# Spatial branching for a special class of convex MIQO problems

Pietro Belotti

Dip. di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Via Ponzio 34/5, Milano, 20133, Italy. ORCID: 0000-0001-6591-6886.

Contributing authors: pietro.belotti@polimi.it;

**Abstract**

In the branch-and-bound algorithm, branching is the key step to deal with the nonconvexity of the problem. For Mixed Integer Linear Optimization (MILO) problems and, in general, Mixed Integer Nonlinear Optimization (MINLO) problems whose continuous relaxation is convex, branching on integer and binary variables suffices, because fixing all integer variables yields a convex relaxation. General, nonconvex MINLO problems, on the other hand, require *spatial branching*, i.e., branching on continuous variables.

While spatial branching could be seen as necessary for the general MINLO class only, we show that when the branching point is carefully chosen, spatial branching can be more effective than its integer counterpart for a special class of problems arising from Support Vector Machines with Ramp Loss (SVMRL), which can be modeled as mixed integer problems with linear constraints and a convex quadratic objective. We present a *strong* spatial branching approach for SVMRL coupled with a procedure to strengthen the continuous relaxation, then report on computational tests on known instances from the literature where our approach yields a significant improvement in solve time.

**Keywords:** Spatial Branching, Mixed Integer Optimization, Support Vector Machines

## 1 Introduction

The branch-and-bound algorithm (BB) is at the core of most algorithms for nonconvex optimization problems [1]. Nonconvexity can arise from modeling constructs: integrality of one or more variables in the class of mixed-integer problems with a *convex*

continuous relaxation, which we refer to as MICO (Mixed Integer Convex Optimization); and from nonlinear, nonconvex constraints or objective function in the MINLO class. The branching operation tackles nonconvexity by removing infeasible solutions: those that violate integrality (MICO) or nonlinear constraints (MINLO).

Given the importance of the branching step, most commercial and open-source solvers for MICO problems implement branching techniques like *strong branching* [2], *pseudocosts* [3], and *reliability branching* [4], or combinations thereof. Some of these have been ported to the MINLO world [5], while other branching techniques such as *violation transfer* [6] have been developed for MINLO specifically.

For MICO problems, branching is needed for integer/binary variables only: assuming the integer variables are have finite lower and upper bounds, there is a finite set of subproblems where all discrete variables are fixed and for which any convex optimization solver can obtain a global optimum. It is therefore unnecessary to branch on continuous variables to ensure termination of BB for solving a MICO problem.

We present a BB variant for a special class of Mixed Integer Quadratic Optimization (MIQO) problems, described in Section 2, where *spatial branching*, i.e., branching on continuous variables, outperforms the default branching rules of a MIQO solver. This is not the first time a MINLO technique is used on a MIQO problem: Section 3 recaps on two MINLO-born techniques that helped solve this problem in the past [7], and that originated from an application in Machine Learning [8]. In Section 4 we propose a new approach to solve this problem: a *strong* spatial branching approach for the selection of a continuous branching variable and its branching point. We report on our computational results in Section 5, where the instances from [7, 8] are used to showcase the advantages of our proposed technique.

## 2 Support Vector Machines with Ramp Loss

We are given a set of $n$ vectors $\boldsymbol{x}^i \in \mathbb{R}^d, i \in [n]$ (here we denote $[n] = \{1, 2, \ldots, n\}$) and a set of $n$ scalars $y_i \in \{-1, 1\}, i \in [n]$. Ideally, one seeks a vector $\boldsymbol{w} \in \mathbb{R}^d$ and a scalar $b$ such that $\boldsymbol{w}^T\boldsymbol{x}^i - b \leq -1$ for all $i \in [n]$ such that $y_i = -1$ and $\boldsymbol{w}^T\boldsymbol{x}^i - b \geq 1$ for all $i \in [n]$ such that $y_i = +1$. Support Vector Machines (SVM) [9] are a well-known paradigm for classification of vectors $\boldsymbol{x}^i, i \in [n]$, and a long line of research works exists for the problem of finding $(\boldsymbol{w}, b)$ for correct classification.

Correct classification of all points requires $y_i(\boldsymbol{w}^T\boldsymbol{x}^i - b) \geq 1$ for all $i \in [n]$. Enforcing all such constraints would likely yield an empty feasible set, so usually a *ramp loss* (RL) penalty term is introduced: the original constraint is relaxed to $y_i(\boldsymbol{w}^T\boldsymbol{x}^i - b) \geq 1 - \delta_i$ after introducing a surplus variable $\delta_i$. The ramp loss penalty term is

$$p(\delta) = \begin{cases} 0 \text{ if } \delta \leq 0 \\ \delta \text{ if } \delta \in (0, 2] \\ 2 \text{ if } \delta > 2. \end{cases}$$

The optimization model associated with the problem above consists in minimizing the sum of all penalty terms, $\sum_{i \in [n]} p(\delta_i)$. To avoid solutions where $\boldsymbol{w}$ and $b$ have large values due to the 1 on the right-hand side, several works in the literature add an $\ell_2$-norm *stabilization* term $\frac{1}{2}||\boldsymbol{w}||_2^2$ to the objective function (or in some cases an $\ell_1$-norm

term [10]), but the emphasis is kept on the penalty part of the objective function by multiplying it by $\frac{C}{n}$ for a large constant $C$. The Ramp Loss penalty terms $p(\delta_i)$ can be implemented with one continuous variable $\xi_i$ and one binary variable $z_i$ for each constraint. Then the Support Vector Machine with Ramp Loss (SVMRL) problem is modeled as follows [8]:

$$\begin{array}{ll} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{z}} & \frac{1}{2}||\boldsymbol{w}||_2^2 + \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2z_i) \\ \text{s.t.} & (z_i = 0) \Rightarrow (y_i(\boldsymbol{w}^T\boldsymbol{x}^i - b) \geq 1 - \xi_i) \quad \forall i \in [n] \\ & (\boldsymbol{w}, b) \in \mathbb{R}^{d+1}, \boldsymbol{\xi} \in [0,2]^n, \boldsymbol{z} \in \{0,1\}^n. \end{array} \tag{1}$$

The main class of constraints, here written as an implication, is made of *indicator constraints*; we outline below three ways of modeling them.

The term $\frac{1}{2}||\boldsymbol{w}||_2^2$ makes the problem a mixed integer quadratic optimization (MIQO) problem. However, the large value of $C$ implies that (1) is, for all practical purposes, a problem where the linear penalty part of the objective function largely determines the solution.

We first introduce a more convenient notation that will be used in the remainder of this paper. Define the $(d+1)$-dimensional variable vector $\boldsymbol{g} := (\boldsymbol{w}, b)$ and the constant $(d+1)$-vectors $\boldsymbol{v}^i := y_i(\boldsymbol{x}^i, -1)$ for $i \in [n]$. We also denote as $\boldsymbol{a}_{[k]}$ the subvector of $\boldsymbol{a} \in \mathbb{R}^p$ containing the first $k \leq p$ entries of $\boldsymbol{a}$. Problem (1) can be rewritten as follows:

$$\begin{array}{ll} \min_{\boldsymbol{g},\boldsymbol{\xi},\boldsymbol{z}} & \frac{1}{2}||\boldsymbol{g}_{[d]}||_2^2 + \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2z_i) \\ \text{s.t.} & (z_i = 0) \Rightarrow (\boldsymbol{g}^T\boldsymbol{v}^i \geq 1 - \xi_i) \quad \forall i \in [n] \\ & \boldsymbol{g} \in \mathbb{R}^{d+1}, \boldsymbol{\xi} \in [0,2]^n, \boldsymbol{z} \in \{0,1\}^n. \end{array} \tag{2}$$

Constraint $(z_i = 0) \Rightarrow (\boldsymbol{g}^T\boldsymbol{v}^i \geq 1 - \xi_i)$ can be formulated in three ways:

1. as the linear, *big-M* constraints $\boldsymbol{g}^T\boldsymbol{v}^i \geq 1 - \xi_i - M_i z_i$ for an appropriate choice of $M_i$ that is, at the same time, small enough to ensure numerical stability but also large enough so that optimal solutions are not excluded;
2. by specifying it as an actual indicator constraint, which is possible in most commercial solvers such as FICO Xpress [11], Gurobi [12], and IBM-Cplex [13];
3. as the nonlinear constraint

$$(1 - z_i)(\boldsymbol{g}^T\boldsymbol{v}^i - 1 + \xi_i) \geq 0. \tag{3}$$

The big-$M$ and indicator constraint formulations are best-known in the literature [8, 14, 15] for several reasons, most importantly as a way to use a MILO solver. The nonlinear nonconvex constraint (3) can only be handled by MINLO solvers such as Antigone [16], Baron [6], Couenne [5], SCIP [17], and FICO Xpress [11].

# 3 MINLO techniques for the SVMRL problem

The MIQO model (1) for the problem of finding optimal $(\boldsymbol{w}, b)$ was used in [8], with the big-$M$ formulation for the main class of constraints. Variants of this model have been proposed for *feature selection*, i.e., for constraining the support of $\boldsymbol{w}$ [14, 18, 19].

An efficient solution method based on a MIQO solver was presented in [7]. It uses two basic building blocks of MINLO solvers, which we outline below. As observed in [7], tight bounds on the $\boldsymbol{g}$ variables translate to tighter (i.e., smaller) big-$M$ parameters, with significant impact on performance, as shown for instance in [20] where a technique for tightening big-$M$'s is presented.

The special structure of the SVMRL model makes it amenable for other solution techniques such as rounding heuristics to obtain a feasible solution from a solution to the continuous relaxation [15]. Also, if $n \gg d$, then the sparsity of the coefficient matrix for the $\boldsymbol{\xi}$ and $\boldsymbol{z}$ variables makes Benders decomposition a suitable choice [21].

### 3.1 Locally implied bound cuts

Exact MINLO solvers create a reformulation of a nonconvex nonlinear problem by introducing *auxiliary* variables for all nonlinear operators and creating a Linear Optimization (LO) relaxation comprising all such auxiliary variables. Model (2) is reformulated by introducing two variables $s_i$ and $t_i$ and rewriting constraint (3) as the system

$$\begin{aligned} s_i &= \boldsymbol{g}^T \boldsymbol{v}^i - 1 + \xi_i \\ t_i &= (1 - z_i) s_i \\ t_i &\geq 0. \end{aligned}$$

Then, given lower and upper bounds $\ell_i$, $u_i$ on $s_i := \boldsymbol{g}^T \boldsymbol{v}^i - 1 + \xi_i$, a LO relaxation contains two of the well-known *McCormick inequalities* [22], namely those providing the upper envelope of the product of two variables:

$$\begin{aligned} t_i &\leq \ell_i(1 - z_i) + s_i - \ell_i \\ t_i &\leq u_i(1 - z_i). \end{aligned}$$

Given that $t_i \geq 0$ and $u_i$ is assumed non-negative (as otherwise $\boldsymbol{v}^i$ cannot be classified for any $\boldsymbol{g}$ and thus one can set $z_i = 1$) the second inequality is redundant. Therefore, the LO relaxation can be simply created with only the $s_i$ variables amended with the inequality $\ell_i(1 - z_i) + s_i - \ell_i \geq 0$, or, more succinctly,

$$s_i - \ell_i z_i = \boldsymbol{g}^T \boldsymbol{v}^i - 1 + \xi_i - \ell_i z_i \geq 0,$$

which is just a big-$M$ constraint with $M_i = -\ell_i$, the opposite of the lower bound on $s_i = \boldsymbol{g}^T \boldsymbol{v}^i - 1 + \xi_i$. While big-$M$ constraints are already in the MILO model, the ever-tightening $\ell_i$ with the progressing of the branch-and-bound yields a tighter big-$M$ and thus faster convergence. These cuts are known as *locally implied bound cuts* and their separator is implemented in IBM-Cplex [7].

### 3.2 Optimization-based bound tightening (OBBT)

Given the generic MINLO problem $\min_{\boldsymbol{x}}\{f(\boldsymbol{x}) : \boldsymbol{x} \in X \cap [\boldsymbol{\ell}, \boldsymbol{u}]\}$, where $[\boldsymbol{\ell}, \boldsymbol{u}] \subseteq \mathbb{R}^n$ is a bounding box, it is well known that solvers aiming at finding a global optimum can benefit from tighter bounds $\boldsymbol{\ell}$ and $\boldsymbol{u}$, and several techniques are normally employed by most MINLO solvers. Among them is OBBT, which uses a relaxation $\min_{\boldsymbol{x}}\{f(\boldsymbol{x}) :$

$\boldsymbol{x} \in X_R \cap [\boldsymbol{\ell}, \boldsymbol{u}]\}$ of the problem at hand, with $X_R \supseteq X$. In order to obtain better bounds on $\boldsymbol{x}$, one solves the following $2n$ problems, two for each $x_i$, $i \in [n]$:

$$\begin{aligned} \check{\ell}_i &= \min_x\{x_i : \boldsymbol{x} \in X_R \cap [\boldsymbol{\ell}, \boldsymbol{u}]\} \geq \ell_i \\ \check{u}_i &= \max_x\{x_i : \boldsymbol{x} \in X_R \cap [\boldsymbol{\ell}, \boldsymbol{u}]\} \leq u_i. \end{aligned}$$

While solving $2n$ relaxations is computationally expensive, it has been proven experimentally (see e.g. [5]) that OBBT is an effective tool for improving the performance of a MINLO solver. An effective variant of OBBT used in [7] proceeds as follows:

1. Run a branch-and-bound solve on (2) using a relatively small BB node limit of 100k nodes, which yields a primal bound $\rho$ from an integer feasible solution.
2. Set an even smaller BB node limit of 10k nodes, then run OBBT on all $g_j$ variables using the mixed integer feasible set of (2), further restricted by an upper bound $\rho$ on the objective function: $\frac{1}{2}||\boldsymbol{g}_{[d]}||_2^2 + \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2z_i) \leq \rho$. While the best integer solution found cannot be used because of the BB node limit, the dual bound at the end of each MIP solve is a valid (lower or upper) bound on $g_j$.
3. Relax the BB node limit and run the BB on (2) with $g_j$'s tightened as above.

A baseline algorithm, which we use for comparison with the approach described below, combines locally implied bound cuts with OBBT.

# 4 Spatial Branching

The main idea of this paper is to introduce *spatial branching* on the problem at hand. The idea stems from a simple observation: for the SVMRL problem, the branch-and-bound must choose among all $n \gg d$ binary variables for branching operations, while it is the $\boldsymbol{g}$ variables that, in fact, determine the classification or misclassification of all points of the instance, their violation and, ultimately, the objective function value.

Consider a BB algorithm that can branch both on the $z_i$'s and by using the following disjunction on $g_j$, for some $j \in [d+1]$, with *branching point* $\tau$:

$$g_j \leq \tau \qquad \vee \qquad g_j \geq \tau. \tag{4}$$

At each BB node, the branching decision consists in choosing which, among the $z_i$ and the $g_j$ variables, to branch on. If the latter is chosen, then the branching point $\tau$ must be selected as well. Branching variable and branching point selection have been studied in the context of MINLO [5, 6].

In general, unlike branching on integer variables, spatial branching does *not* rely on the single disjunction (4) only: branching on a continuous variable does not eliminate any solution from the relaxation of the problem. Indeed the union of the feasible sets of the two subproblems obtained from the mere (4) would be the same as the feasible set of the BB node they are created from. The full spatial branching technique for MINLO consists in *propagating* the disjunction (4) to obtain, for example via bound reduction, tighter bounds on other variables. These new bounds in turn yield a tightened LO relaxation. The resulting two subproblems exclude the solution to the continuous relaxation and are therefore valid for the BB scheme.

Spatial branching is hence much more than a single branching operation: it adds stronger linear constraints (whose coefficients depend on the new variable bounds) to the LO relaxation, whose quality, i.e., how good its lower bound is, ultimately depends on a good choice of branching variable and branching point.

Our spatial branching scheme is based on the big-$M$ formulation of problem (2). We rewrite it here with a slight change in the main constraint class:

$$\begin{array}{lll} \min_{\boldsymbol{g},\boldsymbol{\xi},\boldsymbol{z}} & \frac{1}{2}||\boldsymbol{g}_{[d]}||_2^2 + \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2z_i) & \\ \text{s.t.} & \xi_i + M_i z_i \geq 1 - \boldsymbol{g}^T\boldsymbol{v}^i & \forall i \in [n] \\ & \boldsymbol{\xi} \in [0,2]^n, \boldsymbol{z} \in \{0,1\}^n, \boldsymbol{g} \in [\boldsymbol{\ell},\boldsymbol{u}]. & \end{array}$$

Lower and upper bounds $\boldsymbol{\ell}, \boldsymbol{u}$ on $\boldsymbol{g}$ imply lower/upper bounds on all terms $1 - \boldsymbol{g}^T\boldsymbol{v}^i$. For $i \in [n]$, any upper bound on $1 - \boldsymbol{g}^T\boldsymbol{v}^i$ is a valid big-$M$. Therefore, we compute $M_i$ as follows (we denote as $v_{ij}$ the $j$-th component of vector $\boldsymbol{v}^i$, for $j \in [d+1]$):

$$M_i(\boldsymbol{\ell},\boldsymbol{u}) = 1 - \sum_{j\in[d+1]:v_{ij}<0} v_{ij}u_j - \sum_{j\in[d+1]:v_{ij}>0} v_{ij}\ell_j. \tag{5}$$

Because the penalty-related variables $\xi_i$ and $z_i$ are those that most heavily influence the objective function, it is important that, at every stage of the solver process (i.e., at every BB node), such $M_i$ is valid and as tight as possible.

## 4.1 Strong spatial branching on $g_j$

After applying a given branching decision $g_j \leq \tau \vee g_j \geq \tau$, which we represent with the pair $(j,\tau)$, the new bounding boxes (one for each of the two subproblems) are defined as $[\boldsymbol{\ell}, \tilde{\boldsymbol{u}}(j,\tau)] = \{\boldsymbol{g} \in [\boldsymbol{\ell},\boldsymbol{u}] : g_j \leq \tau\}$ and $[\tilde{\boldsymbol{\ell}}(j,\tau), \boldsymbol{u}] = \{\boldsymbol{g} \in [\boldsymbol{\ell},\boldsymbol{u}] : g_j \geq \tau\}$, where $\tilde{\boldsymbol{\ell}}(j,\tau)$ and $\tilde{\boldsymbol{u}}(j,\tau)$ are vectors whose components are as follows:

$$\tilde{\boldsymbol{\ell}}_k(j,\tau) = \begin{cases} \ell_k & \text{if } k \neq j \\ \tau & \text{otherwise} \end{cases} \qquad \tilde{\boldsymbol{u}}_k(j,\tau) = \begin{cases} u_k & \text{if } k \neq j \\ \tau & \text{otherwise} \end{cases} \qquad \forall k \in [d+1].$$

Using a similar observation to what brought to locally implied bound cuts described in Section 3.1, new bounds should be explicitly taken into account by changing the constraints, if any, that depend on them, i.e., the big-$M$ constraints in the model, specifically by tightening the $M_i$ coefficients.

Therefore, lower bounds on the optimal objective value of the two new subproblems could be obtained by solving the two following continuous problems (strong branching schemes usually solve convex relaxations, rather than their integer version, in order to keep the computational cost low):

$$\begin{array}{lll} \beta(j,\tau,\leq) = \min_{\boldsymbol{g},\boldsymbol{\xi},\boldsymbol{z}} & \frac{1}{2}||\boldsymbol{g}_{[d]}||_2^2 + \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2z_i) & \\ \text{s.t.} & \xi_i + M_i(\boldsymbol{\ell}, \tilde{\boldsymbol{u}}(j,\tau))z_i \geq 1 - \boldsymbol{g}^T\boldsymbol{v}^i & \forall i \in [n] \\ & \boldsymbol{\xi} \in [0,2]^n, \boldsymbol{z} \in [0,1]^n, \boldsymbol{g} \in [\boldsymbol{\ell}, \tilde{\boldsymbol{u}}(j,\tau)]; & \end{array}$$

$$\begin{array}{rll}\beta(j,\tau,\geq) = \min_{\boldsymbol{g},\boldsymbol{\xi},\boldsymbol{z}} & \frac{1}{2}||\boldsymbol{g}_{[d]}||_2^2 + \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2z_i) & \\ \text{s.t.} & \xi_i + M_i(\tilde{\boldsymbol{\ell}}(j,\tau),\boldsymbol{u})z_i \geq 1 - \boldsymbol{g}^T\boldsymbol{v}^i & \forall i \in [n] \\ & \boldsymbol{\xi} \in [0,2]^n, \boldsymbol{z} \in [0,1]^n, \boldsymbol{g} \in [\tilde{\boldsymbol{\ell}}(j,\tau),\boldsymbol{u}]. & \end{array}$$

A *strong branching* approach to finding the right branching rule consists in finding $(j,\tau)$ that maximize a *branching score* defined as $\min\{\beta(j,\tau,\leq),\beta(j,\tau,\geq)\}$.

One could think of evaluating $\beta(j,\tau,\leq)$ and $\beta(j,\tau,\geq)$ for $m$ values of $\tau$ in the interval $[\ell_j,u_j]$ for all $j \in [d+1]$. However, this approach would entail solving $2(d+1)m$ quadratic optimization problems and would be even more impractical than pure MICO-based strong branching, where only two continuous problems must be solved for each variable.

Therefore, we simplify the strong branching mechanism by solving the above problems with respect to the $\boldsymbol{\xi}$ and the $\boldsymbol{z}$ variables only. Specifically, for a given solution $(\boldsymbol{g}^*,\boldsymbol{\xi}^*,\boldsymbol{z}^*)$ to the LO relaxation of the problem and for a given $(j,\tau)$, we fix $g_k$ to $\tilde{g}_k := g_k^*$ for $k \neq j$, then fix $g_j$ to

$$\begin{array}{l}\tilde{g}_j := \min\{g_j^*,\tau\} \text{ for the left-branch problem;} \\ \tilde{g}_j := \max\{g_j^*,\tau\} \text{ for the right-branch problem.}\end{array}$$

This simplification ensures that the right-hand side of the big-$M$ constraints is within the bounds of the newly-created subproblem and close enough to the solution to the continuous relaxation of the current BB node.

We then solve the following new branching selection problems:

$$\begin{array}{rll}\eta(j,\tau,\leq) = \frac{1}{2}||\tilde{\boldsymbol{g}}_{[d]}||_2^2 + \min_{\boldsymbol{\xi},\boldsymbol{z}} & \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2z_i) & \\ \text{s.t.} & \xi_i + M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau))z_i \geq 1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i & \forall i \in [n] \\ & \boldsymbol{\xi} \in [0,2]^n, \boldsymbol{z} \in [0,1]^n; & \end{array} \tag{6}$$

$$\begin{array}{rll}\eta(j,\tau,\geq) = \frac{1}{2}||\tilde{\boldsymbol{g}}_{[d]}||_2^2 + \min_{\boldsymbol{\xi},\boldsymbol{z}} & \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2z_i) & \\ \text{s.t.} & \xi_i + M_i(\tilde{\boldsymbol{\ell}}(j,\tau),\boldsymbol{u})z_i \geq 1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i & \forall i \in [n] \\ & \boldsymbol{\xi} \in [0,2]^n, \boldsymbol{z} \in [0,1]^n. & \end{array} \tag{7}$$

Fixing $\boldsymbol{g}$ simplifies the strong branching problem dramatically although it returns a less reliable estimate of the dual bound improvement. In the next section we show that this simplified strong branching still yields significant computational advantage. For now we observe that these new problems are separable and admit a closed-form solution.

**Proposition 1.** *An optimal objective function value for the two problems* (6) *and* (7) *is*

$$\begin{array}{l}\eta(j,\tau,\leq) = \frac{1}{2}||\tilde{\boldsymbol{g}}_{[d]}||_2^2 + \frac{C}{n}\sum_{i\in[n]}\gamma(1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i, M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau))) \\ \eta(j,\tau,\geq) = \frac{1}{2}||\tilde{\boldsymbol{g}}_{[d]}||_2^2 + \frac{C}{n}\sum_{i\in[n]}\gamma(1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i, M_i(\tilde{\boldsymbol{\ell}}(j,\tau),\boldsymbol{u})),\end{array}$$

*where*

$$\gamma(\alpha,M) := \begin{cases} 0 & \text{if } \alpha \leq 0, \\ \alpha & \text{if } \alpha > 0 \wedge M \leq 2, \\ 2\frac{\alpha}{M} & \text{if } \alpha > 0 \wedge M > 2. \end{cases} \tag{8}$$

*Proof.* Problems (6) and (7) become decomposable after fixing the only linking variables $\boldsymbol{g}$. For fixed $\boldsymbol{g} = \tilde{\boldsymbol{g}}$ we have

$$\eta(j,\tau,\leq) = \tfrac{1}{2}||\tilde{\boldsymbol{g}}_{[d]}||_2^2 + \tfrac{C}{n}\sum_{i\in[n]} \begin{array}{ll} \min_{\xi_i,z_i} & (\xi_i + 2z_i) \\ \text{s.t.} & \xi_i + M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau))z_i \geq 1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i \\ & \xi_i \in [0,2],\, z_i \in [0,1]. \end{array}$$

For each $i \in [n]$, three cases arise depending on $\tau$ and $\tilde{\boldsymbol{g}}$:

1. $1-\tilde{\boldsymbol{g}}^T\boldsymbol{v}^i \leq 0$, i.e., point $\boldsymbol{v}^i$ is correctly classified by $\tilde{\boldsymbol{g}}$ and the only linear constraint is redundant; then $\xi_i = z_i = 0$ is an optimal solution with objective function value equal to 0;
2. $1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i > 0$ and $M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau)) \leq 2$; then the solution $(\xi_i, z_i) = (0,0)$ is excluded and the coefficient of $z_i$ in the objective is greater than its coefficient in the only linear constraint, implying that an optimal solution to the linear relaxation is $(\xi_i^*, z_i^*) = (1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i, 0)$, with objective function value equal to $\xi_i^* = 1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i$;
3. $1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i > 0$ and $M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau)) > 2$; then for $z_i$ the ratio of objective function coefficient to constraint coefficient is less than 1, which implies that an optimal solution is $(\xi_i^*, z_i^*) = \left(0, \frac{1-\tilde{\boldsymbol{g}}^T\boldsymbol{v}^i}{M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau))}\right)$ with objective function value equal to $2z_i^* = 2\frac{1-\tilde{\boldsymbol{g}}^T\boldsymbol{v}^i}{M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau))}$.

The extension of the above to $\eta(j,\tau,\geq)$ is straightforward. □

Given that both $\tilde{\boldsymbol{g}}$ and either of the new bounds $\boldsymbol{\ell}, \boldsymbol{u}$ depend on $\tau$, so do $(1-\tilde{\boldsymbol{g}}^T\boldsymbol{v}^i)$ and the two big-$M$'s, i.e., $M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau))$ and $M_i(\tilde{\boldsymbol{\ell}}(j,\tau),\boldsymbol{u})$. Also, by construction of $M_i(\cdot)$ in (5) we have $1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i \leq M_i(\boldsymbol{\ell},\tilde{\boldsymbol{u}}(j,\tau))$ for the left branch and $1 - \tilde{\boldsymbol{g}}^T\boldsymbol{v}^i \leq M_i(\tilde{\boldsymbol{\ell}}(j,\tau),\boldsymbol{u})$ for the right branch (each may obviously have a distinct vector $\tilde{\boldsymbol{g}}$). Then it is easy to show that the function $\gamma(\alpha, M)$ defined in (8), while nonlinear, is continuous for $(\alpha, M) \in \mathbb{R}^2$, because $\alpha > 0$ implies $M > 0$.

## 4.2 Branching algorithm

By fixing $\boldsymbol{g}$ to $\tilde{\boldsymbol{g}}$, we obtain an efficient strong (spatial) branching algorithm that consists in evaluating the $\eta$ function (i) for every $j \in [d+1]$; (ii) for $\tau \in T_j$, to be defined below; and (iii) for both left ($\leq$) and right ($\geq$) subproblems. Let us define $T_j$ as a set of $m$ equally spaced internal points to the interval $[\ell_j, u_j]$ for a given $m$:

$$T_j = \left\{\ell_j + \tfrac{k}{m+1}(u_j - \ell_j)\,\forall k \in [m]\right\}.$$

Therefore, a heuristic branching strategy amounts to finding

$$(j,\tau) \in \text{argmax}_{j,\tau}\{\min\{\eta(j,\tau,\leq), \eta(j,\tau,\geq)\} : j \in [d+1], \tau \in T_j\}.$$

In principle, this heuristic requires $2(d+1)m$ calls to a function that computes $1-\tilde{\boldsymbol{g}}^T\boldsymbol{v}^i$ and $M_i = M_i(j,\tau)$ for $i \in [n]$, which has a complexity of $O(nd)$, for an overall complexity of $O(nmd^2)$. We developed an efficient implementation which keeps intermediate

values of $1 - \tilde{\boldsymbol{g}}^T \boldsymbol{v}^i$ and similar for $M_i$ computations, achieving a slightly better complexity of $O(nmd)$. Even for $m = 50$, which was our choice in the computational tests, it can be argued that this heuristic is far more efficient than computing $\beta(j, \tau, \leq)$ and $\beta(j, \tau, \geq)$ explicitly.

If the estimated gain in dual bound is lower than a given threshold, then our proposed algorithm lets the branch-and-bound solver make its own selection, which will certainly be on one of the $z_i$ variables. Otherwise, for the choice of branching variable $g_j$ and branching value $\tau$, the two subproblems are created so that the respective bound on $g_j$ is changed. We only add *branching constraints*, i.e., tightened big-$M$ constraints, when they are violated by the solution of the parent node, in order to avoid introducing too many extra linear constraints.

# 5 Computational tests

We have implemented the strong spatial branching technique described in the previous section using callbacks from two commercial solvers: FICO Xpress [11] and IBM Cplex [13]. Both solvers use *callbacks* to access and modify the branch-and-bound solution process, specifically to generate user cuts and user branching rules. We only present below the results we obtained from using the callback library of IBM-Cplex v22.1 with *locally implied bound cuts* activated; we performed tests using FICO Xpress v9.2 as well, but we do not report on this solver here for two reasons: (i) we obtained almost equivalent results in terms of the performance improvement with our strong spatial branching scheme; (ii) we aimed at including *locally implied bound cuts* (see Section 3.1) in the tests, which FICO Xpress does not have.

We used a computer with an Intel i7-10750H CPU with clock speed 2.6GHz and 32GB RAM memory; all code for setting up the solver and for callbacks was compiled with `gcc` version 11.2 with option '`-O3`'. All tests had a time limit of two hours.

We used the same 23 instances as in [7], which are instances of *type B* with $d = 2$ and $n = 100$ originated from [8]. All instances are available for download from https://github.com/merraksh/data/tree/main/SVMRL. We run a comparison between two algorithms:

*Algorithm A0:* the three-step algorithm described in Section 3.2, which uses OBBT to tighten $\boldsymbol{g}$ variables, plus, for the IBM-Cplex solver, locally implied bound cuts enabled;
*Algorithm SB:* the same as A0 with the addition of the strong spatial branching scheme shown in the previous section.

The extra options included in both algorithms (locally implied bound cuts, OBBT) were chosen after a few preliminary tests in which these options were found to be beneficial for both, and thus to ensure that both algorithms had the best setup.

*Results.* Table 1 summarizes the computational results we have obtained for the 23 instances in the test set. The table shows the following data:

- timeOBBT: The time spent in steps 1,2 of the algorithm outlined in Section 3.2 for tightening bounds on $\boldsymbol{g}$;
- timeBB, nodes: for both A0 and SB, the wall-clock time and the number of BB nodes for the final branch-and-cut solver, after $\boldsymbol{g}$ bounds are tightened;

| | | A0 | | SB | | Ratio | |
|---|---|---|---|---|---|---|---|
| Instance | timeOBBT | timeBB | nodes | timeBB | nodes | timeBB | nodes |
| 1 | 8.65 | **2.98** | 13524 | 5.32 | 4511 | 0.56 | 3.00 |
| 2 | 9.99 | 32.37 | 294411 | **4.37** | 3593 | 7.41 | 81.94 |
| 3 | 9.45 | 224.24 | 940931 | **5.66** | 4074 | 39.62 | 230.96 |
| 4 | 9.09 | **1.35** | 10310 | 4.64 | 4045 | 0.29 | 2.55 |
| 5 | 8.64 | **2.54** | 14322 | 4.15 | 3392 | 0.61 | 4.22 |
| 6 | 7.67 | 725.46 | 7273236 | **7.22** | 6085 | 100.48 | 1195.27 |
| 7 | 10.26 | 89.61 | 824963 | **5.76** | 4877 | 15.56 | 169.15 |
| 8 | 9.39 | 55.04 | 913255 | **8.28** | 7755 | 6.65 | 117.76 |
| 9 | 9.26 | 329.12 | 3612792 | **13.95** | 14186 | 23.59 | 254.67 |
| 10 | 9.63 | 160.52 | 1367008 | **5.56** | 4499 | 28.87 | 303.85 |
| 11 | 9.57 | 23.84 | 239005 | **7.04** | 6537 | 3.39 | 36.56 |
| 12 | 8.44 | 41.56 | 325585 | **11.39** | 10959 | 3.65 | 29.71 |
| 13 | 9.15 | 81.66 | 787430 | **5.04** | 4057 | 16.20 | 194.09 |
| 14 | 8.79 | 9.04 | 34997 | **6.51** | 5552 | 1.39 | 6.30 |
| 15 | 8.73 | 116.43 | 1391074 | **8.81** | 8673 | 13.22 | 160.39 |
| 16 | 5.97 | 188.99 | 1593382 | **13.10** | 14011 | 14.43 | 113.72 |
| 17 | 6.74 | 605.07 | 8476978 | **24.41** | 25489 | 24.79 | 332.57 |
| 18 | 7.67 | 1120.04 | 10033031 | **7.53** | 5961 | 148.74 | 1683.11 |
| 19 | 7.81 | 83.92 | 614554 | **11.44** | 11185 | 7.34 | 54.94 |
| 20 | 6.49 | 448.93 | 5492667 | **20.39** | 21822 | 22.02 | 251.70 |
| 21 | 9.04 | 363.32 | 3584341 | **6.12** | 5347 | 59.37 | 670.35 |
| 22 | 9.33 | 168.99 | 1327778 | **10.69** | 9710 | 15.81 | 136.74 |
| 23 | 8.13 | 100.99 | 808066 | **17.05** | 18021 | 5.92 | 44.84 |

**Table 1** Comparison of all instances of type B from [8], also used in [7]. All instances are numbered from 1 to 23 (column 1). Column *timeOBBT* shows the times (in seconds) spent in the OBBT phase, which is common to both algorithms A0 and SB. Then for each of the two algorithms two columns *timeBB* and *nodes* show the time spent by the subsequent branch-and-cut and the number of BB nodes. The two columns under *Ratio* show the ratios between A0 and SB of both branch-and-bound time and number of BB nodes, and indicate the performance gain obtained with SB.

- Ratio: The ratio of BB time of A0 to that of SB, and similar for BB nodes; the higher the ratio, the more advantageous SB is.

Excluding the easiest of these 23 instances, the total solution time improves significantly with our strong spatial branching. The ratio w.r.t. BB nodes is about an order of magnitude larger than the time ratio. An explanation of the different ratios is perhaps the overhead of handling subproblems with the additional branching constraints (with tighter big-$M$'s) and perhaps the time taken by the procedure for branching variable and branching point selection.

We have also carried out computational tests using the branching technique described in the previous section where the branching rule only contained the disjunction $g_j \leq \tau \vee g_j \geq \tau$ and not the additional tightened big-$M$ constraints. A comparison of our variant of the branch-and-cut to the standard solver A0 was not impressive: for all instances at hand, the spatial branching variant timed out and exhibited a very slow increase in dual bound. The performance is below par even if we use the indicator constraint formulation, and it confirms that spatial branching must combine a simple branching rule with the derived branching constraints on both subproblems, unlike MICO solvers where only the single variable branching rule is enforced.

We conclude this section with a remark on performance. For all instances we tested, $d = 2$, hence $\boldsymbol{g}$ has support 3; several other classes of classification instances have $d \leq 10$. While any $\boldsymbol{g}^*$ yields a feasible solution (after computing the corresponding value for $\boldsymbol{z}$, $\boldsymbol{\xi}$), finding a solution with a good objective function value is far from trivial. Yet both MICO solvers we have tested find a good feasible solution at the beginning of the BB process.

The objective function value associated to these solutions can be used to reduce bounds on $\boldsymbol{g}_{[d]}$: a cutoff of $\rho$ implies that $\frac{1}{2}||\boldsymbol{g}_{[d]}||_2^2 \leq \frac{1}{2}||\boldsymbol{g}_{[d]}||_2^2 + \frac{C}{n}\sum_{i\in[n]}(\xi_i + 2y_i) \leq \rho$, hence $g_j \leq 2\sqrt{\rho}\,\forall j \in [d]$. For some instances we observed solutions of objective function value around 10,000, which yields bound intervals $[-200, 200]$.

Observe that the BB algorithm for A0 takes, in some cases (instances 6, 9, 10, 15-18, 19, 20-22), millions of BB nodes to find an optimal solution. Instead of this many node solves and large CPU time, one could obtain a solution in a much shorter time by means of a *brute-force* algorithm (which is also very easily parallelized): by creating a grid of $k \times k$ boxes in the $\boldsymbol{g}_{[d]}$ space, with $k = \lceil\sqrt{q}\rceil$ and $q$ the number of BB nodes, one obtains a much easier set of $q$ problems where $\boldsymbol{g}_{[d]}$ is almost fixed and where the remaining variables $g_{d+1}$, $\boldsymbol{\xi}$, and $\boldsymbol{z}$ can be easily determined.

# 6 Open questions and ongoing research

This paper presents a solution technique for a class of MIQO problems arising in the fields of Support Vector Machines with Ramp Loss, which are used for classifying $d$-dimensional vectors with a binary attribute using $n$ training vectors. Our proposed algorithm outperforms the state-of-the art for MIQO-based solvers of SVMRL, in some cases by a large margin. This result is perhaps due to all instances having $d \ll n$ and the problem structure where all $d$ continuous weight variables effectually *determine* the value of all other $2n$ variables, which in turn heavily impact the objective function.

Perhaps the most striking fact is that after a branching rule tightens the bounds on such important variables within a new subprolem, the solver makes no use of it unless the branching rule is amended with a set of valid branching inequalities. This observation suggests that in MICO solvers (i) this task is only performed after an integer branch, which alone suffices to exclude a portion of the LO relaxation's feasible set, or (ii) MICO solvers delegate to the LO solver the treatment of bounds on continuous variables.

A few research directions are possible, the first two currently being pursued:

*Extension to general MICO problems:* If a problem $\min\{\boldsymbol{c}^T\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}\}$ has a small subset of variables that *drive* other variables (integer or continuous), then an approach similar to that of Section 4 can be applied.
*SVM with bounded support in $\boldsymbol{g}$:* Branching on the weight variables $g_j$ allows for controlling the *support* of the $\boldsymbol{g}$ vector, which is useful in SVM applications with feature selection [14, 18, 19].
*Bound reduction:* When should a MICO solver apply bound reduction on continuous variables? Some solvers already apply OBBT to MICO problems, but new bounds on continuous variables seem to be overlooked in new branch-and-bound subproblems.

In general, this article brings yet another example of MICO and MINLO algorithms that can be successfully applied across the divide. We believe that, while solution techniques for MICO (and especially MILO) have been extended to MINLO, several other effective algorithms that were devised for MINLO could make a difference in MILO solvers.

### Statements and Declarations

*Competing interests.* The author does not have financial or non-financial interests that are directly or indirectly related to this article.

*Data availability.* The data used for all experiments is available for download at https://github.com/merraksh/data/tree/main/SVMRL.

## References


[1] Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica **28**(3), 497–520 (1960)

[2] Applegate, D., Bixby, R., Cook, W., Chvátal, V.: The Traveling Salesman Problem, A Computational Study. Princeton Series in Applied Mathematics, vol. 17. Princeton University Press, Princeton (2006)

[3] Bénichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribière, G., Vincent, O.: Experiments in mixed-integer linear programming. Mathematical Programming **1**, 76–94 (1971)

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

[5] Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optimization Methods & Software **24**(4-5), 597–634 (2009)

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

[7] Belotti, P., Bonami, P., Fischetti, M., Lodi, A., Monaci, M., Nogales-Gómez, A., Salvagnin, D.: On handling indicator constraints in mixed integer programming. Computational Optimization and Applications **65**, 545–566 (2016)

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

[9] Cortes, C., Vapnik, V.: Support-vector networks. Machine learning **20**, 273–297 (1995)

[10] Hess, E.J., Brooks, J.P.: The support vector machine and mixed integer linear programming: Ramp loss SVM with $\ell_1$-norm regularization. In: 14th INFORMS Computing Society Conference (2015). https://doi.org/10.1287/ics.2015.0017

[11] The FICO Xpress Optimizer. https://www.fico.com/en/products/fico-xpress-optimization

[12] Gurobi Optimizer Reference Manual. http://www.gurobi.com

[13] IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/docs/en/icos/22.1.1

[14] Ghaddar, B., Naoum-Sawaya, J.: High dimensional data classification and feature selection using support vector machines. European Journal of Operational Research **265**(3), 993–1004 (2018)

[15] Carrizosa, E., Nogales-Gómez, A., Romero Morales, D.: Heuristic approaches for support vector machines with the ramp loss. Optimization Letters **8**(3), 1125–1135 (2014)

[16] Misener, R., Floudas, C.A.: Antigone: algorithms for continuous/integer global optimization of nonlinear equations. Journal of Global Optimization **59**(2-3), 503–526 (2014)

[17] Vigerske, S., Gleixner, A.: Scip: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. Optimization Methods and Software **33**(3), 563–593 (2018)

[18] Labbé, M., Martínez-Merino, L.I., Rodríguez-Chía, A.M.: Mixed integer linear programming for feature selection in support vector machine. Discrete Applied Mathematics **261**, 276–304 (2019)

[19] Baldomero-Naranjo, M., Martinez-Merino, L.I., Rodriguez-Chia, A.M.: A robust SVM-based approach with feature selection and outliers detection for classification problems. Expert Systems with Applications **178**, 115017 (2021)

[20] Baldomero-Naranjo, M., Martínez-Merino, L.I., Rodríguez-Chía, A.M.: Tightening big $M$s in integer programming formulations for support vector machines with ramp loss. European Journal of Operational Research **286**(1), 84–100 (2020)

[21] Santana, Í., Serrano, B., Schiffer, M., Vidal, T.: Support vector machines with the hard-margin loss: Optimal training via combinatorial Benders cuts. ArXiv **2207.07690** (2022) https://doi.org/10.48550/arXiv.2207.07690

[22] McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. Mathematical Programming **10**, 146–175 (1976)