On Handling Indicator Constraints in Mixed Integer Programming

Pietro Belotti[1], Pierre Bonami[2], Matteo Fischetti[3], Andrea Lodi[4], Michele Monaci[3], Amaya Nogales-Gómez[5], and Domenico Salvagnin[3]

[1] FICO, United Kingdom, pietrobelotti@fico.com
[2] IBM, Spain, pierre.bonami@es.ibm.com
[3] University of Padova, Italy, matteo.fischetti@unipd.it, monaci@dei.unipd.it,salvagni@dei.unipd.it
[4] University of Bologna, Italy, andrea.lodi@unibo.it
[5] Universidad de Sevilla, Spain, amayanogales@us.es

April 24, 2015

Mixed Integer Linear Programming (MILP) is commonly used to model indicator constraints, i.e., constraints that either hold or are relaxed depending on the value of a binary variable. Unfortunately, those models tend to lead to weak continuous relaxations and turn out to be unsolvable in practice, like in the case of Classification problems with Ramp Loss functions that represent an important application in this context.

In this paper we show the computational evidence that a relevant class of these Classification instances can be solved far more efficiently if a nonlinear, nonconvex reformulation of the indicator constraints is used instead of the linear one. Inspired by this empirical and surprising observation, we show that aggressive bound tightening is the crucial ingredient for solving this class of instances, and we devise a pair of computationally effective algorithmic approaches that exploit it within MILP.

More generally, we argue that aggressive bound tightening is often overlooked in MILP, while it represents a significant building block for enhancing MILP technology when indicator constraints and disjunctive terms are present.

Keywords: Mixed-Integer Linear Programming, Mixed-Integer Quadratic Programming, Indicator Constraints.

1 Introduction

Let us consider the linear inequality

$$\alpha^T x \leq x_0, \tag{1}$$

in which both $x \in \mathbb{R}^d$ and $x_0 \in \mathbb{R}$ are variables, while α is a given d-dimensional vector. It is a very well-known modeling trick in Mixed Integer Linear Programming (MILP) to use a binary variable to control whether linear constraint (1) is active or not depending on other parts of the model or at the price of paying a penalty in the objective function. Then, the constraint is reformulated as the following *big-M* or *indicator* constraint

$$\alpha^T x \leq x_0 + Mt, \tag{2}$$

where $t \in \{0, 1\}$ and M is a big enough value that guarantees that the constraint is inactive if $t = 1$.

Although they provide a clean and flexible modeling tool to deal with nonlinearities and logical implications by staying within the MILP framework, it is well-known that indicator constraints present the drawback of having a weak continuous relaxation. Indeed, depending on the value M and on the value attained by expression "$\alpha^T x - x_0$", very small (fractional) values of t might be sufficient to satisfy the constraint. This leads to quality issues with a continuous relaxation value typically very far away from the mixed integer optimum, but, sometimes even more importantly, might lead to numerical issues, with the MILP solvers being unable to assert if a t value below the integer tolerance is in fact a true solution.

An alternative for logical implications that has been used in the Mixed Integer Nonlinear Programming (MINLP) literature for decades is provided by the *complementary* formulation

$$(\alpha^T x - x_0)\bar{t} \leq 0, \tag{3}$$

where $\bar{t} = 1 - t$. However, (3) is a nonconvex constraint. Such a source of nonconvexity might not significantly complicate the solution of already nonconvex MINLP models arising, for example, in Chemical Engineering applications, see [13]. In addition, numerical issues on the choice of the value of M do not appear anymore, at least in the formulation. On the contrary, in the cases where those logical constraints were the only sources of nonconvexity, the common approach has always been that of using constraints (2) and MILP techniques.

Before stating the contribution of the present paper it is worth mentioning that the *big-M* formulation (2) is just the weakest (but easiest and commonly used) disjunctive programming approach (see, e.g., [2, 8] and [13]) to deal with indicator constraints and disjunctions in general. The reader is referred to [4] for a detailed and more theoretical discussion on the topic that is outside the scope of this paper.

Contribution of the paper. In this paper we expose a class of convex Mixed Integer Quadratic Programming (MIQP) problems arising in *Supervised Classification* where the Global Optimization (GO) solver `Couenne` [10] using reformulation (3) is consistently faster than virtually any state-of-the-art commercial MIQP solver like [15], [14] and [23]. This is quite counter-intuitive because, in general, convex MIQPs admit more efficient solution techniques both in theory and in practice, especially by benefiting from virtually all machinery of MILP solvers. Inspired by this empirical and surprising observation, we show that aggressive bound tightening is the crucial ingredient for solving this class of instances, and we devise a pair of computationally effective algorithmic approaches that exploit it within MILP. In particular, we were able to optimally solve in just seconds instances that could not be solved by state-of-the-art MILP solvers in hours. More generally, we argue that aggressive bound tightening is often overlooked in MILP, while it represents a significant building block for enhancing MILP technology when indicator constraints and disjunctive terms are present.

Organization of the paper. The remainder of the paper is organized as follows. In Section 2 we discuss the application we use as an example. In Section 3 we show the initial set of surprising computational results. In Section 4 we discuss why those results are surprising while in Section 5 we carefully analyze the reasons of the success of `Couenne` versus `IBM-Cplex`. In Section 6 we present two approaches to enhance `IBM-Cplex` trying to mimic `Couenne`'s behavior. Finally, some conclusions are drawn in Section 7.

2 Support Vector Machines with the ramp loss

In *Supervised Classification*, see, e.g., [22], we are given a set of objects Ω partitioned into classes and the aim is to build a procedure for classifying new objects. In its simplest form, each object $i \in \Omega$ has associated a pair (x_i, y_i), where the predictor vector x_i takes values on a set $X \subseteq \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ is the class membership of object i, also known as the label of object i.

Support Vector Machines (SVM) methods, see, e.g., [11], have proven to be one of the state-of-the-art methods for Supervised Classification. The SVM aims at separating the two classes by means of a hyperplane, $\omega^\top x + b = 0$, found by solving the optimization problem

$$\min_{\omega \in \mathbb{R}^d,\, b \in \mathbb{R}} \frac{\omega^\top \omega}{2} + \frac{C}{n} \sum_{i=1}^{n} g((1 - y_i(\omega^\top x_i + b))^+),$$

where n is the size of the sample used to build the classifier, $(a)^+ = \max\{a, 0\}$, C is a nonnegative parameter, and g a nondecreasing function in $\mathbb{R}_+$, the so-called *loss* function. The reader is referred to [6] for a recent review on Mathematical Optimization and SVMs.

Several recent papers have been devoted to analyzing SVM with the so-called *ramp loss* function, $g(t) = (\min\{t, 2\})^+$, discussed in details in the next section.

An MIQP formulation. In this paper, we are interested in the SVM with the ramp loss function, see [9] and [20]. In this model, objects are penalized in a different way depending on whether or not they fall inside or outside the margin, i.e., if they fall between $\omega^\top x + b = -1$ and $\omega^\top x + b = 1$. Misclassified objects that fall outside the margin have a fixed loss of 2, while objects that fall inside the margin have a continuous loss between 0 and 2. The state-of-the-art algorithm is given in [5], where the ramp loss model, denoted as RLM, is formulated as the MIQP problem

$$\min_{\omega,b,\xi,z} \frac{1}{2}\sum_{j=1}^{d}\omega_j^2 + \frac{C}{n}\left(\sum_{i=1}^{n}\xi_i + 2\sum_{i=1}^{n}z_i\right) \tag{4}$$

s.t. (RLM)

$$y_i(\omega^\top x_i + b) \geq 1 - \xi_i - Mz_i \qquad \forall i = 1,\dots,n \tag{5}$$
$$0 \leq \xi_i \leq 2 \qquad \forall i = 1,\dots,n \tag{6}$$
$$z \in \{0,1\}^n \tag{7}$$
$$\omega \in \mathbb{R}^d \tag{8}$$
$$b \in \mathbb{R}, \tag{9}$$

where $M > 0$ is a big enough constant, $\xi = (\xi_i)$ denotes the vector of deviation/penalty variables and C is the tradeoff parameter that calls for tuning. For a given object i, the binary variable z_i is equal to 1 if object i is misclassified outside the margin and 0 otherwise. The reader is referred to [5] for further details on this formulation, which is denoted as "SVMIP1(ramp)" in [5].

The appeal of model (4)–(9) relies in the fact that it can (potentially) be solved by a black-box MIQP solver, e.g., [15]. More precisely, objective function (4) is convex while constraints are linear, thus virtually all the very sophisticated and effective machinery for MILP problems can be applied. However, the solution method proposed in [5] is able to solve to optimality only a quite limited number of instances, although some problem-specific cutting planes and reductions are used to help the MILP solver, namely, `IBM-Cplex`. Essentially, this difficulty is due to the *big-M* constraints (5) that make the continuous relaxation of model (4)–(9) very weak. Branching is effective for small problems but the almost-complete enumeration is ineffective for instances of serious size. Cutting planes are not likely to solve the problem, unless they would be specifically designed to face the *big-M* issue, or, more precisely, the disjunctive nature of constraints (5).

A Nonconvex Formulation. Motivated by the discussed difficulty of dealing with constraints (5), we analyzed the alternative nonlinear, nonconvex, formulation of the RLM

$$\min_{\omega,b,\xi,\bar{z}} \frac{1}{2}\sum_{j=1}^{d}\omega_j^2 + \frac{C}{n}\left(\sum_{i=1}^{n}\xi_i + 2\sum_{i=1}^{n}(1-\bar{z}_i)\right) \tag{10}$$

s.t.

$$
\begin{aligned}
(y_i(\omega^\top x_i + b) - 1 + \xi_i) \cdot \bar{z}_i \geq 0 \qquad & \forall i = 1, \ldots, n && (11)\\
0 \leq \xi_i \leq 2 \qquad & \forall i = 1, \ldots, n && (12)\\
\bar{z} \in \{0, 1\}^n & && (13)\\
\omega \in \mathbb{R}^d & && (14)\\
b \in \mathbb{R}. & && (15)
\end{aligned}
$$

Precisely as in RLM (4)–(9), binary variables are used to disable constraints (11), which replace constraints (5), but are the complemented version of z variables (13), i.e., $\bar{z}_i = 1 - z_i$. Namely, $\bar{z}_i = 1$ forces the i-th constraint to be active, thus allowing a maximum violation of $\xi_i = 2$, while $\bar{z}_i = 0$ disables the constraint in a classical "complementary" way.

Of course, constraints (11) are responsible of the nonconvexity of the MINLP model (10)–(15). However, its continuous version obtained by simply replacing constraints (13) with $\bar{z} \in [0, 1]^n$ is solved to (local) optimality by the Nonlinear Programming (NLP) solver [16] providing a mixed binary solution that is very accurate, and relatively quick to compute. Indeed, it is easy to prove that

Proposition 2.1 *Any local optimal solution of the continuous version of model* (10)–(15) *is mixed binary.*

Proof: The proposition is proven by contradiction. A local optimal solution $(\omega, b, \xi, \bar{z})$ is feasible. Thus, constraints (11) are satisfied. For any $i = 1, \ldots, n$, either $\bar{z}_i = 0$ (integer), or if $\bar{z}_i \in (0, 1)$, there exists an equivalent (still feasible) solution $(\omega, b, \xi, \bar{z}_1, \ldots, \bar{z}_{i-1}, 1, \bar{z}_{i+1}, \ldots, \bar{z}_n)$ with $\bar{z}_i = 1$ having smaller objective function value. □

Theorem 2.1 above implies that the global optimal solution is mixed binary as well, thus solving the continuous version of the problem with a GO solver like [10] solves the overall problem to optimality. On the other hand, `Couenne` is an MINLP solver, hence it can handle integrality constraints on (a subset of) the variables. Thus, in the results presented in the next section we have kept integrality on variables $\bar{z}$'s.

3 A raw set of computational results

We have performed an exploratory test for the nonconvex MINLP formulation proposed in Section 2. We consider only the artificial datasets proposed by [5], to be able to control both the dataset and the problem size. In this paper we concentrate on a challenging subset of instances, namely 23 instances of size $n = 100$, $d = 2$, `TypeB` proposed in [5]. We show the surprising behavior of [10] that used "out-of-the-box" performs better on model (10)–(15) than [15] on model (4)–(9). Table 1 reports the straightforward comparison. Computing times (time) in CPU seconds, number of nodes (nodes), percentage gap of the upper (ub) and lower (lb) bounds are reported, as well as the optimal value of each

Table 1: Computational results for `Couenne` and `IBM-Cplex`. Instances of `TypeB` proposed by [5], $n = 100$, time limit of 1 hour, executed on an Intel Xeon E3-1220V2.

		`Couenne`				`IBM-Cplex`			
				% gap				% gap	
	optimal value	time (sec.)	nodes	ub	lb	time (sec.)	nodes	ub	lb
1	157,994.959	151.87	22,574	–	–	×	13,807,517	–	13.28
2	179,368.534	595.06	108,682	–	–	×	15,676,267	–	24.00
3	220,673.592	×	646,606	4.59	10.60	×	16,436,987	–	38.23
4	5,225.994	142.95	22,934	–	–	×	8,883,191	–	16.68
5	5,957.083	1,180.24	251,678	–	–	×	14,841,992	–	28.44
6	11,409,617.494	1,571.70	257,749	–	–	×	15,367,015	–	23.05
7	11,409,058.363	698.72	120,425	–	–	×	15,765,182	–	23.53
8	10,737,725.660	448.38	70,342	–	–	×	14,794,402	–	18.64
9	5,705,364.054	1,433.84	238,028	–	–	×	16,212,686	–	23.54
10	5,704,804.923	578.12	104,368	–	–	×	15,075,606	–	23.54
11	5,369,016.540	348.82	56,161	–	–	×	14,624,208	–	18.62
12	2,853,237.334	1,637.12	267,588	–	–	×	15,225,643	–	23.10
13	2,852,678.203	565.32	101,677	–	–	×	14,846,871	–	23.23
14	2,684,661.980	340.74	54,212	–	–	×	14,453,730	–	18.46
15	1,427,173.974	1,508.68	247,860	–	–	×	15,213,788	–	23.36
16	1,426,614.843	525.55	93,268	–	–	×	16,113,468	–	23.56
17	1,342,484.700	394.45	61,247	–	–	×	15,237,849	–	18.77
18	714,142.294	1,156.81	186,351	–	–	×	15,550,460	–	23.30
19	713,583.163	513.23	91,329	–	–	×	15,762,471	–	23.59
20	671,396.060	498.46	77,747	–	–	×	15,355,449	–	18.79
21	357,626.454	1,084.87	180,408	–	–	×	15,889,182	–	23.66
22	357,067.323	669.17	117,288	–	–	×	15,280,262	–	23.63
23	335,851.740	448.92	71,110	–	–	×	15,322,743	–	18.82

instance for future reference. A time limit of 1 hour is provided to each run and in case such a limit is reached the entry in the column "time" indicates a "×" (computing times are in CPU seconds on an Intel Xeon E3-1220V2). For instances solved to optimality gaps are reported as "–".

The results of Table 1 are quite straightforward to interpret, with a strict dominance of `Couenne` with respect to `IBM-Cplex`. In the unique instance `Couenne` is unable to solve to optimality (instance 3) the issue is probably that the upper bound is not improved enough, thus being unable to propagate (see next section) and strengthen the formulation. Conversely, `IBM-Cplex` is always able to find the right upper bound (namely, the optimal solution value) but the lower bound value remains far from the optimal value, thus being unable to prove optimality.

Table 1 reports detailed numbers only for `IBM-Cplex` but we solved the convex MIQP model (4)–(9) with the convex MIQP solvers developed extending the three major MILP solvers, namely [14] and [23], and `IBM-Cplex` itself. The three solvers behave very similarly in the considered instances, thus indicating that the weakness shown in Table 1 is structurally associated with solving the *big-M* formulation, or, conversely, that solving the nonconvex formulation through a GO solver is effective.

4 Why are these results surprising?

Although, as anticipated in the introduction, convex MIQP solvers should be more effective than GO ones especially because they can exploit the very sophisticated MILP machinery, one can still argue that a comparison in performance between two different solution methods and computer codes is anyway hard to perform. However, digging into the way `Couenne` solves the problem leads to confirm the initial surprise or even increase it.

McCormick Linearization. The first observation is that the way constraints (11) are managed by `Couenne` is through the classical McCormick linearization (see, [17]). Namely, for $i = 1, \dots, n$, two new auxiliary variables ϑ_i and u_i are introduced that are associated with expressions in (11):

1. $\vartheta_i = y_i(\omega^\top x_i + b) - 1 + \xi_i$, with $\vartheta_i^L \le \vartheta_i \le \vartheta_i^U$
2. $u_i = \vartheta_i \bar{z}_i$.

Then, the product corresponding to each new variable u_i is linearized as

$$\begin{aligned} u_i &\ge 0 && (16)\\ u_i &\ge \vartheta_i^L \, \bar{z}_i && (17)\\ u_i &\ge \vartheta_i + \vartheta_i^U \, \bar{z}_i - \vartheta_i^U && (18)\\ u_i &\le \vartheta_i + \vartheta_i^L \, \bar{z}_i - \vartheta_i^L && (19)\\ u_i &\le \vartheta_i^U \, \bar{z}_i, && (20) \end{aligned}$$

again for $i = 1, \dots, n$, where (16) corresponds precisely to (11). Essentially, setting $\bar{z}_i = 0$ again deactivates constraint i by simply enforcing the loose $\vartheta_i \in [\vartheta_i^L, \vartheta_i^U]$, where ϑ_i^L plays the role of the *big M*.

In other words, `Couenne` initially builds a linear *big-M* formulation itself, with the difference that a specific ϑ_i^L value for each i is computed. Although such an internal computation is not responsible of the higher effectiveness of `Couenne` (typically `Couenne` is more conservative than the static values of M used in the literature and especially by [5], this is, in practice, not a negligible issue because a safe value of M is not trivial to be determined a priori.

In the next section we will extensively discuss how McCormick inequalities are strengthened, as well as the bounds on ϑ variables. This will be shown to be crucial for `Couenne` but, at first, this similarity confirms the surprise.

Branching. It is well known that a major component of GO solvers is the iterative tightening of the convex (most of the time linear) relaxation of the nonconvex feasible region by branching on continuous variables (see, e.g., [3]). Another surprising fact here is that the default version of `Couenne` does not take advantage of this possibility and branches on the binary variables $\bar{z}$'s. Because everything is linear (after McCormick linearization) and the objective function convex, as soon as all binaries are fixed the problem is solved.

Even better performance for `Couenne` could be obtained by branching on continuous variables. Namely, instructed to branch preferably on continuous variables, `Couenne` always selects ϑ variables, which clearly lead to additional bound tightening with respect to branch on binaries. Indeed, if in a given relaxation we have $\vartheta_i = c$, the two branches $\vartheta_i \leq c$ OR $\vartheta_i \geq c$ propagate as follows: (i) if $c < 0$ then $\vartheta_i \leq c$ implies $\bar{z}_i = 0$ OR (ii) if $c > 0$ then $\vartheta_i \geq c$ implies $\bar{z}_i = 1$. The results are reported in Table 2 and clearly show the computational advantage of this choice. Thus, again, it is surprising that the default branching strategy of `Couenne` leads to an improvement over the sophisticated branching framework of `IBM-Cplex`.

Table 2: Computational results for `Couenne` default and `Couenne` branching emphasis on continuous variables. Instances of `TypeB` proposed by [5], $n = 100$, time limit of 1 hour, executed on an Intel Xeon E3-1220V2.

		`Couenne` default				`Couenne` continuous			
				% gap				% gap	
	optimal value	time (sec.)	nodes	ub	lb	time (sec.)	nodes	ub	lb
1	157,994.959	151.87	22,574	–	–	257.54	74,424	–	–
2	179,368.534	595.06	108,682	–	–	479.86	140,405	–	–
3	220,673.592	$\times$	646,606	4.59	10.60	774.60	216,118	–	–
4	5,225.994	142.95	22,934	–	–	408.33	121,896	–	–
5	5,957.083	1,180.24	251,678	–	–	724.94	209,346	–	–
6	11,409,617.494	1,571.70	257,749	–	–	640.58	184,391	–	–
7	11,409,058.363	698.72	120,425	–	–	912.07	269,332	–	–
8	10,737,725.660	448.38	70,342	–	–	492.00	143,426	–	–
9	5,705,364.054	1,433.84	238,028	–	–	609.29	177,236	–	–
10	5,704,804.923	578.12	104,368	–	–	921.80	276,704	–	–
11	5,369,016.540	348.82	56,161	–	–	505.37	145,282	–	–
12	2,853,237.334	1,637.12	267,588	–	–	624.38	176,251	–	–
13	2,852,678.203	565.32	101,677	–	–	855.99	255,654	–	–
14	2,684,661.980	340.74	54,212	–	–	466.56	138,252	–	–
15	1,427,173.974	1,508.68	247,860	–	–	571.29	163,552	–	–
16	1,426,614.843	525.55	93,268	–	–	811.76	251,205	–	–
17	1,342,484.700	394.45	61,247	–	–	361.27	108,762	–	–
18	714,142.294	1,156.81	186,351	–	–	513.45	148,947	–	–
19	713,583.163	513.23	91,329	–	–	722.22	219,986	–	–
20	671,396.060	498.46	77,747	–	–	382.18	118,811	–	–
21	357,626.454	1,084.87	180,408	–	–	459.15	133,964	–	–
22	357,067.323	669.17	117,288	–	–	611.38	185,459	–	–
23	335,851.740	448.92	71,110	–	–	404.87	116,966	–	–

L_1 **norm.** A natural question is if the results reported in Table 1 are due to the somehow less sophisticated evolution of `IBM-Cplex` in its MIQP extension with respect to the MILP one. In order to answer this question we performed an experiment in which the quadratic part of the objective function was replaced by the L_1 norm on ω. More precisely, the sum of the absolute values of ω_i is minimized (and linear constraints to deal with the absolute value are added). This results in a pure MILP once the *big-M* constraints (5) are considered (solved by `IBM-Cplex`) or a nonconvex MINLP with linear objective function (solved by `Couenne`) if constraints (11) are used instead. This linear objective variant of the RLM does not result in any change for the comparison and `Couenne` continues achieving better results than `IBM-Cplex`.

5 Bound reduction in nonconvex MINLP problems

Bound reduction is a crucial tool in MINLP: it allows one to eliminate portions of the feasible set while guaranteeing that at least one optimal solution is retained. Although its origins can be traced back to Artificial Intelligence (see, [12]), it finds wide application in Constraint Programming and in solvers for both Nonlinear Optimization ([18]) and for MILP problems ([1] and [19]).

Consider a generic optimization problem $\min\{f(x) : x \in X, \ell \leq x \leq u\}$, where $X \subset \mathbb{R}^n$ and $x, \ell, u \in \mathbb{R}^n$. Also, suppose an upper bound $U \in \mathbb{R} \cup \{+\infty\}$ on the objective function value of the optimal solution is available: if $U < +\infty$, then a feasible solution $x \in X \cap [\ell, u]$ is available such that $f(x) = U$. Bound reduction attempts to find tighter lower bounds $\ell'_i > \ell_i$ and upper bounds $u'_i < u_i$. In general, a good upper bound is often the key to a strong bound reduction.

An ideal bound reduction procedure obtains bounds by exploiting the full problem structure:

$$\begin{array}{rcll} \ell'_i & = & \max & \{x_i : x \in X, \ell \leq x \leq u, f(x) \leq U\}, \\ u'_i & = & \min & \{x_i : x \in X, \ell \leq x \leq u, f(x) \leq U\}. \end{array} \tag{21}$$

However, the $2n$ optimization problems above can be as hard as the original optimization problem itself, therefore this approach is impractical.

A fast bound reduction procedure, known as *Feasibility Based Bound Tightening* (FBBT), yields new bounds on a variable x_i using bounds on other variables that are linked to x_i through a constraint or the objective function. For instance, the constraint $x_1 x_2 \leq 4$ and the bounds $x_1 \geq 1, x_2 \geq 1$ yield new upper bounds $x_1 \leq 4, x_2 \leq 4$. An example that is closer to our application is the constraint $x_i^2 \leq u$, where $u \geq 0$, which obviously implies $x_i \in [-\sqrt{u}, \sqrt{u}]$.

A specialized version of this procedure applies to affine functions, and is commonly used in MILP solvers, see [1]. Consider the range constraint

$$\ell_0 \leq \alpha_0 + \sum_{j=1}^{n} \alpha_j x_j \leq u_0.$$

Define $J^+ = \{j = 1, \dots, n : \alpha_j > 0\}$ and $J^- = \{j = 1, \dots, n : \alpha_j < 0\}$. Bounds ℓ_0, u_0 on the expression imply new (possibly tighter) bounds ℓ'_j, u'_j on $x_j, j = 1, \dots, n : \alpha_j \neq 0$:

$$\begin{array}{rrcl} \forall j : \alpha_j > 0, & \ell'_j & = & \frac{1}{\alpha_j}\left(\ell_0 - \left(\alpha_0 + \sum_{i \in J^+ \setminus \{j\}} \alpha_i u_i + \sum_{i \in J^-} \alpha_i \ell_i\right)\right), \\ & u'_j & = & \frac{1}{\alpha_j}\left(u_0 - \left(\alpha_0 + \sum_{i \in J^+ \setminus \{j\}} \alpha_i \ell_i + \sum_{i \in J^-} \alpha_i u_i\right)\right); \\ \forall j : \alpha_j < 0, & \ell'_j & = & \frac{1}{\alpha_j}\left(u_0 - \left(\alpha_0 + \sum_{i \in J^+} \alpha_i \ell_i + \sum_{i \in J^- \setminus \{j\}} \alpha_i u_i\right)\right), \\ & u'_j & = & \frac{1}{\alpha_j}\left(\ell_0 - \left(\alpha_0 + \sum_{i \in J^+} \alpha_i u_i + \sum_{i \in J^- \setminus \{j\}} \alpha_i \ell_i\right)\right). \end{array} \tag{22}$$

5.1 Applying bound reduction to model (10)-(15)

`Couenne` is a branch-and-bound solver for MINLP problems that uses, among others, several bound reduction techniques, including FBBT. At the beginning, `Couenne` runs a greedy rounding procedure to obtain a feasible solution of the problem, and hence an upper bound U. Applying FBBT using two rules mentioned above (for affine functions and for the square operator) yields tight bounds on ω_i at the root node of `Couenne`'s branch-and-bound tree. Consider the objective function of our problem

$$\frac{1}{2}\sum_{j=1}^{d} \omega_j^2 + \frac{C}{n}\left(\sum_{i=1}^{n} \xi_i + 2\sum_{i=1}^{n}(1 - \bar{z}_i)\right),$$

which is bounded from above by U. Also, note that $\sum_{i=1}^n \xi_i + 2\sum_{i=1}^n (1 - \bar{z}_i)$ is nonnegative. Since $\sum_{j=1}^d \omega_j^2 \geq 0$ and $\sum_{i=1}^n \xi_i + 2\sum_{i=1}^n (1 - \bar{z}_i) \geq 0$, and also ξ_i and $\bar{z}_i$ are nonnegative, we have

$$\omega_i \in \left[-\sqrt{2U}, \sqrt{2U}\right] \qquad \forall i = 1, \dots, d.$$

A related observation concerns the constraints of our model. The tighter bounds on the ω_i's variables, which are initially unbounded in the definition of the problem, do not seem to have an influence on constraints (11), where the variable b remains unbounded. This family of nonlinear constraints can be simplified to $\vartheta_i \bar{z}_i \geq 0$, with $\vartheta_i = (y_i(\omega^\top x_i + b) - 1 + \xi_i) \in [-\infty, +\infty]$ and $\bar{z}_i \in \{0, 1\}$, for all $i = 1, \dots, n$. Due to the infinite bounds, this constraint does not admit a linear relaxation, which is useful for any MINLP solver to obtain a lower bound. When imposing fictitious bounds $[-M, M]$ on ϑ_i, with large enough M, one gets the constraint $\vartheta_i \geq M(\bar{z}_i - 1)$, which is the *big* M constraint used in MILP. In these cases, *probing* techniques can be of help. A probing bound reduction algorithm works as follows: impose a fictitious upper bound $\lambda_i \in (\ell_i, u_i)$ on a variable x_i, thereby restricting x_i to $[\ell_i, \lambda_i]$. If the restricted problem can be proven (through FBBT, for example) to be infeasible or to have a lower bound that is above a cutoff U, then no feasible solution with better objective function value can be found in the restriction. Therefore, the new lower bound $x_i \geq \lambda_i$ is valid. This procedure can be applied to tighten the upper bound as well, by imposing a fictitious lower bound $\mu_i \in (\ell_i, u_i)$. Although applying it to all variables is time consuming, it is especially useful for unbounded variables. Probing is a common tightening technique in MILP and MINLP solvers (see [19], and [3] and [21], respectively.)

We will now describe the specific reductions that `Couenne` applied to the RLM, and we will computationally show that these reductions are crucial to obtain the results shown in Section 4.

5.2 Strengthening McCormick constraints

The coefficients of the McCormick constraints are the lower and upper bounds on the variables involved. Hence, these constraints can be replaced by stronger ones as soon as tighter bounds on the variables are available. `Couenne` does this automatically by means of a cut separator that is called at every branch-and-bound node, and only adds tighter McCormick cuts if they are violated by the Linear Programming (LP) solution available at that node.

Note that McCormick cuts are only useful if both variables ϑ_i and $\bar{z}_i$ are not fixed, as, otherwise, the constraint $u_i = \vartheta_i \bar{z}_i$ becomes linear. Of course, new McCormick cuts are making previous ones redundant and `Couenne` relies on the branch-and-bound manager (specifically, [7]) to deal with redundancy in the constraint set.

While looking for reasons of `Couenne`'s performance, we have run an experiment where McCormick cuts were only added to the initial LP relaxation but excluded from separation at all nodes. The performance worsened dramatically on all instances, which indicates that the bound on the involved variables $u_i, \vartheta_i, \bar{z}_i$ is tightened and should be exploited.

5.3 Bound tightening

`Couenne` uses several techniques for bound tightening among those described above. In the context of this problem, tightening is based on the following elements:

- the objective function, if a cutoff U is available;
- the definition $u_i = \vartheta_i \bar{z}_i (\geq 0)$ and related constraint $u_i \geq 0$;
- the definition $\vartheta_i = (y_i(\omega^\top x_i + b) - 1 + \xi_i)$.

Propagation possibly generates new bounds on ω, b, ξ, which are obtained through standard presolve procedures ([1]). For the sake of completeness, we add them here by defining the coefficients $\alpha_{ik} = x_{ik} y_i$, where x_{ik} is the $k-$th feature ($k = 1, \ldots, d$) of object i ($i = 1, \ldots, n$). Also, we use superscripts "L" (resp. "U") to denote lower (resp. upper) bounds. The bounds are then updated by formulas

$$
\begin{array}{lrcl}
\forall k : \alpha_{ik} > 0 & \omega_k^L & = & \frac{1}{\alpha_{ik}} \left(\vartheta_i^L - \sum_{j:\alpha_{ij}<0} \alpha_{ij}\omega_j^L - \sum_{j\neq k:\alpha_{ij}>0} \alpha_{ij}\omega_j^U - (y_ib)^U - \xi^U + 1 \right), \\
 & \omega_k^U & = & \frac{1}{\alpha_{ik}} \left(\vartheta_i^U - \sum_{j:\alpha_{ij}<0} \alpha_{ij}\omega_j^U - \sum_{j\neq k:\alpha_{ij}>0} \alpha_{ij}\omega_j^L - (y_ib)^L - \xi^L + 1 \right), \\
\forall k : \alpha_{ik} < 0 & \omega_k^L & = & \frac{1}{\alpha_{ik}} \left(\vartheta_i^U - \sum_{j\neq k:\alpha_{ij}<0} \alpha_{ij}\omega_j^U - \sum_{j:\alpha_{ij}>0} \alpha_{ij}\omega_j^L - (y_ib)^L - \xi^L + 1 \right), \\
 & \omega_k^U & = & \frac{1}{\alpha_{ik}} \left(\vartheta_i^L - \sum_{j\neq k:\alpha_{ij}<0} \alpha_{ij}\omega_j^L - \sum_{j:\alpha_{ij}>0} \alpha_{ij}\omega_j^U - (y_ib)^U - \xi^U + 1 \right), \\
\forall i : y_i > 0, & b^L & = & \frac{1}{y_i} \left(\vartheta_i^L - \sum_{j:\alpha_{ij}<0} \alpha_{ij}\omega_j^L - \sum_{j:\alpha_{ij}>0} \alpha_{ij}\omega_j^U - \xi^U + 1 \right), \\
 & b^U & = & \frac{1}{y_i} \left(\vartheta_i^U - \sum_{j:\alpha_{ij}<0} \alpha_{ij}\omega_j^U - \sum_{j:\alpha_{ij}>0} \alpha_{ij}\omega_j^L - \xi^L + 1 \right), \\
\forall i : y_i < 0, & b^L & = & \frac{1}{y_i} \left(\vartheta_i^U - \sum_{j:\alpha_{ij}<0} \alpha_{ij}\omega_j^U - \sum_{j:\alpha_{ij}>0} \alpha_{ij}\omega_j^L - \xi^L + 1 \right), \\
 & b^U & = & \frac{1}{y_i} \left(\vartheta_i^L - \sum_{j:\alpha_{ij}<0} \alpha_{ij}\omega_j^L - \sum_{j:\alpha_{ij}>0} \alpha_{ij}\omega_j^U - \xi^U + 1 \right), \\
 & \xi^L & = & \vartheta_i^L - \sum_{j:\alpha_{ij}<0} \alpha_{ij}\omega_j^L - \sum_{j:\alpha_{ij}>0} \alpha_{ij}\omega_j^U - (y_ib)^U + 1, \\
 & \xi^U & = & \vartheta_i^U - \sum_{j:\alpha_{ij}<0} \alpha_{ij}\omega_j^U - \sum_{j:\alpha_{ij}>0} \alpha_{ij}\omega_j^L - (y_ib)^L + 1,
\end{array}
$$

where, as an example, the value ω_k^L is the maximum lower bound value on ω_k over all $i = 1, \ldots, n$ of the first formula above. The same holds for all other bounds.

When solving an instance of our problem, tightening typically happens after obtaining a new integer feasible solution or after branching on a variable. We describe here in detail the tightening steps. Note that the sequence of tightening steps is often repeated to ensure "tight enough" bounds on all variables, especially the "critical" ones, i.e., those that can be free and unbounded, like ω's and b in out case. This loop is of fundamental importance in GO solvers that, generally, cannot rely on tight continuous relaxations.

- After branching on a binary variable $\bar{z}_i$, if $\bar{z}_i$ is fixed to 0 the lower bound on the objective function is increased, which may allow for extra tightening on the variables appearing in the objective or, if the lower bound is above the cutoff, for pruning the node.

- If $\bar{z}_i$ is instead fixed to 1, `Couenne` uses the new lower bound on ϑ_i to obtain a better bound on ω and b.

- When a new upper bound U is found (`Couenne` finds a good one at the beginning), this triggers a tightening of ω. No tightening is done on any $\bar{z}_i$ variable since all of them have the same coefficient in the objective.

Note that in both cases (branching on $\bar{z}_i$ and new bound U) the tightening of ω's and b above allows one to strengthen the McCormick inequalities, that are necessary to build a linear relaxation of the problem. To summarize, both branches and a new cutoff value allow to tighten the bounds on ϑ, ω, and b, and this, in turn, allows for strengthening McCormick inequalities at every node.

Needless to say, disabling bound reduction in `Couenne` leads to a dramatic worsening of the performance. It appears therefore that both bound reduction and McCormick cuts are the responsible of the success of `Couenne` on this class of instances, and, conversely, the lack of those ingredients, or at least their "too light" use, seems to be crucial in the troubles encountered by MILP solvers. In the next section we will show that enhancing the MILP solvers in this line is possible.

6 Enhancing MILP solvers

Inspired by the outperforming results of `Couenne` over `IBM-Cplex` (and virtually all other MILP solvers), and by the analysis in the previous section, we have tried to exploit the successful MINLP tools, namely bound tightening, to deal with the weak MILP relaxations associated with *big-M* constraints. We will show two ways of enhancing the behavior of `IBM-Cplex`

- either by using online (and cheap) tightening of Section 5.3 and locally-valid McCormick constraints,
- or by an *a priori* (more expensive) strengthening of the formulation, where the bounds on ω's and b are computed by solving MILPs as in (21).

The former approach, outlined in Section 6.1, leans towards a real integration of aggressive bound tightening in MILP, while the latter can be seen as a sophisticated MILP algorithm that exploits bound tightening and is described in Section 6.2.

6.1 Local cuts

As explained in Section 5.2, one of the main ingredients used by `Couenne` to solve the classification instances is the iterative strengthening of constraints (5) in the tree. This process can be mimicked in a MILP solver by using locally valid versions of the *big-M* constraints (5).

Basically, any time one tightens the upper and lower bounds on ω and b through propagation, one can recompute a potentially smaller value of M by recomputing the minimal value the left-hand-side of constraint (5) can take under these new bounds. If this value is indeed smaller, one can add a tighter version of (5). Note that this new constraint strictly dominates the previous one. As explained in Section 5, the combination of branching and propagation gives tighter bounds for ω and b at the nodes of the branch-and-bound tree. However, since these bounds on ω and b are computed for a node of the tree, the resulting strengthened version of (5) is only valid for the sub-tree rooted in that node.

In [15], locally valid versions of constraints (5) can be automatically added as *local cuts* within the branch-and-bound tree. These inequalities as refereed to as *local implied bound cuts.*

The way in which `IBM-Cplex` generates these cuts is through the use of so-called *implications.* An implication is the logical description of an indicator constraint: $t =$

0 (or 1) $\Rightarrow \alpha^\top x \leq x_0$. Implications can be either automatically found by presolve procedures that analyze the structure of the model or, alternatively, they can be directly given as an input to the solver [15, 23]. Implications naturally imply linear cuts by using their *big-M* form (2).

To assess the efficiency of local implied bound cuts on the supervised classification instances, we replaced the constraints (5) with the indicator constraints

$$z = 0 \Rightarrow y_i(\omega^\top x_i + b) \geq 1 - \xi_i \qquad \forall i = 1, \dots, n.$$

We then solved the 23 instances of Table 1, without local implied bound cuts (default settings) and with a very aggressive setting for separating local implied bound cuts. In this very aggressive settings, Cplex tries to recompute a smaller *big M* for each indicator constraint at every node of the branch-and-bound tree. Every time a smaller *big M* is found, the previous *big-M* constraint is removed and replaced by the tighter one.

Computing times in CPU seconds and number of nodes are reported in Table 3. `IBM-Cplex` is executed (using 4 threads), in the default settings and with local implied bound cuts set to very aggressive. While default `IBM-Cplex` is able to solve to optimality just one instance within the time limit, the enhanced version solves to optimality 15 out of 23 instances.

6.2 Iterative domain reduction

Iterative domain reduction can be seen as a preprocessing tool to enhance the behavior of `IBM-Cplex`. An initial bound tightening is performed by solving a sequence of MILP problems to strengthen the lower and upper bounds on the ω variables and on b.

Let us denote by P the set of feasible solutions of the RLM, by $Z(\omega, \xi, z)$ the objective value (4) of solution (ω, ξ, z), and by U the value of an upper bound on (4). To simplify notation, let ω_0 denote the b variable. Lower (l_i) and upper (u_i) bounds on each ω_i are iteratively tightened by solving the following MILP problems:

$$\begin{aligned} l_i &= \min\{\omega_i : (w, \xi, z) \in P, Z(w, \xi, z) \leq U\}, \forall i = 0, \dots, d, \qquad (23)\\ u_i &= \max\{\omega_i : (w, \xi, z) \in P, Z(w, \xi, z) \leq U\}, \forall i = 0, \dots, d. \qquad (24) \end{aligned}$$

where, at each step, solutions in P must satisfy all the ω-bounds computed in the previous iterations. In order to limit computing time, MILP problems (23) and (24) are solved within a node limit. This iterative process is applied in a cyclic way until no bound improvement is obtained.

We have tested this approach on the 23 instances from Table 1. First, an initial upper bound U is computed by solving the RLM with a node limit of $100k$ (plus 25 polish nodes). Then, for each lower and upper bound tightening, the MILP problems are solved within a node limit of $10k$. When no bound improvement is obtained, the final RLM is solved with all new bounds on ω variables, unless each variable has lower and upper bounds equal each other. The results are reported in Table 4, where computing times and nodes refer to

Table 3: Computational results for `IBM-Cplex` default and `IBM-Cplex` with local implied bound cuts . Instances of Type B proposed by [5], $n = 100$, executed using 4 threads on an Intel Xeon E3-1220V2 at 3.10 GHz.

		default				local implied bound cuts			
				% gap				% gap	
	optimal value	time (sec.)	nodes	ub	lb	time (sec.)	nodes	ub	lb
1	157,994.959	×	14,751,130	–	0.45	118.09	1,087,597	–	–
2	179,368.534	×	23,937,160	–	18.07	2,403.80	23,626,498	–	–
3	220,673.592	×	26,815,311	–	35.08	×	12,614,389	–	31.69
4	5,225.994	699.49	24,047,360	–	–	137.69	1,124,760	–	–
5	5,957.083	×	23,669,434	–	22.59	×	19,269,146	–	15.68
6	11,409,617.494	×	22,575,895	–	16.09	×	17,916,288	–	13.05
7	11,409,058.363	×	24,923,091	–	17.65	652.26	10,103,379	–	–
8	10,737,725.660	×	21,271,292	–	9.50	2,894.60	20,226,701	–	–
9	5,705,364.054	×	22,662,326	–	16.16	2,735.92	23,849,291	–	–
10	5,704,804.923	×	25,715,542	–	18.57	×	14,419,022	–	24.45
11	5,369,016.540	×	20,587,001	–	10.65	×	17,361,715	–	14.94
12	2,853,237.334	×	22,655,051	–	16.22	×	19,850,228	–	2.40
13	2,852,678.203	×	25,200,627	–	17.66	563.05	6,833,483	–	–
14	2,684,661.980	×	20,656,438	–	8.99	×	16,340,754	–	12.12
15	1,427,173.974	×	22,523,409	–	15.90	1,422.46	13,136,479	–	–
16	1,426,614.843	×	26,691,246	–	20.29	1,029.83	13,181,911	–	–
17	1,342,484.700	×	21,233,022	–	10.30	1,101.30	14,605,572	–	–
18	714,142.294	×	22,664,594	–	16.33	×	28,391,324	–	5.90
19	713,583.163	×	26,342,005	–	18.17	489.94	6,124,166	–	–
20	671,396.060	×	21,371,127	–	10.29	394.34	5,979,371	–	–
21	357,626.454	×	22,634,311	–	16.05	1,314.65	14,112,522	–	–
22	357,067.323	×	24,273,339	–	17.72	979.23	12,231,339	–	–
23	335,851.740	×	25,632,766	–	12.69	844.96	11,706,613	–	–

the overall scheme, i.e., they include the computation of the initial solution, the iterative bound tightening and possibly the final run. To our pleasant surprise, all instances were solved to optimality in less than one minute, and required less that 30 seconds and 150k nodes on average.

7 Conclusions

We have shown that the nonconvex reformulation of so-called *big-M* constraints and the consequent use of a general-purpose MINLP solver instead of an MIQP solver can lead, surprisingly, to faster computing times for a special class of classification problems. Through a careful analysis of `Couenne`'s features and components we have been able to isolate those that make a difference, namely aggressive bound tightening and iterative strengthening of the McCormick linearization. We have proposed two ways of integrating these ingredients within MILP approaches, both leading to finally being able to computationally solve these classification problems. One of these methods is currently part of

Table 4: Computational results for `IBM-Cplex` default and `IBM-Cplex` enhanced with iterative domain reduction (i.d.r.). Instances of `TypeB` proposed by [5], $n = 100$, executed using 4 threads on an Intel Xeon E3-1220V2 at 3.10 GHz.

		`IBM-Cplex` default				`IBM-Cplex` i.d.r.			
				% gap				% gap	
	optimal value	time (sec.)	nodes	ub	lb	time (sec.)	nodes	ub	lb
1	157,994.959	$\times$	14,751,130	–	0.45	25.60	145,300	–	–
2	179,368.534	$\times$	23,937,160	–	18.07	33.87	148,105	–	–
3	220,673.592	$\times$	26,815,311	–	35.08	29.12	149,472	–	–
4	5,225.994	699.49	24,047,360	–	–	31.03	145,718	–	–
5	5,957.083	$\times$	23,669,434	–	22.59	23.01	148,848	–	–
6	11,409,617.494	$\times$	22,575,895	–	16.09	28.63	148,595	–	–
7	11,409,058.363	$\times$	24,923,091	–	17.65	24.68	145,889	–	–
8	10,737,725.660	$\times$	21,271,292	–	9.50	43.15	146,024	–	–
9	5,705,364.054	$\times$	22,662,326	–	16.16	26.94	147,621	–	–
10	5,704,804.923	$\times$	25,715,542	–	18.57	28.82	145,978	–	–
11	5,369,016.540	$\times$	20,587,001	–	10.65	31.38	145,956	–	–
12	2,853,237.334	$\times$	22,655,051	–	16.22	33.67	147,909	–	–
13	2,852,678.203	$\times$	25,200,627	–	17.66	22.66	145,568	–	–
14	2,684,661.980	$\times$	20,656,438	–	8.99	27.92	146,771	–	–
15	1,427,173.974	$\times$	22,523,409	–	15.90	28.98	148,504	–	–
16	1,426,614.843	$\times$	26,691,246	–	20.29	25.61	145,754	–	–
17	1,342,484.700	$\times$	21,233,022	–	10.30	31.55	156,405	–	–
18	714,142.294	$\times$	22,664,594	–	16.33	28.39	148,358	–	–
19	713,583.163	$\times$	26,342,005	–	18.17	25.13	146,030	–	–
20	671,396.060	$\times$	21,371,127	–	10.29	36.44	146,281	–	–
21	357,626.454	$\times$	22,634,311	–	16.05	28.65	161,660	–	–
22	357,067.323	$\times$	24,273,339	–	17.72	25.74	145,992	–	–
23	335,851.740	$\times$	25,632,766	–	12.69	26.86	145,656	–	–

the arsenal of `IBM-Cplex` 12.6.1.

More generally, we argue that aggressive bound tightening is often overlooked in MILP, while it represents a significant building block for enhancing MILP technology when indicator constraints and disjunctive terms are present. Finally, it is also conceivable that other ingredients that are fundamental in MINLP could prove beneficial for MILP.

Acknowledgements

The research of Fischetti, Monaci and Salvagnin was supported by the University of Padova (Progetto di Ateneo "Exploiting randomness in Mixed Integer Linear Programming"). The research of Fischetti, Lodi, Monaci and Salvagnin was supported by MiUR, Italy (PRIN project "Mixed-Integer Nonlinear Optimization: Approaches and Applications"). The work of Nogales-Gómez was supported by an STSM Grant from COST Action TD1207.

References

[1] E. Andersen and K. Andersen. Presolving in linear programming. *Mathematical Programming*, 71:221–245, 1995.

[2] E. Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[3] 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.

[4] P. Bonami, A. Lodi, A. Tramontani, and S. Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 2015.

[5] J.P. Brooks. Support vector machines with the ramp loss and the hard margin loss. *Operations Research*, 59(2):467–479, 2011.

[6] E. Carrizosa and D. Romero Morales. Supervised classification and mathematical optimization. *Computers and Operations Research*, 40:150–165, 2013.

[7] Cbc, v. 2.9. `https://projects.coin-or.org/Cbc`.

[8] S. Ceria and J. Soares. Convex programming for disjunctive convex optimization. *Mathematical Programming*, 86:595–614, 1999.

[9] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 201–208, 2006.

[10] `Couenne`, v. branch/CouenneClassifier, r1046. `https://projects.coin-or.org/Couenne`.

[11] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[12] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331, 1987.

[13] I. E. Grossmann and F. Trespalacios. Systematic Modeling of Discrete-Continuous Optimization Models through Generalized Disjunctive Programming. *AIChE Journal*, 59(9):3276–3295, 2013.

[14] `Gurobi`, v. 6.0.2. `http://www.gurobi.com`.

[15] `IBM-Cplex`, v. 12.6.1.
`http://www-01.ibm.com/software/integration/optimization/cplex/`.

[16] `Ipopt`, v. 3.9.2. `http://projects.coin-or.org/Ipopt`.

[17] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I - Convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[18] F. Messine. Deterministic global optimization using interval constraint propagation techniques. *RAIRO-RO*, 38(4):277–294, 2004.

[19] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[20] X. Shen, G.C. Tseng, X. Zhang, and W.H. Wong. On ψ-learning. *Journal of the American Statistical Association*, 98:724–734, 2003.

[21] M. Tawarmalani and N.V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications.* Kluwer Academic Publishers, Boston MA, 2002.

[22] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, Z.-H. Zhou, M. Steinbach, D.J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14:1–37, 2007.

[23] `Xpress`, v. 7.8.
`http://www.fico.com/en/products/fico-xpress-optimization-suite`.