Noname manuscript No.
(will be inserted by the editor)

Bound reduction using pairs of linear inequalities

Pietro Belotti

Received: date / Accepted: date

Abstract We describe a procedure to reduce variable bounds in Mixed Integer Nonlinear Programming (MINLP) as well as Mixed Integer Linear Programming (MILP) problems. The procedure works by combining *pairs* of inequalities of a linear programming (LP) relaxation of the problem. This bound reduction technique extends the *feasibility based* bound reduction technique on linear functions, used in MINLP and MILP. However, it can also be seen as a special case of *optimality based* bound reduction, a method to infer variable bounds from an LP relaxation of the problem.

For an LP relaxation with m constraints and n variables, there are $O(m^2)$ pairs of constraints, and a naïve implementation of our bound reduction scheme has complexity $O(n^3)$ for each pair. Therefore, its overall complexity $O(m^2n^3)$ can be prohibitive for relatively large problems. We have developed a more efficient procedure that has complexity $O(m^2n^2)$, and embedded it in two Open-Source solvers: one for MINLP and one for MILP. We provide computational results which substantiate the utility of this bound reduction technique for several instances.

Keywords MINLP · MILP · Bound reduction · Implied bounds

1 Introduction

We present an algorithm to reduce the bounds of a set of variables in a Mixed Integer Nonlinear Programming (MINLP) problem such as the following:

$$\begin{array}{rll} \min & g_0(x) & \\ \text{s.t.} & g_j(x) \leq 0 & \forall j = 1, 2 \dots, m \\ & \ell_i \leq x_i \leq u_i & \forall i = 1, 2 \dots, n_0 \\ & x \in \mathbb{Z}^q \times \mathbb{R}^{n_0 - q}, & \end{array} \tag{1}$$

where $g_j : \mathbb{R}^{n_0} \to \mathbb{R}$, for $j = 0, 1, 2 \dots, m$, is in general a multivariate, nonconvex function; n_0 is the number of variables; and q is the number of integer variables. The variable bounds form the hyperrectangle $\{x \in \mathbb{R}^{n_0} : \ell_i \leq x_i \leq u_i \ \forall i = 1, 2 \dots, n_0\}$, which we

Dept. of Mathematical Sciences, Clemson University. Clemson SC 29634.
Email: pbelott@clemson.edu

denote $[\ell, u]$. Although bounds ℓ_i and u_i on x_i are usually assumed finite, the procedure discussed here works with infinite bounds too. An important subclass of (1), in which all g_j's are affine, is that of Mixed Integer Linear Programming (MILP) problems.

In general, nonconvex problems such as (1) are solved by implicitly enumerating all local minima through *branch-and-bound* procedures (BB in the remainder of this paper) [14]. Several BB procedures work by recursively partitioning the hyperrectangle $[\ell, u]$ into two or more subsets, creating BB *subproblems*, on which the BB is applied. In order to solve a subproblem, a *lower bound* on the optimal value of $g_0(x)$ is necessary. Several techniques have been developed to determine a lower bound on the optimal solution of (1) or of any restriction of (1) to a subset $[\ell', u'] \subseteq [\ell, u]$, for the general case in which the g_j's are nonconvex functions [18, 27], convex functions [1, 6], and for classes of problems such as Quadratically Constrained Quadratic Programming (QCQP) [20].

We do not discuss lower bounding techniques any further as they are out of the scope of this work. However, it is worth pointing out that the lower bound at a BB subproblem depends strongly on the variable bounds. In fact, the lower bound can improve dramatically when *bound tightening* (BT) techniques (also known as *bound reduction*) are employed. These techniques return a sub-interval $[\ell'', u''] \subseteq [\ell', u']$ that retains at least one optimal solution to the restriction of (1) to $[\ell', u']$.

One well known bound reduction technique is the *Feasibility Based Bound Tightening* (FBBT), developed originally for Artificial Intelligence [9] and Constraint Programming and then successfully employed in Nonlinear Programming [19] and Integer Programming [2, 25]. Optimization solvers apply FBBT by inferring new variable bounds from single constraints $g_j(x) \leq 0$, often using interval methods [13, 22]. FBBT can be repeated multiple times for each constraint; however, while every iteration may produce a tighter bound interval, this process may not terminate [9]. This lack of convergence of FBBT to a *fixed point*, where another round of bound reduction would yield no tightened bound, has been addressed through Linear Optimization techniques [4].

Another reduction technique relies on the LP relaxation $\min\{c^\top x : Ax \geq b, x \in [\ell, u]\}$ of (1). If one is available, valid bounds are

$$\min\{x_i : Ax \geq b, x \in [\ell, u]\} \leq x_i \leq \max\{x_i : Ax \geq b, x \in [\ell, u]\}. \tag{2}$$

This technique, called *Optimality Based Bound Tightening* (OBBT) [21], can be used to strengthen the linear relaxation. Given that computing all lower and upper bounds amounts to solving two LP problems for each variable, repeated calls to OBBT are discouraged. In [7], the iterated OBBT is proven to yield equivalent results to a method that fixes the variable whose bounds are tightened. Other BT methods use the *reduced costs* at the optimal solution of the linearization [23].

Our contribution is a bound tightening procedure that uses *pairs* of inequalities of the linear relaxation of a MINLP or MILP problem. Specifically, we apply FBBT to a convex combination of the two inequalities, yielding tighter bounds on the variables with nonzero coefficient in either or both inequalities. To the best of our knowledge, this technique was not presented or implemented before. We first recall, in the next section, the application of FBBT on a single inequality, and describe our method in Section 3. In Section 4 we discuss an efficient procedure for reducing bounds using all pairs of inequalities. Section 5 presents computational results on problems from problem libraries available on the Web.

2 Implied bounds from a single inequality

Let us first recall the FBBT procedure applied to linear inequalities, used in both MILP and MINLP. Suppose that an LP relaxation of problem (1) is available. In MILP, this is simply the LP problem obtained by relaxing the integrality constraints on the variables. For MINLP problems, an LP relaxation is computed by first generating a *reformulation* of the problem, which introduces *auxiliary variables* for each nonlinear expression of the MINLP, and then applying linear relaxation techniques to each newly generated auxiliary variable [15, 18, 26]. Most of today's MINLP solvers apply this type of technique [24, 5, 16].

Suppose the LP relaxation has m constraints and n variables. In general, $n \geq n_0$ as extra variables are introduced in the relaxation in the MINLP case, while $n = n_0$ in the MILP case. Define index sets for variables and constraints of the LP relaxation, $N = \{1, 2 \ldots, n\}$ and $M = \{1, 2 \ldots, m\}$. Denote the LP relaxation as

$$\begin{array}{rll} \min & \sum_{i \in N} c_i x_i & \\ \text{s.t.} & \sum_{i \in N} a_{ji} x_i \geq b_j & \forall j \in M \\ & \ell_i \leq x_i \leq u_i & \forall i \in N. \end{array} \tag{3}$$

Consider a generic inequality of the relaxation, $\sum_{i \in N} a_i x_i \geq b$. Define $P^+ = \{i \in N : a_i > 0\}$ and $P^- = \{i \in N : a_i < 0\}$. Then the following are valid bounds [2, 25]:

$$\begin{array}{l} \forall i \in P^+,\ x_i \geq \frac{1}{a_i} \left(b - \sum_{k \in P^+ \setminus \{i\}} a_k u_k - \sum_{k \in P^- \setminus \{i\}} a_k \ell_k \right); \\ \forall i \in P^-,\ x_i \leq \frac{1}{a_i} \left(b - \sum_{k \in P^+ \setminus \{i\}} a_k \ell_k - \sum_{k \in P^- \setminus \{i\}} a_k u_k \right). \end{array} \tag{4}$$

Example. Consider the inequality $x_1 - x_2 \geq 3$, with $(x_1, x_2) \in [1, 5] \times [1, 3]$. Then $x_1 \geq 3 + \ell_2 = 4$, thus returning a new value of $\ell_1 = 4$. Also, $x_2 \leq u_1 - 3 = 2$. As a result, the new bounds for x_1 and x_2 are $[4, 5]$ and $[1, 2]$, respectively.

3 Implied bounds from two inequalities

Consider now two inequalities of the LP relaxation (3) of (1):

$$\sum_{i \in N} a'_i x_i \geq b', \qquad \sum_{i \in N} a''_i x_i \geq b''. \tag{5}$$

Their convex combination, defined by a parameter $\lambda \in [0, 1]$, is

$$\lambda \sum_{i \in N} a'_i x_i + (1 - \lambda) \sum_{i \in N} a''_i x_i \geq \lambda b' + (1 - \lambda) b'', \tag{6}$$

or, for short, $\sum_{i \in N} \bar{a}_i(\lambda) x_i \geq \bar{b}(\lambda)$, where $\bar{a}_i(\lambda) = \lambda a'_i + (1 - \lambda) a''_i$ and $\bar{b}(\lambda) = \lambda b' + (1 - \lambda) b''$. Define $P^+(\lambda) = \{i \in N : \bar{a}_i(\lambda) > 0\}$ and $P^-(\lambda) = \{i \in N : \bar{a}_i(\lambda) < 0\}$. Also, to simplify notation, define

$$\begin{array}{l} L_i(\lambda) = \bar{b}(\lambda) - \sum_{k \in P^+(\lambda) \setminus \{i\}} \bar{a}_k(\lambda) u_k - \sum_{k \in P^-(\lambda) \setminus \{i\}} \bar{a}_k(\lambda) \ell_k, \\ U_i(\lambda) = \bar{b}(\lambda) - \sum_{k \in P^+(\lambda) \setminus \{i\}} \bar{a}_k(\lambda) \ell_k - \sum_{k \in P^-(\lambda) \setminus \{i\}} \bar{a}_k(\lambda) u_k. \end{array}$$

Then (4) can be applied to (6) for any $\lambda \in [0, 1]$:

$$\forall i \in P^+(\lambda), x_i \geq \frac{L_i(\lambda)}{\bar{a}_i(\lambda)}; \qquad \forall i \in P^-(\lambda), x_i \leq \frac{U_i(\lambda)}{\bar{a}_i(\lambda)}. \tag{7}$$

Clearly, the fact that variables with positive (resp. negative) coefficient may only improve their lower (resp. upper) bound only depends on the sign of the inequality. We derive our result using inequality constraints, but our analysis is easily extended to equality and range constraints.

Example. Consider the two inequalities

$$x_1 + x_2 + x_3 \geq 3, \qquad x_1 - x_2 + x_3 \geq 2,$$

with $(x_1, x_2, x_3) \in [-1, 3] \times [-1, 1] \times [0, 1]$. From (3) and similar to the example in the previous section, each inequality separately implies a tighter lower bound on x_1:

$$\begin{array}{lll} x_1 + x_2 + x_3 \geq 3 & \Rightarrow & x_1 \geq 3 - u_2 - u_3 = 1 \\ x_1 - x_2 + x_3 \geq 2 & \Rightarrow & x_1 \geq 2 + \ell_2 - u_3 = 0, \end{array}$$

thus returning a new value of $\ell_1 = 1$ (the bounds on x_2 and x_3 are unchanged). However, their convex combination with $\lambda = \frac{1}{2}$ gives a better lower bound:

$$\begin{array}{r} \frac{1}{2}(x_1 + x_2 + x_3) + \frac{1}{2}(x_1 - x_2 + x_3) \geq \frac{1}{2}3 + \frac{1}{2}2 \\ x_1 + x_3 \geq \frac{5}{2} \\ \Rightarrow x_1 \geq \frac{5}{2} - u_3 = \frac{3}{2}. \end{array}$$

As will become clear below, $\lambda = \frac{1}{2}$ yields the tightest bound on x_1 because the coefficient $\lambda - (1 - \lambda) = 2\lambda - 1$ of x_2 in the combination of the two inequalities vanishes for $\lambda = \frac{1}{2}$. Other values of λ, say $\frac{3}{4}$, yield different bounds:

$$\begin{array}{r} \frac{3}{4}(x_1 + x_2 + x_3) + \frac{1}{4}(x_1 - x_2 + x_3) \geq \frac{3}{4}3 + \frac{1}{4}2 \\ x_1 + \frac{1}{2}x_2 + x_3 \geq \frac{11}{4} \\ \Rightarrow x_1 \geq \frac{11}{4} - \frac{1}{2}u_2 - u_3 = \frac{5}{4} < \frac{3}{2}. \end{array}$$

Before describing our bound tightening procedure, it is important to point out some properties of $L_i(\lambda)$ and $U_i(\lambda)$, which are piecewise linear by construction.

Lemma 1 *$L_i(\lambda)$ and $U_i(\lambda)$ are continuous for $\lambda \in \mathbb{R}$.*

Proof It suffices to prove the lemma for $L_i(\lambda)$ as the result can be easily extended to $U_i(\lambda)$.

$$\begin{array}{ll} & \lim_{\epsilon \to 0} L_i(\lambda + \epsilon) - L_i(\lambda) = \\ = & \lim_{\epsilon \to 0} \bar{b}(\lambda + \epsilon) - \bar{b}(\lambda) + \\ & \quad - \sum_{j \in N \setminus \{i\} : \bar{a}_j(\lambda+\epsilon) > 0, \bar{a}_j(\lambda) > 0} (\bar{a}_j(\lambda + \epsilon) - \bar{a}_j(\lambda)) u_j & (\gamma_{pp}) \\ & \quad - \sum_{j \in N \setminus \{i\} : \bar{a}_j(\lambda+\epsilon) < 0, \bar{a}_j(\lambda) < 0} (\bar{a}_j(\lambda + \epsilon) - \bar{a}_j(\lambda)) \ell_j & (\gamma_{nn}) \\ & \quad - \sum_{j \in N \setminus \{i\} : \bar{a}_j(\lambda+\epsilon) > 0, \bar{a}_j(\lambda) < 0} (\bar{a}_j(\lambda + \epsilon) u_j - \bar{a}_j(\lambda) \ell_j) & (\gamma_{pn}) \\ & \quad - \sum_{j \in N \setminus \{i\} : \bar{a}_j(\lambda+\epsilon) < 0, \bar{a}_j(\lambda) > 0} (\bar{a}_j(\lambda + \epsilon) \ell_j - \bar{a}_j(\lambda) u_j). & (\gamma_{np}) \end{array}$$

While $\bar{b}(\lambda + \epsilon) - \bar{b}(\lambda)$, (γ_{pp}), and (γ_{nn}) tend to zero for $\epsilon \to 0$ for continuity of $\bar{b}(\lambda)$ and $\bar{a}_i(\lambda)$, the terms (γ_{pn}) and (γ_{np}) vanish as $\bar{a}_i(\lambda + \epsilon)$ and $\bar{a}_i(\lambda)$, which have opposite sign in (γ_{pn}) and (γ_{np}), must tend to equality for $\epsilon \to 0$ for continuity and hence tend both to 0. Because the limit is null for any λ, $L_i(\lambda)$ and, analogously, $U_i(\lambda)$ are continuous. □

In general, $L_i(\lambda)$ and $U_i(\lambda)$ are nondifferentiable: for $h \in N$ such that $a'_h a''_h < 0$, the coefficient $\bar{a}_h(\lambda) = \lambda a'_h + (1-\lambda)a''_h = a''_h + (a'_h - a''_h)\lambda$ changes sign for $\lambda = \lambda_h = \frac{a''_h}{a''_h - a'_h}$. Suppose w.l.o.g. $a'_h - a''_h < 0$. Upon a change of sign, the term associated with x_h changes from $\bar{a}_h(\lambda)u_h$ to $\bar{a}_h(\lambda)\ell_h$, hence the derivative of $L_i(\lambda)$, i.e., the coefficient of λ, has distinct limits for $\lambda \to \lambda_h$ and is thus discontinuous:

$$\lim_{\lambda \uparrow \lambda_h} \frac{dL_i}{d\lambda} = \eta - (a'_h - a''_h)u_h; \qquad \lim_{\lambda \downarrow \lambda_h} \frac{dL_i}{d\lambda} = \eta - (a'_h - a''_h)\ell_h,$$

with $\eta = b' - b'' - \sum_{k \in P^+(\lambda_h) \setminus \{i,h\}} (a'_k - a''_k)u_k - \sum_{k \in P^-(\lambda_h) \setminus \{i,h\}} (a'_k - a''_k)\ell_k$. Hence $L_i(\lambda)$ – and analogously $U_i(\lambda)$ – is nondifferentiable.

Notice that for $\lambda \in \{0, 1\}$ we obtain the same bounds as in (4), hence we are only interested in fractional values of λ, i.e., $\lambda \in (0, 1)$ from this point on. Also, we do not consider i such that $a'_i = a''_i$ as that makes $\bar{a}_i(\lambda) = a'_i$ regardless of λ.

While $L_i(\lambda)$ and $U_i(\lambda)$ are continuous, the two rational functions in (7) may not be. However, this is not an issue. For $\bar{a}_i(\lambda) \downarrow 0$, i.e., for $\lambda \downarrow \frac{a''_i}{a''_i - a'_i}$ if $a'_i - a''_i > 0$ and $\lambda \uparrow \frac{a''_i}{a''_i - a'_i}$ if $a'_i - a''_i < 0$, three cases are possible:

- $\lim_{\bar{a}_i(\lambda) \downarrow 0} L_i(\lambda) < 0$: as $\lim_{\bar{a}_i(\lambda) \downarrow 0} \frac{L_i(\lambda)}{\bar{a}_i(\lambda)} = -\infty$, the lower bound on x_i is unchanged;
- $\lim_{\bar{a}_i(\lambda) \downarrow 0} L_i(\lambda) = 0$: because both $L_i(\lambda)$ and $\bar{a}_i(\lambda)$ are affine functions with a zero at $\frac{a''_i}{a''_i - a'_i}$, the limit $\lim_{\bar{a}_i(\lambda) \downarrow 0} \frac{L_i(\lambda)}{\bar{a}_i(\lambda)}$ is finite and equal to a (possibly tighter) bound. In addition, $\lim_{\bar{a}_i(\lambda) \downarrow 0} \frac{L_i(\lambda)}{\bar{a}_i(\lambda)}$ and $\lim_{\bar{a}_i(\lambda) \uparrow 0} \frac{L_i(\lambda)}{\bar{a}_i(\lambda)}$ might differ, but for continuity of $L_i(\lambda)$ they would still be valid — the strictest can be used as a new lower bound on x_i.
- $\lim_{\bar{a}_i(\lambda) \downarrow 0} L_i(\lambda) > 0$: the problem is infeasible, since

$$\lim_{\bar{a}_i(\lambda) \downarrow 0} \bar{a}_i(\lambda)x_i = 0 < \lim_{\bar{a}_i(\lambda) \downarrow 0} L_i(\lambda),$$

but (6) implies $\bar{a}_i(\lambda)x_i \geq L_i(\lambda)$ and (7) cannot hold.

Similar considerations hold when $\bar{a}_i(\lambda) \uparrow 0$, and the same can be easily extended to $U_i(\lambda)$.

In order to find (possibly) tighter bounds $[\bar{\ell}_i, \bar{u}_i]$ on x_i, one can solve the one-dimensional optimization problems with piecewise rational objective function (note that any local optimum provides a valid tightening):

$$\bar{\ell}_i = \max \left\{ \frac{L_i(\lambda)}{\bar{a}_i(\lambda)} : \lambda \in (0,1) \wedge \bar{a}_i(\lambda) > 0 \right\}; \tag{8}$$

$$\bar{u}_i = \min \left\{ \frac{U_i(\lambda)}{\bar{a}_i(\lambda)} : \lambda \in (0,1) \wedge \bar{a}_i(\lambda) < 0 \right\}, \tag{9}$$

and tighten the bounds on x_i to $[\bar{\ell}_i, \bar{u}_i] \cap [\ell_i, u_i]$. We are aware of no efficient method to solve these very specific problems, and we prove below that we can restrict our search to the values of $\lambda \in (0, 1)$ such that at least one of the $\bar{a}_j(\lambda)$, for $j \in N$, is zero.

Proposition 1 *All optimal solutions of the two problems (8), (9) are in the set* $\Lambda = \{\lambda \in (0,1) : (\exists j \in \{1, 2 \ldots, n\} : \bar{a}_j(\lambda) = 0)\}$.

Proof We prove this for (8) only, as the extension to (9) is straightforward. Rewrite $\bar{a}_i(\lambda) = \lambda a'_i + (1-\lambda)a''_i$ as $r\lambda + s$, where $r = a'_i - a''_i$ and $s = a''_i$, and suppose w.l.o.g. that $r < 0$. Hence $\frac{L_i(\lambda)}{\bar{a}_i(\lambda)}$ is a valid lower bound for any $\lambda \in [0, \bar{\lambda}]$, where $\bar{\lambda} = \min\{1, -\frac{s}{r}\}$.

The set $\Lambda = \{\lambda \in (0,1) : (\exists j \in \{1, 2 \dots, n\} : \bar{a}_j(\lambda) = 0)\}$ contains all *breakpoints* of the piecewise linear $L_i(\lambda)$, i.e., values of λ for which at least one $\bar{a}_i(\lambda)$ is zero. Rewrite it as $\Lambda = \{\lambda_1, \lambda_2 \dots, \lambda_p\}$ where $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_p < 1$. Upon further restricting $\lambda \leq \bar{\lambda}$ because of the sign of $\bar{a}_i(\lambda)$, we now define $\frac{L_i(\lambda)}{\bar{a}_i(\lambda)}$ using breakpoints in $\Lambda \cap [0, \bar{\lambda}] = \{\lambda_1, \lambda_2 \dots, \lambda_k\}$ with $k \leq p$.

$$\frac{L_i(\lambda)}{\bar{a}_i(\lambda)} = \begin{cases} \frac{\alpha_0\lambda+\beta_0}{r\lambda+s} & \text{if } \lambda \in (0, \lambda_1] \\ \frac{\alpha_1\lambda+\beta_1}{r\lambda+s} & \text{if } \lambda \in (\lambda_1, \lambda_2] \\ \quad\vdots & \\ \frac{\alpha_k\lambda+\beta_k}{r\lambda+s} & \text{if } \lambda \in (\lambda_k, \bar{\lambda}), \end{cases}$$

where $\alpha_h = (b' - b'') - \sum_{j \in P^+(\lambda_h)\setminus\{i\}}(a'_j - a''_j)u_j - \sum_{j \in P^-(\lambda_h)\setminus\{i\}}(a'_j - a''_j)\ell_j$ and $\beta_h = b'' - \sum_{j \in P^+(\lambda_h)\setminus\{i\}} a''_j u_j - \sum_{j \in P^-(\lambda_h)\setminus\{i\}} a''_j \ell_j$ for $h = 0, 1, 2 \dots, k$. Note that $\frac{\alpha_h\lambda+\beta_h}{r\lambda+s}$ is the ratio of two affine functions of λ, and admits no local maximum in any open interval $(\lambda_h, \lambda_{h+1})$ for $h = 0, 1, 2 \dots, k$ (we denote $\lambda_{k+1} = \bar{\lambda}$). As a consequence, all local maxima are found in Λ. A similar reasoning, whose details are omitted for simplicity, holds when $r > 0$, and the set of breakpoints is a finite subset of $[\bar{\lambda}, 1]$, with $\bar{\lambda} = \max\{0, -\frac{s}{r}\}$. The result carries over to upper bounds derived from $\frac{U_i(\lambda)}{\bar{a}_i(\lambda)}$. □

Example. Consider again the two inequalities $x_1 + x_2 + x_3 \geq 3, x_1 - x_2 + x_3 \geq 2$, with $(x_1, x_2, x_3) \in [-1, 3] \times [-1, 1] \times [0, 1]$. We have $\bar{a}_1(\lambda) = 1$, $\bar{a}_2(\lambda) = \lambda - (1-\lambda) = 2\lambda - 1$, $\bar{a}_3 = 1$, and $\bar{b} = 3\lambda + 2(1-\lambda) = 2 + \lambda$. Hence

$$\begin{aligned} L_1(\lambda) &= \bar{b} \quad - (\bar{a}_2(\lambda)w_2 \quad + \bar{a}_3(\lambda)w_3) \\ &= 2 + \lambda - ((2\lambda - 1)w_2 + w_3), \end{aligned}$$

where $w_j = u_j$ if $\bar{a}_j(\lambda) \geq 0$ and $w_j = \ell_j$ otherwise, for $j \in \{2, 3\}$. This yields

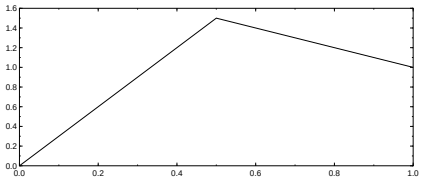
$$\frac{L_1(\lambda)}{\bar{a}_1(\lambda)} = L_1(\lambda) = \begin{cases} 3\lambda & \text{if } \lambda \in (0, \frac{1}{2}] \\ 2 - \lambda & \text{if } \lambda \in (\frac{1}{2}, 1], \end{cases}$$

which attains, for $\lambda = \frac{1}{2}$, a maximum $L_i(\frac{1}{2}) = \frac{3}{2}$, a tighter lower bound on x_1. $\frac{L_1(\lambda)}{\bar{a}_i(\lambda)}$ is depicted in Figure 1a.

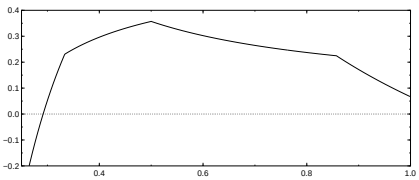
Example. Consider the two inequalities $15x_1 + 2x_2 + x_3 - 2x_4 \geq 4$ and $-x_1 - 2x_2 - 6x_3 + x_4 \geq 5$, with $(x_1, x_2, x_3, x_4) \in [-1, 3] \times [0, 2] \times [-1, 1] \times [1, 6]$. Then

$$\begin{aligned} \bar{a}_1(\lambda) &= 15\lambda - (1-\lambda) = 16\lambda - 1 \\ \bar{a}_2(\lambda) &= 2\lambda - 2(1-\lambda) = 4\lambda - 2 \\ \bar{a}_3(\lambda) &= \lambda - 6(1-\lambda) = 7\lambda - 6 \\ \bar{a}_4(\lambda) &= -2\lambda + (1-\lambda) = -3\lambda + 1 \\ \bar{b}(\lambda) &= 4\lambda + 5(1-\lambda) = -\lambda + 5, \end{aligned}$$

and $\frac{L_1(\lambda)}{\bar{a}_1(\lambda)} = \frac{5 - \lambda - ((4\lambda - 2)w_2 + (7\lambda - 6)w_3 + (1 - 3\lambda)w_4)}{16\lambda - 1}$, where $w_j = u_j$ if $\bar{a}_j(\lambda) \geq 0$ and $w_j = \ell_j$ otherwise, for $j \in \{2, 3, 4\}$. This provides a valid lower bound



(a) Maximum $\frac{L_1(\bar{\lambda})}{\bar{a}_1(\lambda)} = \frac{3}{2}$ at $\bar{\lambda} = \frac{1}{2}$.



(b) Maximum $\frac{L_1(\bar{\lambda})}{\bar{a}_1(\lambda)} = \frac{5}{14}$ at $\bar{\lambda} = \frac{1}{2}$.

Fig. 1 Plots of $\frac{L_i(\lambda)}{\bar{a}_i(\lambda)}$ for two pairs of inequalities. In (a), the two inequalities $x_1 + x_2 + x_3 \geq 3, x_1 - x_2 + x_3 \geq 2$, with $(x_1, x_2, x_3) \in [-1, 3] \times [-1, 1] \times [0, 1]$ yield a tighter lower bound on x_1. In (b), the two inequalities $15x_1 + 2x_2 + x_3 - 2x_4 \geq 4$ and $-x_1 - 2x_2 - 6x_3 + x_4 \geq 5$, with $(x_1, x_2, x_3, x_4) \in [-1, 3] \times [0, 2] \times [-1, 1] \times [1, 6]$, provide a tighter lower bound $\frac{5}{14} \approx .35714 > -1$ on x_1.

on x_1 for $\bar{a}_1(\lambda) = 16\lambda - 1 > 0$ or $\lambda \in (\frac{1}{16}, 1]$. $\frac{L_1(\lambda)}{\bar{a}_1(\lambda)}$ attains a maximum for $\bar{a}_2(\lambda) = 0$, or $\lambda = \frac{1}{2}$, and thus a tighter bound $\ell_1' = \frac{5}{14} > \ell_1 = -1$, as shown in Figure 1b.

OBBT and bounds from linear inequalities. Tightening a lower bound via OBBT, as shown in (2), consists of solving the LP problem $\min\{x_i : Ax \geq b, x \in [\ell, u]\}$. Note that the FBBT procedure shown in Section 2 is equivalent to solving a relaxation that considers a single inequality: $\ell_i' = \min\{x_i : \sum_{j \in N} a_j x_j \geq b, x \in [\ell, u]\}$. Analogously, inferring new bounds using pairs of inequalities, as shown in this section, is equivalent to solving the LP problem $\ell_i' = \min\{x_i : \sum_{j \in N} a_j' x_j \geq b', \sum_{j \in N} a_j'' x_j \geq b'', x \in [\ell, u]\}$, which is a relaxation w.r.t. the LP problem used in OBBT but, in accordance with what observed above, a restriction (thus guaranteeing a better bound) of the FBBT of Section 2.

On a related note, solving the LP problem used in OBBT is equivalent to solving the Lagrangian dual problem $\max_{\lambda \geq 0} \mathcal{L}(\lambda)$, whose objective function is

$$\mathcal{L}(\lambda) = \min_{x \in [\ell, u]} \left\{ x_i - \sum_{j \in M} \lambda_j \left(\sum_{h \in N} a_{jh} x_h - b_j \right) \right\}$$

(note that for any $\lambda \in \mathbb{R}^m_+$ $\mathcal{L}(\lambda)$ gives a valid lower bound on x_i). The equivalence is trivial as $\min\{x_i : Ax \geq b, x \in [\ell, u]\} = \max_{\lambda \geq 0} \mathcal{L}(\lambda)$. While FBBT is a restriction of the dual problem $\max_{\lambda \geq 0} \mathcal{L}(\lambda)$ where all Lagrangian multipliers but one are fixed to zero, the technique we presented above is a restriction where all but *two* multipliers are fixed to zero.

4 Separation procedure

This section presents a procedure for bound tightening based on the ideas described above. An exhaustive bound tightening should consider all and only the pairs of linear inequalities that have a potential for bound reduction, without making the generation of new bounds too computationally intensive.

Corollary 1 *The pair of inequalities (5) tightens a variable bound only if there is at least one variable x_i whose coefficients a_i' and a_i'' are nonzero and have opposite sign:*

$$\exists i \in N : a_i' a_i'' < 0. \tag{10}$$

Proof If coefficients a'_i and a''_i have the same sign for $i \in N$, the set Λ defined above is empty, and for Proposition 1 there is no bound tightening. □

Denote $N' = \{i \in N : a'_i \neq 0\}$ and similarly N''. Define $I = \{i \in N' \cap N'' : a'_i a''_i < 0\}$, $k = |I|$ and $\Lambda = \{\lambda \in (0,1) : (\exists i \in N : \bar{a}_i(\lambda) = 0)\}$. Rewrite $\Lambda = \{\lambda_1, \lambda_2 \ldots, \lambda_k\}$ where $0 < \lambda_1 \leq \lambda_2 \ldots \leq \lambda_k$, and denote $\lambda_0 = 0$.

A straightforward method for evaluating, for every $\lambda \in \Lambda$ and for every variable x_i, the potential new bounds $\frac{L_i(\lambda)}{\bar{a}_i(\lambda)}$ and $\frac{U_i(\lambda)}{\bar{a}_i(\lambda)}$ has a complexity of $O(n^3)$, because $L_i(\lambda)$ and $U_i(\lambda)$ are sums of at most n terms and hence require each $O(n)$ operations. However, both $L_i(\lambda)$ and $U_i(\lambda)$ can be computed for any λ_j incrementally from λ_{j-1} for all $j \in \{1, 2 \ldots, k\}$.

Specifically, for λ_0 we have $\bar{b}(\lambda_0) = \bar{b}(0) = b''$, $\bar{a}_i(0) = a''_i$ for $i \in N' \cup N''$, while $P^+(0) = \{i \in N : a''_i > 0\}$ and $P^-(0) = \{i \in N : a''_i < 0\}$. Hence,

$$\begin{array}{l} L_i(0) = b'' - \sum_{k \in P^+(0) \setminus \{i\}} a''_k u_k - \sum_{k \in P^-(0) \setminus \{i\}} a''_k \ell_k \\ U_i(0) = b'' - \sum_{k \in P^+(0) \setminus \{i\}} a''_k \ell_k - \sum_{k \in P^-(0) \setminus \{i\}} a''_k u_k. \end{array}$$

Also, notice that $P^+(\lambda) = P^+(0)$ for all $\lambda \in (0, \lambda_1)$, i.e., the coefficients do not change sign between 0 and λ_1. Define

$$\begin{array}{l} L' = b' - \sum_{k \in P^+(0)} a'_k u_k - \sum_{k \in P^-(0)} a'_k \ell_k \\ U' = b' - \sum_{k \in P^+(0)} a'_k \ell_k - \sum_{k \in P^-(0)} a'_k u_k, \end{array}$$

and analogously L'' and U'' using coefficients a'' and b'' in place of a' and b'. Notice that lower and upper bounds are chosen according to the sign of $\bar{a}(\lambda)$ rather than the sign of the coefficients a' and a'' of the single inequalities. Then

$$\begin{array}{l} L_i(\lambda) = \lambda L' + (1-\lambda) L'' + \bar{a}_i(\lambda) d_i; \\ U_i(\lambda) = \lambda U' + (1-\lambda) U'' + \bar{a}_i(\lambda) f_i, \end{array}$$

where $(d_i, f_i) = (u_i, \ell_i)$ if $\bar{a}_i(\lambda) > 0$ and $(d_i, f_i) = (\ell_i, u_i)$ otherwise. Suppose $\lambda_1 \in \Lambda$ is related to a variable x_h such that $\bar{a}_h(\lambda) = (a'_h - a''_h)\lambda + a''_h = r\lambda + s$ is zero for $\lambda = \lambda_1$. If $r = a'_h - a''_h > 0$, the (combined) coefficient $\bar{a}_h(\lambda)$ turns from negative to positive at $\lambda = \lambda_1$, and L', U', L'', U'' must be updated accordingly. This is easily accomplished by

$$\begin{array}{ll} L' \leftarrow L' + a'_h \delta & L'' \leftarrow L'' + a''_h \delta \\ U' \leftarrow U' - a'_h \delta & U'' \leftarrow U'' - a''_h \delta, \end{array} \tag{11}$$

where $\delta = \ell_h - u_h$. If $r < 0$, $\delta = u_h - \ell_h$. A similar update will be needed for λ_j with $j > 1$, but at each step, for continuity, $L_i(\lambda_j)$ and $U_i(\lambda_j)$ can be computed in $O(1)$. The separation procedure for a pair of inequalities is summarized in Algorithm 1. We omit the obvious rounding step for integer variables for the sake of simplicity. The core of the procedure is in lines 9 to 19. Given that the update step in line 18 has complexity $O(1)$ and that both Λ and $N' \cap N''$ have at most n elements, COMBINEINEQS has a complexity of $O(n^2)$.

To prevent numerical problems associated with the incremental update of the parameters L', U', L'', U'', large lower and upper bounds ℓ_k, u_k are not added explicitly to the expression. If an infinite upper bound u_k is considered in the update step in line 18 and makes, for instance, $L' = -\infty$, the addition is not carried out but a distinct flag is used to avoid the update of ℓ_i. We omit the technical details and refer the interested reader to the source code.

In the more general case, each constraint of the LP relaxation may have both a lower and an upper bound, one of them possibly infinite. A pair of constraints can be defined as

$$b' \leq \sum_{i \in N} a'_i x_i \leq g', \qquad b'' \leq \sum_{i \in N} a''_i x_i \leq g'',$$

Algorithm 1 COMBINEINEQS$(a', a'', b', b'', \ell, u)$

$I \leftarrow \{i \in N' \cap N'' : a'_i a''_i < 0\}$
$\Lambda \leftarrow \{\frac{a''_i}{a'_i - a''_i} : i \in I\}$
denote $\Lambda = \{\lambda_1, \lambda_2 \ldots, \lambda_k\}$ s.t. $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_k < 1$ and $k = |I|$
$L' \leftarrow b' - \sum_{k \in N'' : a''_k > 0} a'_k u_k - \sum_{k \in N'' : a''_k < 0} a'_k \ell_k$
$U' \leftarrow b' - \sum_{k \in N'' : a''_k > 0} a'_k \ell_k - \sum_{k \in N'' : a''_k < 0} a'_k u_k$
$L'' \leftarrow b'' - \sum_{k \in N'' : a''_k > 0} a''_k u_k - \sum_{k \in N'' : a''_k < 0} a''_k \ell_k$
$U'' \leftarrow b'' - \sum_{k \in N'' : a''_k > 0} a''_k \ell_k - \sum_{k \in N'' : a''_k < 0} a''_k u_k$
for $i \in N' \cup N''$: $\bar{\ell}_i \leftarrow -\infty$; $\bar{u}_i \leftarrow +\infty$
for $j \in \{1, 2 \ldots, k\}$ **do**
  for $i \in N' \cup N''$ **do**
    if $\bar{a}_i(\lambda_j) > 0$ **then**
      $\bar{\ell}_i \leftarrow \max\left\{\bar{\ell}_i, \frac{\lambda_j L' + (1-\lambda_j) L'' + \bar{a}_i(\lambda_j) u_i}{\bar{a}_i(\lambda_j)}\right\}$
    end if
    if $\bar{a}_i(\lambda_j) < 0$ **then**
      $\bar{u}_i \leftarrow \min\left\{\bar{u}_i, \frac{\lambda_j U' + (1-\lambda_j) U'' + \bar{a}_i(\lambda_j) \ell_i}{\bar{a}_i(\lambda_j)}\right\}$
    end if
  end for
  update L', U', L'', U'' using (11)
end for
return $[\bar{\ell}, \bar{u}]$

with $b', b'' \in \mathbb{R} \cup \{-\infty\}$ and $g', g'' \in \mathbb{R} \cup \{+\infty\}$. In equality constraints, $b' = g'$ and $b'' = g''$. In this more general case, multiple calls should be made, namely:

- if $b' > -\infty$ and $b'' > -\infty$, COMBINEINEQS $(a', a'', b', b'', \ell, u)$;
- if $b' > -\infty$ and $g'' < +\infty$, COMBINEINEQS $(a', -a'', b', -g'', \ell, u)$;
- if $g' < +\infty$ and $b'' > -\infty$, COMBINEINEQS $(-a', a'', -g', b'', \ell, u)$; and
- if $g' < +\infty$ and $g'' < +\infty$, COMBINEINEQS $(-a', -a'', -g', -g'', \ell, u)$.

Our implementation runs COMBINEINEQS on each suitable pair of inequalities; after each call, if tighter bounds are obtained, they are updated for use in the next call.

5 Computational tests

This section presents computational results obtained on two BB algorithms: one for MINLP and one for MILP. For both, we have implemented the procedure in the previous section, which we call TWOIMPL. The two implementations are virtually identical, as their only input is an LP relaxation available to both MILP and MINLP solvers. The BB procedure, at the root node and at every subproblem, obtains a lower bound on the objective function. This works as follows:

- solve the LP relaxation (3), where new variable bounds might have been introduced by branching rules or other bound reduction methods;
- apply cut generators to generate valid inequalities violated by the LP solution obtained.

This is repeated until either no new cuts are found, or a limit in the number of iterations or CPU time has been reached.

At the first iteration of the root node, a relaxation (3) of (1) is created. Assuming that (3) consists of m inequalities, an initial bound reduction using TWOIMPL has a complexity of $O(m^2n^2)$. In all other calls to TWOIMPL, an initial relaxation and a set of cuts (the result of other cut generators) are available. We denote the overall relaxation as $Ax \geq b$, and index the inequalities of the initial relaxation with the set $M = \{1, 2 \ldots, m\}$ and the amended inequalities with $M' = \{m+1, m+2 \ldots, m+h\}$. Then a pair $(k', k'') \in (M \cup M')^2$ of inequalities may improve a variable bound, i.e., it is worth to run COMBINEINEQS on the pair, if (10) is satisfied and one of the following holds:

- $k' \in M'$ or $k'' \in M'$, as the tightening produced by pairs of inequalities containing at least one new cut has not yet been studied;
- $k' \in M$ and $k'' \in M$, but at least one of the variables with nonzero coefficients in the inequalities k' or k'' has a tighter bound.

This does not improve the worst case complexity, which is $O((m+h)^2n^2)$, but it is useful in practice for MINLP solvers, which usually produce very sparse LP relaxations and few bounds are expected to change due to branching rules.

TWOIMPL is described in Algorithm 2, where we denote as A_i the row vector corresponding to the coefficients of the i-th inequality of $Ax \geq b$, and as $x(A_i)$ the set of variables whose coefficient is nonzero in $A_i^\top x \geq b_i$.

Algorithm 2 TWOIMPL(A, b, M, M')

```
for j ∈ M ∪ M' do
  for h ∈ M ∪ M' \ {i} do
    if M' = ∅ or i ∈ M' or h ∈ M' or
       (at least one bound of x(A_i) ∪ x(A_h) changed since previous call to TWOIMPL) then
      [ℓ, u] ← COMBINEINEQS (A_i, A_h, b_i, b_h, ℓ, u)
    end if
  end for
end for
```

TWOIMPL requires a substantial portion of the total CPU time required or allotted to solve a MINLP/MILP problem. Therefore, it is best used only when it reduces variable bounds. Although we cannot predict whether or not this procedure will be useful, one or more ineffective calls justify an early exclusion of the procedure. To this purpose, we use four parameters (f, n, d, s) to control the number of runs of TWOIMPL:

- f: a *frequency* parameter. TWOIMPL is run every f BB nodes. If $f = 1$, it is run at every node; if $f = k$, with $k > 1$, it is run at every k-th node. If $f = -k$, with $k > 0$, it is run at the root node and, if at least one bound is tightened at the root node, it is run at every k-th of the subsequent nodes, otherwise it is never called again.
- n: the maximum number of times TWOIMPL is run at each node. If no bounds are tightened at the i-th iteration, with $i < n$, the procedure is stopped.
- d: the depth of the BB tree after which the frequency of calls to TWOIMPL decreases. If node i is at depth l of the BB tree, TWOIMPL is run (given that the conditions on the other parameters are satisfied) with probability equal to 1 if $l \leq d$, and $\frac{1}{1+l-d}$ otherwise.
- s: the depth of the BB tree after which TWOIMPL is no longer run.

Parameters f, d, and s are used together, i.e., a BB node must satisfy the conditions on all three parameters in order for TWOIMPL to be run. For example, if $f = 3$, $d = 5$, and $s = 8$,

TWOIMPL is run at node 9 at depth 4; it is run with probability $\frac{1}{2}$ at node 18 at depth 6; it is not run at node 2 at depth 1; and it is not run at node 24 at depth 9.

We have used five variants, each with a time limit of two hours. Each variant is identified by a 4-tuple of parameter values (note that the parameter f is negative in all cases, which implies that TWOIMPL is only used throughout the BB if successful at the root node):

plain: TWOIMPL is never run;

twoRepeat: $(f, n, d, s) = (-1, 10, 5, 20)$. Although its use is limited to nodes with depths of 20 or smaller, at each node TWOIMPL is run up to ten times. The main purpose of this test is to check the utility of multiple calls early in the development of the BB tree.

twoStandard: $(f, n, d, s) = (-1, 2, 5, 20)$. This is a standard parameter setting as it limits the application of TWOIMPL to BB nodes with small depths, thus especially suited for mid-size problems.

twoMinimal: $(f, n, d, s) = (-1, 1, 1, 5)$. This a lighter version where TWOIMPL is allotted much fewer resources (it may be called in not more than $2^5 - 1 = 31$ nodes), and is targeted at large MINLP/MILP problems.

twoSparse: $(f, n, d, s) = (-10, 1, 10, +\infty)$. This version allows execution of TWOIMPL at any node of the BB tree irrespective of its depth, but only one every ten nodes can be tested and with probability decreasing with depth.

We aimed at diversifying the experimental setup, hence we have generated a heterogeneous set of variants. It is obviously hard to identify a variant of TWOIMPL that performs reasonably well on most practical MINLP and MILP instances, and even tuning all parameters to a single instance is not trivial. We have opted for a number of variants with a reasonable trade-off between completeness and clarity, although we keep $f < 0$ in all variants in order to make the root node decisive in deciding whether TWOIMPL should be used throughout the BB or not. Limited preliminary experiments, in fact, showed mild correlation in actual bound reduction between the root node and the remaining ones.

5.1 Tests on MINLP problems

We have implemented TWOIMPL in Couenne [3], an Open-Source branch-and-bound solver for MINLP that is available in the COIN-OR repository (http://www.coin-or.org), and is distributed under the Eclipse Public License [10]. Installation and usage instructions are available at http://www.coin-or.org/Couenne. Couenne employs linearization, various bound reduction methods (including FBBT and OBBT), a simple rounding heuristic for finding feasible solutions, and a *reliability branching* scheme [5]. Couenne obtains a lower bound on the optimal solution of the MINLP (1) by reformulating the problem as outlined in Section 2 and described in more detail in [15, 18, 26], and generating an LP relaxation (3). At each node of the BB, various cut generation procedures are used: up to four rounds of cuts to refine the linear relaxation and, after each round of linearization, a round of TWOIMPL, to ensure that newly generated linearization cuts can be used for bound tightening. The BB algorithm applies a *best bound* policy, i.e., the BB node with lowest lower bound is selected at every iteration. All tests were run using the *trunk* version of Couenne[1].

Couenne relies on other programs, some of them also part of the COIN-OR repository: a local minimum for the LP relaxations is found by IPOPT [28], while the BB is implemented using routines of CBC [8], and the LP solver to solve relaxations is CLP [12]. The generator

[1] The source code with the implementation of the procedure described in this paper is available at https://projects.coin-or.org/Couenne/browser/trunk/Couenne/src/two_implied_bt

of linearization inequalities is a specialization of the CGL library for cut generation [17]. Couenne also uses software not from the COIN-OR framework: BLAS, LAPACK, routines `ma27` and `mc19` from HSL, and the AMPL Solver Library (ASL).

We have conducted tests on several MINLP instances from the online libraries GLOBALLIB[2], MINLPLIB[3], and MACMINLP[4]. There are 791 instances in total, but we present the results on a subset of them, excluding those for which

- none of the variants took more than one minute (these are easily solved instances, and we are not interested in solving these more efficiently);
- no substantial difference is observed in the CPU time, lower bound, or number of BB nodes. This may happen because of how TWOIMPL is employed: in fact, it may be excluded from Couenne after a few runs if it proves ineffective, and as a consequence the result is similar to that of the **plain** version.

We have performed a set of tests using the Palmetto cluster at Clemson University, with several machines equipped with different processors and RAM memory. Although two instances may have been solved on different machines, all five variants of Couenne used the same machine for each instance.

Tables 2, 3, and 4 below summarize the results on MINLP instances, whereas Table 1 shows results on Nonlinear Programming (NLP) problems, with continuous variables only. The first four columns are a description of the instance (name, number of variables, integer variables, and constraints). For each of the five variants, we report the CPU time and the number of BB nodes; if no optimal solution was found after the time limit, we report the lower bound (in brackets) and the gap w.r.t. the best integer solution found by any of the five variants (or a "–" if no such upper bound is found). For the four variants using TWOIMPL, we report the number n_{bt} of calls to TWOIMPL that tightened at least one bound and the total time t_{bt} spent by TWOIMPL. The best performances are highlighted in bold when differing significantly. Unfortunately, for some of the instances, either numerical problems or unstable components of Couenne terminated the Branch-and-bound without collecting performance data. For those instances, we add a "–" in the tables.

Out of the 791 MINLP instances approached, only 218 survived the exclusion criteria outlined above: for 69 NLP and 33 MINLP instances (out of 448 and 343, respectively), all variants had very similar performances, while the remaining 471 instances were solved by all variants in less than a minute. The tables show that at least one variant of Couenne with TWOIMPL has the best performance in a significant percentage of the instances. However, the CPU time used by TWOIMPL is sometimes a large portion of the available time limit, although for some cases it does not result in an improved overall performance. The two versions of TWOIMPL that appear to have an overall better result are **twoStandard** and **twoMinimal**, suggesting that, while TWOIMPL should not be applied at all BB nodes (unlike **twoSparse**), it is more beneficial at nodes with limited depth, and that one iteration (as opposed to the ten iterations of **twoRepeat**) is sufficient on average.

This variability in the performance of TWOIMPL is apparent even within the same class of instances. The first rows of Table 3 contain convex MINLP instances (i.e., MINLP problems whose continuous relaxation is a convex NLP). For some instances, TWOIMPL may help Couenne find a tighter lower bound, but for others it simply occupies CPU resources without improving any bound.

[2] http://www.gamsworld.org/global/globallib.htm
[3] http://www.gamsworld.org/minlp/minlplib.htm
[4] http://wiki.mcs.anl.gov/leyffer/index.php/MacMINLP

Preliminary experiments showed that applying TWOIMPL to all BB nodes was detrimental to Couenne, as for several instances it had no positive effect. Introducing the control layer based on the aforementioned parameters (f, n, d, s) has improved dramatically the impact of TWOIMPL. This suggests that the utility of this bound tightening procedure is very dependent on early detection of its impact (or lack thereof), and a more sophisticated procedure – for example, excluding variables whose bounds are seldom tightened – would be helpful in this regard, especially because of the CPU time invested in it.

5.2 Tests on MILP problems

Although we had MINLP problems in mind while developing the first version of TWOIMPL, we have tested it on MILP problems as well. We have added a version of TWOIMPL to the Open-Source MILP solver Cbc [11] and tested 94 instances from the MIPLIB3[5] and MIPLIB2003[6] online libraries. Out of 94 instances, 47 were solved in under a minute by all variants, and a similar performance of all variants was observed for another 32. This leaves 15 instances. Table 5, with the same format as Tables 1-4, summarizes the results for these.

The results appear less favorable for TWOIMPL than they do in the MINLP case. The most important cause for worse performance is clear from instances *air04*, *air05*, *mod011*, *mzzv11*, *mzzv42z*, and *nw04*, which are also the largest instances: although the procedure did tighten bounds, this has no effect on the CPU time of the branch-and-bound, even though the time spent in the bound tightening procedure is of the same order of magnitude as that of the BB. For some of the instances which the **plain** variant cannot solve in two hours, TWOIMPL does improve the lower bound and sometimes (with *roll3000*) closes the gap.

Although extending the application of TWOIMPL to MILP is straightforward, we should remember that LP relaxations (3) used to find lower bounds of MINLP problems such as (1) have a very peculiar structure that derives from the lower bounding technique. In general, one cannot expect MINLP problems to be sparser than MILP problems, although lower bounding techniques for MINLP, in general, do create sparse LP relaxations. Whether a problem has an LP relaxation with a sparse or a dense coefficient matrix has different consequences on TWOIMPL: in a sparser problem, fewer terms are involved in the computation for each pair of constraints, and hence TWOIMPL may take less CPU time.

Another explanation of why TWOIMPL seems less useful for MILP than it is for MINLP is that modern state-of-the-art MILP software can solve far larger problems than MINLP problems. The complexity $O(n^2m^2)$ of our procedure clearly makes it less appealing, and even less so given that bound tightening is not as important in MILP as it is for MINLP problems. On a related note, the variant that seems to be more successful among those using TWOIMPL is *twoMinimal*. This suggests that for MILP problems it is worth running TWOIMPL only for a few nodes at the beginning, in agreement with the scaling argument.

6 Concluding remarks and open questions

We have presented a bound reduction technique for MILP and MINLP. An implementation for MINLP is readily available for download in the COIN-OR repository, and that for MILP is available upon request. The procedure is beneficial to MINLP solvers, and partially to

[5] http://www.caam.rice.edu/~bixby/miplib/miplib3.html
[6] http://miplib.zib.de

MILP solvers, for a number of difficult instances, albeit the significant CPU time spent sometimes makes it counterproductive. This could be improved in different ways:

– with a parallel implementation. Since only a portion of the algorithm cannot be decomposed (namely lines 9-19 of COMBINEINEQS), if $O(m^2)$ processors were available the procedure would have a complexity of $O(n^2)$. This can be further improved to $O(n)$ by observing that the inner loop of lines 10-18 can also be parallelized.
– conditions might be developed to avoid tests between certain pairs of inequalities that provably cannot yield tightened bounds.

Extending this procedure to obtain bounds using more than two inequalities seems a much more challenging task: in fact, choosing three multipliers (i.e., two degrees of freedom) does away with the relatively simple one-dimensional optimization problem discussed here.

7 Acknowledgments

The author is grateful to Pierre Bonami, Jeff Linderoth, and François Margot for useful discussions in a preliminary phase of this work, and to two anonymous referees for insightful comments.

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. E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Mathematical Programming*, 71:221–245, 1995.
3. P. Belotti. COUENNE: a user's manual. Technical report, Lehigh University, 2009.
4. P. Belotti, S. Cafieri, J. Lee, and L. Liberti. Feasibility-based bounds tightening via fixed points. In W. Wu and O. Daescu, editors, *Combinatorial Optimization and Applications*, volume 6508 of *Lecture Notes in Computer Science*, pages 65–76. Springer Berlin / Heidelberg, 2010.
5. P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4-5):597–634, 2009.
6. 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.
7. A. Caprara and M. Locatelli. Global optimization problems and domain reduction strategies. *Mathematical Programming*, 125:123–137, 2010.
8. CBC-2.1. Available from `https://projects.coin-or.org/cbc`.
9. E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331, 1987.
10. Eclipse public license version 1.0. `http://www.eclipse.org/legal/epl-v10.html`.
11. J. Forrest. CBC, 2004. Available from `http://www.coin-or.org/Cbc`.
12. J. Forrest. CLP: COIN-OR linear program solver. http://www.coin-or.org/Clp/index.html, 2006.
13. E. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker, Inc., New York, 1992.

14. A. H. Land and A. G. Doig. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3):497–520, 1960.
15. L. Liberti. Reformulations in mathematical programming: Definitions and systematics. *RAIRO-RO*, 43(1):55–85, 2009.
16. Y. Lin and L. Schrage. The global solver in the LINDO API. *Optimization methods and software*, 24(4):657–668, 2009.
17. R. Lougee-Heimer. Cut generation library. `http://projects.coin-or.org/Cgl`, 2006.
18. G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. *Mathematical Programming*, 10:146–175, 1976.
19. F. Messine. Deterministic global optimization using interval constraint propagation techniques. *RAIRO-RO*, 38(4):277–294, 2004.
20. A. Qualizza, P. Belotti, and F. Margot. Linear programming relaxations of quadratically constrained quadratic programs. In *IMA Volume Series*. Springer, 2011. To appear.
21. I. Quesada and I. E. Grossmann. Global optimization of bilinear process networks and multicomponent flows. *Computers & Chemical Engineering*, 19(12):1219–1242, 1995.
22. 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.
23. H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, Mar. 1996.
24. N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.
25. M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.
26. E. M. B. Smith and C. C. Pantelides. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Computers & Chem. Eng.*, 23:457–478, 1999.
27. 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.
28. 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.

			plain		twoRepeat				twoStandard				twoMinimal				twoSparse			
Name	var	con	t(lb)	nd(gp)	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}
bayes2_10	86	72	5800	90k	**373**	97k	92	3.1	1118	309k	106	0.9	1078	312k	10	0.0	1958	565k	2444	12.9
bayes2_20	86	74	(0)	0.0%	(0)	0.0%	57	2.6	**6353**	1499k	34	0.3	(0)	0.0%	8	0.0	(0)	0.0%	162	1.0
bayes2_30	86	75	6442	1639k	(0)	0.0%	204	8.5	(0)	0.0%	98	1.0	(0)	0.0%	17	0.1	**4961**	57k	405	2.3
bayes2_50	86	76	(0)	52.0%	(0)	52.0%	114	7.1	–	–	–	–	**5765**	2624	9	0.1	–	–	–	–
chenery	43	38	151	96k	60	34k	19	0.1	59	34k	19	0.1	**35**	14k	5	0.0	38	18k	72	0.1
ex1243-cont	57	75	**(36369)**	1.9%	(36279)	2.2%	934	6.2	(36267)	2.2%	948	5.0	(36230)	2.3%	8	0.0	(36249)	2.2%	14k	45.0
ex14_1_7	10	17	–	–	**21**	3651	25	0.4	111	10k	26	2.0	1624	43k	5	0.0	(0)	0.0%	484	3526.3
ex5_2_5	32	19	(-4981.49)	29.7%	(-4576.33)	23.5%	2496	73.6	**(-4287.39)**	18.4%	3239	37.8	(-4689.69)	25.4%	23	0.1	(-4578.26)	23.5%	15k	102.7
ex5_3_3	62	53	(2.035)	39.5%	(2.0945)	36.8%	51	0.5	**(2.109)**	36.2%	42	0.2	(2.074)	37.8%	7	0.0	(1.971)	42.5%	5590	22.4
ex5_4_4	27	19	(7171.75)	40.5%	(6875.75)	46.6%	19k	67.6	(6424.35)	56.9%	13k	30.8	**1493**	684k	54	0.0	(8872.82)	13.6%	138k	155.6
ex6_1_1	8	6	92	15k	84	15k	207	3.3	83	16k	274	1.6	**62**	12k	32	0.1	78	15k	590	2.7
ex6_1_3	12	9	241	28k	237	28k	–	0.0	243	28k	–	0.0	227	28k	–	0.0	225	28k	2	0.1
ex6_2_11	3	1	254	30k	289	29k	2121	66.2	271	29k	2080	52.7	271	32k	65	0.4	287	25k	6068	90.4
ex6_2_12	4	2	**169**	16k	232	14k	2390	84.3	187	14k	2367	47.1	233	18k	89	0.4	256	16k	6102	81.8
ex6_2_9	4	2	696	31k	676	26k	1123	53.1	**523**	26k	1178	35.1	601	25k	114	0.5	640	24k	2174	46.6
ex7_2_3	8	6	(2624.65)	162.1%	(2215.16)	210.5%	184	0.3	(2208.28)	211.5%	178	0.2	**1978**	2452k	47	0.0	(2196.85)	213.1%	18k	23.8
ex8_4_2	24	10	(0.262)	17.6%	**(0.266)**	17.4%	4263	68.2	(0.265)	17.4%	4337	55.7	(0.261)	17.7%	6	0.1	(0.214)	22.4%	20k	234.8
ex8_4_7	62	40	**232**	2537	995	6k	17	0.4	991	6k	17	0.4	459	2996	2	0.0	1214	7k	84	1.3
ex8_4_8_bnd	42	30	5513	21k	**1321**	5k	48	8.6	1488	6k	43	14.0	4547	19k	6	0.0	2930	10k	226	372.7
ex8_4_8	42	30	**1562**	6k	4069	14k	45	78.5	2135	8k	52	9.9	3838	13k	19	0.3	(3.038)	6.8%	520	3041.5
ex8_5_1	6	4	81	27k	62	19k	409	21.5	56	20k	360	11.6	**49**	21k	13	0.0	84	20k	1472	40.8
ex8_5_2	6	4	**345**	71k	4550	306k	2425	618.8	4817	297k	2585	712.7	3650	289k	28	0.0	(-1.78e-3)	0.0%	17k	2734.3
ex8_5_3	5	4	75	18k	76	8k	657	54.8	479	17k	1013	378.1	10	5k	37	0.1	8	2289	1077	4.8
ex8_5_6	6	4	375	48k	268	35k	496	41.8	–	–	–	–	579	62k	32	0.0	755	50k	3432	320.6
hhfair	27	25	87	21k	**9**	3454	28	0.1	**9**	3454	28	0.0	**9**	3754	8	0.0	**9**	3530	133	0.0
least	3	0	3	612	(0)	0.0%	4	0.0	3	398	5	0.0	(0)	0.0%	4	0.0	3	636	18	0.0
polygon25	48	323	(-5.327)	71.9%	**(-1.571)**	30.9%	1	0.8	**(-1.571)**	30.9%	1	0.8	**(-1.571)**	30.9%	1	0.4	**(-1.571)**	30.9%	5	2.4
qp1	50	2	(-0.244)	19.7%	**(-0.161)**	14.0%	19	51.7	**(-0.161)**	14.0%	19	51.7	(-0.240)	19.5%	4	5.7	**(-0.162)**	14.1%	362	594.9
qp2	50	2	**(-0.104)**	9.5%	(-0.182)	15.5%	32	59.7	(-0.182)	15.5%	32	59.8	(-0.153)	13.4%	5	7.9	(-0.185)	15.8%	400	622.6
qp3	100	52	(-0.253)	20.3%	(-0.173)	14.9%	54	48.3	(-0.173)	14.9%	54	34.6	(-0.193)	16.3%	13	4.3	**(-0.145)**	12.8%	5994	2206.1

Table 1 Comparison between five versions of Couenne, one without and four with TWOIMPL. We report: number of variables and constraints of each instance (var, con); CPU time (t) in seconds and number of branch-and-bound nodes (nd) if the variant solved the instance within the two hour time limit, otherwise the lower bound (lb) in brackets and the optimality gap (gp) measured w.r.t. the best integer solution found by any of the five variants. For the four variants using TWOIMPL, we also report the number of iterations of TWOIMPL that reduced at least one bound (n_{bt}) and the total time spent within the procedure (t_{bt}) in seconds. These instances are from the `globallib` instance library, and have no integer variables.

				plain		twoRepeat				twoStandard				twoMinimal				twoSparse			
Name	var	int	con	t(lb)	nd(gp)	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}
conv																					
RSyn0805H	720	296	1886	5350	13k	5068	13k	117	19.7	5704	13k	117	18.2	5504	14k	7	0.9	5360	13k	586	54.0
RSyn0805M02M	360	148	769	1111	9k	1106	9k	107	4.6	1213	9k	107	4.4	1163	8k	17	0.4	1125	9k	224	5.4
RSyn0805M03M	540	222	1284	3927	16k	3859	16k	207	23.2	3854	16k	207	21.5	4144	17k	16	0.8	3750	15k	940	100.3
RSyn0805M04M	720	296	1886	5277	13k	4818	13k	117	18.5	4949	13k	117	17.2	5276	14k	7	0.8	4962	13k	586	50.9
RSyn0810H	820	336	2140	(-9052.6)	27.3%	(-8895.1)	26.0%	279	52.0	**(-8878.9)**	25.9%	287	49.7	**(-8872.8)**	25.8%	20	2.5	**(-8878.9)**	25.9%	851	102.0
RSyn0810M03M	615	252	1452	(-4060.6)	32.9%	(-4070.4)	33.1%	486	51.4	(-4070.5)	33.1%	481	48.9	(-4070.5)	33.1%	15	1.4	(-4070.5)	33.1%	1453	114.0
RSyn0810M04M	820	336	2140	(-9061.1)	27.4%	(-8878.9)	25.9%	287	53.1	(-8878.9)	25.9%	286	49.6	**(-8867.0)**	25.8%	20	2.4	(-8878.9)	25.9%	849	101.5
RSyn0815H	940	376	2430	(-6066.7)	44.8%	**(-6004.6)**	44.2%	89	24.4	**(-6004.6)**	44.2%	94	18.6	(-6025.9)	44.4%	8	1.0	(-6043.6)	44.6%	323	37.4
RSyn0815M02M	470	188	981	(-2397.0)	26.5%	(-2389.6)	26.3%	443	42.4	**(-2384.6)**	26.1%	399	24.6	**(-2385.2)**	26.1%	17	0.4	(-2395.6)	26.5%	1971	97.8
RSyn0815M03M	705	282	1647	(-4121.6)	31.7%	(-4052.3)	30.5%	259	42.4	**(-4049.0)**	30.5%	239	28.0	**(-4046.1)**	30.4%	18	1.2	(-4056.8)	30.6%	1020	76.9
RSyn0815M04M	940	376	2430	(-6064.9)	44.8%	(-6027.4)	44.4%	87	24.0	**(-6025.9)**	44.4%	94	18.7	(-6027.4)	44.4%	8	1.0	(-6047.0)	44.6%	320	36.9
Syn15M03H	453	90	768	**287**	0	458	0	1	0.0	451	0	1	0.0	447	0	1	0.0	444	0	1	0.0
Syn15M03M	255	90	537	80	536	75	534	119	5.7	72	534	119	3.3	73	534	14	0.2	83	534	470	7.4
Syn15M04H	604	120	1114	182	116	**55**	2	3	0.3	**55**	2	3	0.1	4268	0	1	0.0	4218	0	1	0.0
Syn15M04M	340	120	806	299	1767	325	1763	289	23.0	314	1761	267	12.7	301	1761	9	0.3	323	1765	893	23.2
mac/conv																					
c-sched2	400	308	137	(-2.69e6)	94.2%	(-5.12e6)	96.9%	16	34.4	**(-2.56e6)**	93.9%	34	28.7	–	–	–	–	(-2.65e6)	94.1%	180	141.0
trimloss12	800	644	372	(1.765)	0.0%	(4.069)	0.0%	9	5.1	(4.070)	0.0%	9	3.8	(3.277)	0.0%	1	0.6	**(4.961)**	0.0%	13	3.3
trimloss4	105	85	64	2366	17k	**2117**	9k	153	1.9	4696	24k	149	1.6	5015	96k	12	0.1	2855	11k	589	3.7
trimloss5	161	131	90	(5.692)	–	(5.918)	–	65	1.3	(5.918)	–	65	1.2	**(6.1)**	–	4	0.0	–	–	–	–
trimloss6	345	289	154	(4.184)	–	**(4.633)**	–	15	1.1	**(4.634)**	–	15	1.0	(4.4)	–	5	0.2	(4.6)	–	39	1.3
trimloss7	345	289	154	(4.184)	–	**(4.633)**	–	15	1.1	(4.6)	–	15	1.0	(4.3)	–	5	0.1	(4.6)	–	39	1.3
mac/nconv																					
space-25	893	750	235	(89.681)	–	(93.116)	–	16	0.4	(93.281)	–	14	0.2	**(94.432)**	–	4	0.0	–	–	–	–
space-960-ir	3617	960	3617	(6.49e6)	–	(6.49e6)	–	1	9.3	(6.49e6)	–	1	9.3	(6.49e6)	–	1	5.5	–	–	–	–
trimlon4	24	24	24	(5.895)	34.9%	**4**	1586	14	0.0	**4**	1586	14	0.0	(8.236)	0.7%	3	0.0	(7.078)	15.1%	35	0.0
trimlon5	35	35	30	(7.357)	35.2%	**1381**	833k	18	0.0	**1357**	833k	18	0.0	1595	894k	7	0.0	(9.513)	7.5%	1360	3.0
trimlon7	63	63	42	(6.267)	129.8%	**(7.212)**	103.3%	6	0.0	**(7.213)**	103.3%	6	0.0	(6.593)	119.9%	3	0.0	–	–	–	–
pooling																					
example4	26	0	35	17	6k	12	2617	85	3.9	13	3600	75	1.0	168	55k	18	0.1	7	1616	101	0.8
foulds3	168	0	48	**(-71.274)**	87.5%	(-74.747)	88.1%	26	2.8	(-74.747)	88.1%	26	2.2	(-74.240)	88.0%	6	0.3	(-73.908)	88.0%	171	7.8

Table 2 Comparison between five versions of Couenne, one without and four with TwoImpl. We report: number of variables, integer variables, and constraints of each instance (var, int, con); CPU time (t) in seconds and number of branch-and-bound nodes (nd) if the variant solved the instance within the two hour time limit, otherwise the lower bound (lb) in brackets and the optimality gap (gp) measured w.r.t. the best integer solution found by any of the five variants. For the four variants using TwoImpl, we also report the number of iterations of TwoImpl that reduced at least one bound (n_{bt}) and the total time spent within the procedure (t_{bt}) in seconds. These instances are from multiple sources.

				plain		twoRepeat				twoStandard				twoMinimal				twoSparse			
Name	var	int	con	t(lb)	nd(gp)	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}
du-opt5	18	11	6	959	1308	**651**	1009	4	2.8	**650**	1009	4	1.7	1384	2471	1	0.4	877	1528	11	2.5
du-opt	20	13	8	**2407**	11k	(3.556)	0.0%	5	3.7	(3.556)	0.0%	5	2.6	(3.556)	0.0%	1	0.5	(3.556)	0.0%	214	51.4
fo8_ar4_1	144	56	347	(16.365)	47.7%	**(18.399)**	32.2%	2626	210.9	**(18.392)**	32.3%	2604	150.3	(17.908)	35.7%	28	1.0	(16.236)	48.8%	49k	1569.4
fo7_2	112	42	211	**(17.726)**	0.1%	(3.703)	298.7%	4298	224.3	(3.706)	298.4%	4304	136.5	(9.537)	77.9%	24	0.5	(3.264)	339.7%	13k	223.3
fo7_ar2_1	112	42	269	(20.737)	18.9%	(22.080)	12.0%	18k	1123.5	**(22.202)**	11.4%	19k	815.6	(19.595)	25.5%	31	0.7	(20.324)	21.2%	63k	1444.9
fo7_ar25_1	112	42	269	(18.993)	20.7%	(19.143)	19.8%	8632	482.2	(19.160)	19.7%	8588	358.4	(18.871)	21.4%	31	0.7	**(21.760)**	6.0%	60k	1362.6
fo7_ar3_1	112	42	269	(18.48)	20.7%	(18.512)	20.5%	5861	371.8	(18.418)	21.1%	5706	241.2	(16.430)	34.9%	30	0.7	**(21.495)**	4.5%	16k	405.4
fo7_ar4_1	112	42	269	(18.561)	11.1%	(18.682)	10.4%	6763	483.3	(18.652)	10.6%	6713	292.3	**6445**	1483k	32	0.8	(17.993)	14.4%	66k	1607.3
fo7_ar5_1	112	42	269	(11.918)	45.1%	(13.941)	25.5%	544	31.2	**1530**	381k	4604	194.5	(15.337)	14.8%	25	0.7	2597	566k	24k	596.4
fo7	112	42	211	**(10.533)**	114.3%	(4.097)	384.8%	7678	361.8	(3.997)	394.6%	8557	266.4	(4.329)	363.7%	15	0.3	(7.223)	200.5%	88k	1524.5
fo8_ar2_1	144	56	347	**(23.792)**	26.4%	(20.283)	47.3%	15k	1194.8	(20.395)	46.5%	15k	931.4	(19.548)	52.5%	38	1.3	(22.669)	32.4%	59k	1971.8
fo8_ar25_1	144	56	347	(20.566)	46.8%	(20.087)	50.2%	12k	1064.5	**(20.720)**	45.8%	14k	904.0	(18.238)	64.6%	31	1.1	–	–	–	–
fo8_ar3_1	144	56	347	(19.862)	28.3%	(21.221)	20.5%	2167	202.2	(21.225)	20.4%	2177	140.0	(20.032)	27.3%	19	0.7	**(22.064)**	16.1%	11k	388.3
fo8_ar5_1	144	56	347	(15.688)	98.4%	(10.486)	188.3%	885	70.5	(10.486)	188.3%	885	54.5	**(16.806)**	86.0%	21	0.8	(11.010)	175.7%	8796	307.6
fo8	144	56	273	(4.802)	380.4%	(2.133)	789.5%	11k	623.4	(2.185)	775.0%	12k	521.6	**(6.038)**	296.0%	27	0.8	(3.368)	538.1%	69k	1793.9
fo9_ar2_1	180	72	435	(22.884)	83.6%	**(24.602)**	71.3%	2001	183.7	**(24.601)**	71.3%	1998	167.6	(24.345)	73.0%	27	1.3	(23.860)	76.4%	35k	1663.5
fo9_ar25_1	180	72	435	**(21.808)**	133.0%	(16.332)	206.6%	452	39.6	(16.332)	206.6%	454	37.7	(17.788)	182.8%	32	1.5	(17.803)	182.6%	7275	356.1
fo9_ar3_1	180	72	435	**(20.032)**	178.0%	(19.333)	187.6%	3464	372.7	(19.351)	187.3%	3463	298.8	**(20.032)**	178.0%	21	1.1	–	–	–	–
fo9	180	72	343	(2.584)	1355.4%	(2)	1638.9%	3649	303.4	(2)	1638.9%	4013	261.8	**(5.75)**	672.9%	21	1.1	–	–	–	–
gasnet	86	10	67	(3.357e6)	108.5%	(3.406e6)	105.5%	119	0.8	(3.436e6)	103.7%	123	0.6	(3.484e6)	100.9%	24	0.1	**(3.512e6)**	99.3%	739	2.9
m7_ar2_1	112	42	269	**221**	76k	3538	1185k	11k	578.9	3406	1185k	11k	454.5	1663	667k	21	0.5	993	330k	7623	159.9
m7_ar3_1	112	42	269	897	285k	**92**	18k	266	14.4	**89**	18k	266	11.2	349	96k	29	0.7	148	21k	2483	59.3
m7_ar4_1	112	42	269	959	260k	266	46k	1498	79.6	454	101k	2292	87.1	**26**	3615	20	0.5	292	48k	4021	94.2
m7_ar5_1	112	42	269	1528	447k	934	250k	2664	151.8	894	254k	2676	105.2	**505**	144k	24	0.6	1262	304k	12k	279.4
m7	112	42	211	485	199k	236	92k	478	23.0	204	90k	462	13.2	**118**	47k	29	0.6	210	74k	1215	20.5
minlphix	37	7	38	$(-\infty)$	–	13	50	10	0.0	**1**	30	13	0.0	13	50	5	0.0	**1**	40	16	0.0
no7_ar2_1	112	42	269	(92.367)	16.5%	3906	635k	12k	850.4	**3800**	644k	12k	607.6	(85.399)	25.9%	17	0.5	5582	894k	39k	1126.8
no7_ar25_1	112	42	269	(82.171)	34.8%	(82.953)	33.6%	6966	474.0	(83.188)	33.2%	7444	343.0	**(104.758)**	6.0%	23	0.7	(69.598)	58.8%	52k	1283.6
no7_ar3_1	112	42	269	**(81.070)**	37.6%	(76.54)	45.6%	2279	165.5	(76.720)	45.3%	2285	103.7	(69.749)	59.6%	20	0.6	(74.602)	49.3%	11k	282.4
no7_ar4_1	112	42	269	(55.436)	90.5%	(55.383)	90.6%	278	19.8	(55.383)	90.6%	279	12.9	**(76.137)**	39.3%	22	0.7	–	–	–	–
no7_ar5_1	112	42	269	**(73.313)**	23.5%	(61.807)	46.2%	5210	400.3	(61.906)	45.9%	5288	260.1	(60.654)	48.9%	18	0.6	(61.728)	46.3%	40k	1058.5
nous1	48	2	41	(1.149)	22.3%	(1.1802)	20.6%	103	0.9	**(1.398)**	9.6%	372	1.6	(1.235)	17.6%	12	0.0	**(1.396)**	9.7%	8241	30.5

Table 3 Comparison between five versions of Couenne, one without and four with TwoImpl. We report: number of variables, integer variables, and constraints of each instance (var, int, con); CPU time (t) in seconds and number of branch-and-bound nodes (nd) if the variant solved the instance within the two hour time limit, otherwise the lower bound (lb) in brackets and the optimality gap (gp) measured w.r.t. the best integer solution found by any of the five variants. For the four variants using TwoImpl, we also report the number of iterations of TwoImpl that reduced at least one bound (n_{bt}) and the total time spent within the procedure (t_{bt}) in seconds. These instances are from the `minlplib` instance library.

				plain		twoRepeat				twoStandard				twoMinimal				twoSparse			
Name	var	int	con	t(lb)	nd(gp)	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}
nvs17	7	7	7	**22**	18k	(-1102.23)	0.2%	40	0.7	(-1102.23)	0.2%	40	0.7	**26**	5k	5	0.0	**32**	3051	134	1.5
nvs23	9	9	9	1433	628k	1692	746k	14	0.6	1644	746k	14	0.6	3976	1873k	5	0.1	**823**	355k	1912	45.4
nvs24	10	10	10	(-1791.2)	42.3%	(-1188.61)	13.1%	12	0.7	(-1189.37)	13.1%	12	0.7	(-1106.94)	6.7%	7	0.2	**5059**	1627k	5602	212.4
o7_2	112	42	211	(53.975)	120.9%	**(67.381)**	77.6%	3333	191.7	**(67.7163)**	76.7%	3376	112.6	(49.795)	139.0%	21	0.5	(56.416)	111.5%	17040	314.1
o7_ar2_1	112	42	269	**(114.388)**	22.6%	(104.495)	34.0%	1384	80.8	(104.495)	34.0%	1393	62.0	(104.914)	33.5%	29	0.8	–	–	–	–
o7_ar25_1	112	42	269	(106.138)	32.0%	(106.996)	30.9%	8151	584.8	**(107.368)**	30.5%	8624	381.2	(95.547)	46.5%	30	0.8	(99.720)	40.4%	59024	1473.8
o7_ar3_1	112	42	269	(89.810)	53.3%	(98.098)	40.5%	3137	231.9	(99.565)	38.4%	4377	189.3	**(102.007)**	35.1%	25	0.7	(94.306)	46.1%	12165	298.5
o7_ar4_1	112	42	269	**(90.24)**	52.4%	(79.025)	73.7%	12k	912.0	(79.208)	73.3%	13k	589.8	(72.076)	90.2%	20	0.7	(89.811)	53.1%	57892	1508.5
o7_ar5_1	112	42	269	**(89.380)**	34.3%	(77.172)	55.3%	8974	626.1	(77.069)	55.5%	9007	410.8	(73.908)	62.1%	28	0.8	(84.746)	41.6%	53898	1437.9
o7	112	42	211	(29.7)	379.4%	(28.824)	393.5%	14k	700.6	(29.381)	384.5%	15k	486.6	**(48.788)**	195.6%	19	0.4	**(48.719)**	196.0%	90112	1692.3
o8_ar4_1	144	56	347	(92.973)	192.9%	**(103.621)**	163.1%	7483	702.4	**(103.846)**	162.5%	7759	503.9	(96.516)	182.2%	23	1.0	(101.071)	169.6%	40986	1564.1
o9_ar4_1	180	72	435	**(84.620)**	–	(59.276)	–	545	66.7	(84.1074)	–	4500	416.6	(77.283)	–	27	1.6	–	–	–	–
parallel	152	20	96	444	3033	**28**	88	20	1.9	**28**	88	20	1.6	43	172	8	0.3	**22**	66	23	1.1
spectra2	70	30	40	(12.936)	7.5%	2053	1271k	10	0.1	**1956**	1271k	10	0.1	(13.419)	3.9%	5	0.0	5220	3499k	5980	24.2
st_e35	29	7	33	1070	495k	700	267k	82	0.5	640	223k	91	0.3	**31**	192	12	0.0	176	64k	398	0.9
tln4	24	24	24	(5.895)	34.9%	**4**	1586	14	0.0	**4**	1586	14	0.0	(8.235)	0.7%	3	0.0	(7.078)	15.1%	35	0.0
tln5	35	35	30	(7.357)	35.2%	**1369**	833k	18	0.0	**1369**	833k	18	0.0	1639	894k	7	0.0	(9.513)	7.5%	1360	3.1
tln6	48	48	36	(8.690)	70.3%	(11.457)	32.5%	10	0.1	(11.456)	32.5%	10	0.1	**(12.334)**	23.7%	6	0.0	(12.296)	24.1%	3722	13.4
tln7	63	63	42	(6.267)	129.8%	**(7.212)**	103.3%	6	0.0	**(7.212)**	103.3%	6	0.0	(6.592)	119.9%	3	0.0	–	–	–	–
tloss	48	48	53	(16.3)	0.0%	4	110	12	0.1	4	110	12	0.1	**1**	29	6	0.0	4	105	15	0.1
tls12	792	648	372	(6.164)	–	(7.330)	–	5	2.8	(7.330)	–	5	1.9	(7.280)	–	1	0.4	**(7.486)**	–	16	3.6
tls4	105	89	64	2681	8k	3621	14k	150	1.7	**1657**	12k	156	1.6	3647	7k	14	0.0	1907	7k	434	2.6
tls5	161	136	90	(5.961)	–	(6.35)	–	39	0.7	(6.35)	–	41	0.6	**(6.5)**	–	3	0.0	**(6.5)**	–	212	2.1
tls6	213	177	120	(7.736)	–	**(8.6)**	–	22	0.6	**(8.6)**	–	22	0.6	(8.5)	–	4	0.0	(8.3)	–	167	3.1
tls7	342	293	154	(4.3)	–	(4.5)	–	13	0.7	(4.5)	–	14	0.7	(4.6)	–	2	0.0	**(4.7)**	–	59	1.5
water4	195	126	137	**(767.51)**	22.1%	(574.254)	63.2%	60	0.8	(573.25)	63.5%	58	0.6	(725.465)	29.2%	10	0.0	(746.823)	25.5%	3997	25.7
waterx	70	14	54	(639.234)	42.7%	(874.43)	4.4%	42	0.4	**3258**	131k	31	0.3	(644.874)	41.5%	5	0.0	(639.637)	42.6%	1287	6.6
waterz	195	126	137	(659.638)	56.4%	(630.313)	63.6%	115	1.3	(623.511)	65.4%	106	1.0	**(679.59)**	51.8%	12	0.0	–	–	–	–

Table 4 Comparison between five versions of Couenne, one without and four with TWOIMPL. We report: number of variables, integer variables, and constraints of each instance (var, int, con); CPU time (t) in seconds and number of branch-and-bound nodes (nd) if the variant solved the instance within the two hour time limit, otherwise the lower bound (lb) in brackets and the optimality gap (gp) measured w.r.t. the best integer solution found by any of the five variants. For the four variants using TWOIMPL, we also report the number of iterations of TWOIMPL that reduced at least one bound (n_{bt}) and the total time spent within the procedure (t_{bt}) in seconds. These instances are from the `minlplib` instance library.

				plain		twoRepeat				twoStandard				twoMinimal				twoSparse			
Name	var	int	con	t(lb)	nd(gp)	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}	t(lb)	nd(gp)	n_{bt}	t_{bt}
a1c1s1	3567	192	3231	(9913.8)	19.7%	(9888.26)	20.0%	58	22.6	(9700.91)	22.4%	2296	158.2	**(9934.52)**	19.5%	675	44.9	(9700.91)	22.4%	2337	162.7
air04	7673	7673	615	**101.3**	582	289.4	1950	1	104.5	399.2	1950	1	212.5	279.7	1950	1	94.7	398.5	1950	1	212.4
air05	6200	6200	343	**78.0**	982	87.8	357	1	48.4	145.8	359	6	103.1	89.7	357	2	47.8	140.9	359	6	101.7
mod011	6762	112	1422	**42.7**	92	106.6	218	72	38.6	153.7	198	213	90.9	105.6	204	47	36.8	155.3	198	213	91.2
momentum1	2957	1304	14019	(88797.4)	30.1%	(88808.3)	30.1%	28	101.2	**(94683.4)**	**22.0%**	82	177.6	(88801.1)	30.1%	25	95.1	**(96254.1)**	**20.0%**	183	169.2
mzzv11	9027	9027	8404	**262.4**	562	665.2	777	1	371.3	1120.8	534	5	819.8	673.6	777	1	370.0	819.5	557	2	529.7
mzzv42z	11046	11046	9818	**113.6**	10	707.0	17	1	577.4	859.1	17	1	742.4	690.0	17	1	573.4	861.2	17	1	744.7
noswot	120	95	171	3222.0	2.1m	1853.1	1.43m	15	3.1	1975.7	1.77m	5482	12.7	**1585.7**	1.58m	2150	5.0	1950.5	1.77m	5482	12.5
nw04	46190	46190	36	**18.1**	0	104.7	0	1	86.5	107.8	0	1	89.1	105.2	0	1	86.6	104.9	0	1	88.2
roll3000	877	738	1510	(12739.9)	1.1%	(12879)	0.0%	26	480.5	(12789.9)	0.7%	108	647.5	**5388.4**	43553	121	515.4	(12785.2)	0.8%	106	606.6
rout	555	315	290	465.8	85670	**379.3**	67079	15	20.6	840.5	150249	919	35.0	605.8	108904	190	22.4	846.9	153373	868	34.8
set1ch	643	235	423	1182.8	89981	1722.8	108098	59	55.3	**685.0**	53318	428	38.5	1919.9	123042	574	58.3	2455.5	181092	2517	59.5
timtab1	295	146	166	(575936)	29.3%	(583952)	27.5%	44	41.6	(581635)	28.0%	6437	229.0	**(600954)**	**23.9%**	2580	83.2	(584003)	27.5%	6528	220.3
timtab2	524	255	287	**(519324)**	**105.4%**	(489607)	117.9%	34	150.0	(475677)	124.2%	3146	310.6	(466557)	128.6%	1434	181.6	(473370)	125.3%	2796	287.7
tr12-30	1052	352	722	(114597)	16.5%	**(117524)**	**13.6%**	47	209.7	(117069)	14.0%	669	320.0	**(117598)**	**13.5%**	566	251.5	(117069)	14.0%	664	320.3

Table 5 Comparison, over MILP instances, of five versions of Couenne, one without and four with TWOIMPL. We report: number of variables, integer variables, and constraints of each instance (var, int, con); CPU time (t) in seconds and number of branch-and-bound nodes (nd) if the variant solved the instance within the two hour time limit, otherwise the lower bound (lb) in brackets and the optimality gap (gp) measured w.r.t. the best integer solution found by any of the five variants, if any. For the four variants using TWOIMPL, we also report the number of iterations of TWOIMPL that reduced at least one bound (n_{bt}) and the total time spent within the procedure (t_{bt}) in seconds. These instances are from the `miplib3` and `miplib2003` instance library.