4.780e-02)	-4.771e-02	(-4.775e-02)	-4.351e-02	**135.7**	-4.775e-02	(-7.035e-02)	-4.741e-02	(-4.797e-02)	-4.748e-02

Table 15. Tests for branching techniques on MINLP instances (continued).

	str-br		rb-int-br-rev		br-plain		pure-vt		rb-inf	
Name	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub	time (lb)	ub
ex5_2_5	(-7078.7)	-3500	(-6900.79)	-3500	**(-5186.13)**	-3500	(-7225.22)	-3500	(-7079.57)	-3500
ex5_3_3	(1.88559)	3.24012	(1.83954)	3.24012	(1.66301)	3.39097	(1.65172)	3.39097	(1.8111)	3.23402
foulds3	(-55.7946)	-8	(-57.3748)	-8	(-76.1689)	-8	(-68.7305)	-8	(-55.4884)	-8
qp1	(-0.226998)	8.093e-04	**(-8.387e-02)**	8.093e-04	**(-0.130191)**	8.093e-04	**(-9.678e-02)**	8.093e-04	(-0.356874)	8.093e-04
qp2	(-0.233894)	8.093e-04	(-8.195e-02)	8.093e-04	(-0.143293)	8.093e-04	(-9.556e-02)	8.093e-04	(-0.303211)	8.093e-04
qp3	(-0.320343)	8.093e-04	(-0.28616)	8.093e-04	(-0.303239)	8.093e-04	(-0.33229)	8.093e-04	(-0.308834)	8.093e-04
spar030-060-1	(-754.843)	-706	**1817.1**	-706	3226.8	-706	2460.6	-706	**1908.9**	-706
spar030-060-2	48.6	-1377.17	25.4	-1377.17	**20.7**	-1377.17	**20.4**	-1377.17	**21.7**	-1377.17
spar040-050-1	(-1358.83)	-1154.5	**(-1183.65)**	-1154.5	**(-1172.88)**	-1154.5	**(-1195.22)**	-1154.5	**(-1175.81)**	-1154.5
spar040-050-2	(-1557.62)	-1430.98	**3106.3**	-1430.98	3238.4	-1430.98	4269.2	-1430.98	**2905.5**	-1430.98
spar040-060-1	(-1939.54)	-1322.67	(-1716.17)	-1311.06	(-1735.62)	-1322.67	(-1725.41)	-1322.67	(-1699.1)	-1322.67
spar050-040-1	(-1809.56)	-1411	**(-1595.22)**	-1411	**(-1613.34)**	-1411	**(-1616.34)**	-1411	**(-1601.27)**	-1411
spar050-050-1	(-2496.53)	-1193	(-2204.72)	-1193	(-2254.94)	-1193	(-2289.31)	-1193	(-2202.69)	-1193
spar060-020-1	1550.7	-1212	422.2	-1212	**405.2**	-1212	729.0	-1212	**366.5**	-1212
spar060-020-2	47.1	-1925.5	45.5	-1925.5	33.1	-1925.5	36.3	-1925.5	45.1	-1925.5
spar070-025-1	(-3036.2)	-2538.91	(-2853.54)	-2538.91	(-2881.28)	-2538.91	(-2848.78)	-2538.91	(-2852.95)	-2538.91
spar080-025-1	(-4030)	-3157	(-3759.91)	-3157	(-3786.06)	-3157	(-3817.44)	-3157	(-3763.5)	-3157
spar090-025-1	(-5181.81)	-3372.5	(-4865.31)	-3361.5	(-4899.62)	-3361.5	(-5261.44)	-3361.5	(-4879.44)	-3372.5
cvxqp1_s	(10769)	–	(10790.5)	12468	(10680.2)	–	(11159.4)	11800	(10764)	–
cvxqp2_s	(7327.87)	–	(7297.41)	–	(7424.12)	–	**(7677.29)**	–	(7329.36)	–
cvxqp3_s	(11942.9)	–	(11942.8)	–	(11943.4)	13727.9	**3957.4**	11943.4	(11942.9)	–
dual1	(-214.431)	3.501e-02	(-214.431)	3.501e-02	**(-78.21)**	3.501e-02	(-214.431)	3.501e-02	(-214.431)	3.501e-02
dual2	(-221.265)	3.373e-02	(-221.494)	3.373e-02	(-221.494)	3.373e-02	(-221.265)	3.373e-02	(-221.265)	3.373e-02
dual4	(-304.143)	0.746091	(-304.143)	0.746091	**(-78.3223)**	0.746091	**(-79.7629)**	0.746091	(-304.143)	0.746091
gouldqp2	(-0.171471)	1.843e-04	(-0.165717)	1.843e-04	(-0.15444)	1.843e-04	(-0.161887)	1.843e-04	(-0.162164)	1.843e-04
qbrandy	317.8	28375.1	46.7	28375.1	**32.7**	28375.1	(28375.1)	28411.7	(28375.1)	–
qadlittl	(480318)	–	6404.6	480319	**2186.2**	480319	(480317)	–	(480304)	–
qshare1b	(720056)	–	(720048)	–	6571.0	720078	(720078)	–	(720078)	–
dualc8	(18307.1)	18309.3	(18309.2)	18309.2	(18309.1)	18309.2	(18296.9)	18323.6	(17741.6)	18324.2
qetamacr	(63739.6)	86760.4	(63739.6)	86760.4	(63739.6)	86760.4	(63739.6)	86760.4	(63739.6)	86760.4
primal4	1214.5	-0.746091	1384.4	-0.746091	1180.6	-0.746091	1194.9	-0.746091	1216.1	-0.746091
stcqp1	(148329)	155144	(148329)	155144	**(151116)**	155144	**(151064)**	155144	(148329)	155144
values	(-13.6214)	-1.39662	(-13.6214)	-1.39662	(-7.28079)	-1.39662	(-7.03803)	-1.39662	(-13.6214)	-1.39662

Table 16. Tests for branching techniques on continuous NLP instances.

Name	str-br			rb-int-br-rev			br-plain			pure-vt			rb-inf		
	time	nodes	depth	time	nodes	depth	time	nodes	depth	time	nodes	depth	time	nodes	depth
Multistage	**533.7**	**1517**	**85**							2565.2	171396	**87**			
trimlon7				3593.8	753835	52							**1012.6**	**208004**	**45**
csched1	35.6	**774**	46	24.9	3810	66				**5.4**	880	**36**			
ex1233							1528.8	141176	**68**	**911.1**	**76662**	**70**			
ex1243	9.4	196	95	4.8	650	65	47.8	17115	89	103.5	22256	**41**	**2.9**	**157**	57
ex1244	135.0	604	71				**7.8**	**240**	**16**	8.9	410	21	744.9	119672	85
ex1252	89.8	**1357**	**23**	91.7	17519	46	48.3	9956	30	**15.5**	2359	**25**			
nous2	577.1	4348	143	463.6	75630	88	**4.5**	**270**	61	5.2	324	**31**	104.7	14344	111
nvs19	208.7	4924	**39**	**31.8**	**4004**	**39**							58.2	7534	**40**
nvs23	927.3	17166	**50**	**153.5**	**15450**	**51**							310.1	28350	**47**
du-opt	564.6	287	102	359.6	192	67	**57.7**	**71**	**10**	**57.9**	**71**	**10**	301.2	177	59
du-opt5	1777.6	1149	64				**47.2**	**112**	**13**	**47.3**	**111**	**13**	525.6	407	43
eniplac				**1314.5**	**118360**	**23**	**1280.1**	173026	**24**	**1290.2**	173026	**24**	2267.8	194677	29
enpro56pb	57.3	**220**	35	64.4	1062	36	772.0	3032	**29**	751.3	3032	**29**	**44.4**	503	33
enpro48pb	83.2	**222**	42	84.8	782	46	**49.8**	408	**18**	**49.7**	408	**18**	454.5	3573	45
m6							**141.1**	**14882**	**25**	**138.9**	**14882**	**25**			
stockcycle	1581.6	**7218**	137				**133.8**	8860	**108**	**134.2**	8860	**108**			
tln5	132.3	**5768**	**23**	57.0	22440	28				95.6	42223	46	**41.8**	15610	26
tln6	2133.0	**58007**	**34**										**1451.6**	376797	**31**
iran8x32				**5095.2**	**60354**	**81**							**5125.9**	68626	**89**
ibell3a	3381.4	**22111**	67	**953.3**	53384	59	**975.4**	37706	**31**	**922.1**	42641	**33**	1213.8	56548	66
ibienst1	5265.8	**3462**	**21**	4292.4	48912	24	4536.7	42754	**23**	4328.2	42388	**23**	**3671.0**	42998	24
CLay0204H	101.8	**291**	**62**							**64.5**	2512	72			
shortfall_30_0	791.0	**597**	79	867.5	4303	95	**104.9**	6128	61	324.9	26026	**44**	2888.3	33489	89
classical_40_0	1195.9	**783**	67	**1070.1**	16857	68				1254.9	32393	**51**			
classical_40_1	882.6	**593**	52	**146.8**	1485	**46**	6031.5	250966	68	2006.4	57899	**45**	190.3	1565	**47**
spar030-060-1				**1817.1**	**100786**	**24**	3226.8	166140	**22**	2460.6	126538	**23**	**1908.9**	**102234**	25
spar030-060-2	48.6	**20**	**4**	25.4	86	14	**20.7**	118	12	**20.4**	108	9	**21.7**	62	13
spar040-050-2				**3106.3**	125946	**26**	3238.4	129420	**24**	4269.2	152992	34	**2905.5**	**112460**	**24**
spar060-020-1	1550.7	**5004**	**20**	422.2	23392	23	405.2	23204	22	729.0	42274	**21**	**366.5**	19398	**21**
spar060-020-2	47.1	**58**	**5**	45.5	158	7	**33.1**	284	8	**36.3**	242	8	45.1	158	7
qbrandy	317.8	1587	83	46.7	1025	70	**32.7**	**704**	**61**						
qadlittl				6404.6	400781	**167**	**2186.2**	**138477**	**157**						

Table 17. Number of nodes and maximum sBB tree depth for MINLP and continuous NLP instances.