CANADA EXCELLENCE RESEARCH CHAIR

DATA SCIENCE FOR REAL-TIME DECISION-MAKING

LEARNING A CLASSIFICATION OF MIXED-INTEGER QUADRATIC PROGRAMMING PROBLEMS

Pierre Bonami
Andrea Lodi
G. Zarpellon

December 2017

DS4DM-2017-013

POLYTECHNIQUE MONTRÉAL
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
Pavillon André-Aisenstadt
Succursale Centre-Ville C.P. 6079
Montréal - Québec
H3C 3A7 - Canada
Téléphone: 514-340-5121 # 3314

Learning a Classification of Mixed-Integer Quadratic Programming Problems

Pierre Bonami[1], Andrea Lodi[2], and Giulia Zarpellon[2]

[1] CPLEX Optimization, IBM Spain pierre.bonami@es.ibm.com
[2] École Polytechnique Montréal {andrea.lodi, giulia.zarpellon}@polymtl.ca

Abstract. Within state-of-the-art optimization solvers such as IBM-CPLEX the ability to solve both convex and nonconvex Mixed-Integer Quadratic Programming (MIQP) problems to proven optimality goes back few years, but still presents unclear aspects. We are interested in understanding whether for solving an MIQP problem it is favorable to linearize its quadratic part or not. Our approach employs Machine Learning techniques to learn a classifier that predicts, for a given MIQP instance, the most suitable resolution method within IBM-CPLEX's algorithmic framework. We aim as well at gaining methodological insights about the instances' features leading this discrimination. Together with a new generated dataset, we examine part of CPLEX internal testbed and discuss different scenarios to integrate learning and optimization processes. By defining novel measures, we interpret learning results and evaluate the quality of the tested classifiers from the optimization point of view.

1 Introduction

The tight integration and development of discrete optimization and machine learning (ML) is a recent but already fruitful research theme. The aim of this combination is twofold: on the one side, various are the discrete optimization settings that could benefit from a ML-based heuristic approach; on the other side, ML algorithms themselves could benefit from choices of discrete type, which have been until now disregarded. We focus on the former direction, that one could call "learning for optimization": we believe that learning algorithms could provide new predictive tools to improve the current way decision-making is performed. Especially in those situations where the state-of-the-art choices follow a heuristic-based approach, ML predictive capabilities could be exploited in order to gain significant methodological insights.

Although a number of fresh applications of ML in discrete optimization is recently appearing, one could identify two main topics in this thread of research. On the one side, ML-based approaches are proposed for the branch-and-bound scheme for solving Mixed-Integer Linear Programming (MILP) problems (see [1] for a survey on the theme); on the other side, predictions are leveraged to deal with the solvers' computational aspects and their configuration (see, e.g., the work of [2] and [3]).

Shifting from those two main ideas and positioning ourselves somehow in between, we tackle a new application of ML in discrete optimization. In particular, in the area of Mixed-Integer Nonlinear Programming (MINLP) we consider Mixed-Integer Quadratic Programming (MIQP), which stands by itself as a prominent class of problems: not only quadratic programs are of interest for modeling diverse practical applications (see, e.g., [4], [5]), they also represent a theoretical ground for a first extension of MILP algorithms into MINLP ones.

Within state-of-the-art optimization solvers such as IBM-CPLEX [6], the ability to solve both convex and nonconvex MIQPs to proven optimality goes back few years (see, e.g., [7]), but resolution methods for this class of problems still present unclear aspects. We are interested in understanding whether it is favorable to linearize the quadratic part of an MIQP or not, in order to solve it most efficiently. Currently, the decision *linearize vs. not linearize* can be specified by CPLEX users via the linearization switch parameter, but it is not so clear when this switch should be turned on or off in order to benefit the resolution process. The idea that perhaps MIQPs should be solved in an intelligent and adapted way was firstly suggested in [8], and naturally calls for a predictive machinery, given the fact that the theoretical and computational knowledge of their resolution methods does not seem fully understood yet.

We interpret the question *linearize vs. not linearize* as a classification one, and tackle it with a ML approach. Our aim is to learn a classifier predicting for a given MIQP the most suited resolution method within CPLEX's framework, and possibly gain methodological insights about the problems features leading to this prediction. The developed framework takes care of all learning-related aspects - such as dataset generation, features design and labeling procedure definition, and we discuss different scenarios to integrate learning and optimization processes. We do not forget the peculiar optimization point of view in the development and in the evaluation of our procedure, and define novel measures to interpret the learning results in optimization terms.

2 Optimization Background

We consider general MIQP problems, i.e., optimization problems in which a quadratic objective function is minimized over a set of linear constraints, and (a share of) bounded variables are required to be binary. Formally, an MIQP can be written as

$$\min \quad \frac{1}{2}x^T Qx + c^T x \tag{1}$$

$$Ax = b \tag{2}$$

$$x_j \in \{0,1\} \quad \forall\, j \in I \tag{3}$$

$$l \leq x \leq u, \tag{4}$$

where $Q = \{q_{ij}\}_{i,j=1,\dots,n} \in \mathbb{R}^{n\times n}$ is a real symmetric matrix that together with $c \in \mathbb{R}^n$ defines the objective function (1), while the set of linear constraints (2)

is formed by the matrix $A \in \mathbb{R}^{m \times n}$ and the right hand side vector $b \in \mathbb{R}^m$. Variables $x \in \mathbb{R}^n$ are bounded by lower and upper bounds $l, u \in \mathbb{R}^n$ in (4), and $I \subseteq N = \{1, \ldots, n\}$ in (3) is the set of indices of variables that are required to be binary. We say the problem is *pure (binary)* when $I = N$, and *mixed* otherwise. Note that we do not consider the continuous case, i.e., the case of $I = \emptyset$. We refer to an MIQP *relaxation* as the continuous version of the same problem, where integrality requirements (3) are dropped.

The complexity class of MIQP problems is that of $\mathcal{NP}$-hardness: clearly, MILP is contained in MIQP as the special case in which no quadratic term is present, and continuous Quadratic Programming (QP) problems are in general $\mathcal{NP}$-hard as well [9], [7].

2.1 Solving MIQPs with CPLEX

Depending on its relaxation being convex or nonconvex, and on the types of variables involved, an MIQP can be tackled in different ways. We briefly go through the alternative algorithmic scenarios to solve MIQPs, focusing on the specific framework provided by the IBM-CPLEX commercial solver. An extensive report can be found in [7].

Convex Problems A relaxed MIQP is *convex* if and only if the matrix Q is positive semi-definite ($Q \succeq 0$). In this case, the MIQP can be solved exploiting the mature technology for convex MINLP problems (see [10] for a comprehensive overview). In particular, in both pure and mixed cases, convex MIQPs can be solved by the nonlinear programming-based branch and bound (NLP B&B) [11], a natural extension of the integer linear B&B scheme [12] in which a QP is solved at each node of the tree (and the root node corresponds to the relaxed MIQP). Another common resolution approach for convex problems is that of Outer Approximation algorithms [13], which are however not implemented in CPLEX for MIQPs.

Nonconvex Problems When the relaxed MIQP is *not* convex, i.e., $Q \not\succeq 0$ is indefinite, more sophisticated resolution techniques are required, and variables' types come to play an import role. If all variables are binary, the nonconvex MIQP can be easily transformed into a convex one by means of augmenting the main diagonal of matrix Q. Exploiting the fact that $x_j = x_j^2$ for $x_j \in \{0, 1\}$, $x^T Q x$ can be replaced by

$$x^T (Q + \rho \mathbb{I}_n) x - \rho e^T x, \tag{5}$$

where $Q + \rho \mathbb{I}_n \succeq 0$ for some suitable $\rho > 0$, $\mathbb{I}_n$ denotes the $n \times n$ identity matrix and e the n^{th} vector with all ones.

Alternatively, instead of performing a convexification, the binary nonconvex MIQP can be linearized and transformed into a MILP. Again using the fact that $x_j = x_j^2$ for $x_j \in \{0, 1\}$, diagonal quadratic terms $q_{ii} x_i^2$ are rewritten as

$q_{ii}x_i$, while bilinear terms $q_{ij}x_ix_j$ are replaced with the so-called McCormick inequalities [14]: a new variable $y_{ij} \geq 0$ is added to represent the product x_ix_j, together with linear constraints

$$x_i + x_j - 1 \leq y_{ij} \text{ if } q_{ij} > 0, \text{ or } y_{ij} \leq x_i, y_{ij} \leq x_j \text{ if } q_{ij} < 0, \tag{6}$$

enforcing that $y_{ij} = x_ix_j$ at all possible binary realizations. In this way, the size of the problem formulation grows in number of variables and constraints, but the resulting model can be solved with standard MILP techniques, i.e., exploiting the power of largely more mature algorithms and software.

For mixed nonconvex MIQPs, applying the above tricks to convexify or linearize is not straightforward, and CPLEX relies on the so-called Spatial branch and bound (see, e.g., [15]) to solve these problems to global optimality. Note that a number of different possibilities can be explored in order to perform linearization and convexification. However, the full discussion of such methods (as of those related to general nonconvex MIQPs) is not within the scope of this paper. For more details, we refer the reader to [7] and the references therein.

3 Linearize vs. Not Linearize

In light of the algorithmic scenarios above, one could rightly assume that the linearization approach discussed for pure nonconvex problems could be beneficial for the convex case as well. Indeed, in the binary case the linearization step reduces the MIQP into a MILP, while in the mixed convex case, one could linearize all bilinear products $q_{ij}x_ix_j$ between a binary $x_i \in \{0, 1\}$ and a bounded continuous variable $l_j \leq x_j \leq u_j$ with a generalization of the McCormick inequalities. Note however that nonzero products between two continuous variables cannot be linearized, and will eventually remain in the formulation, so that a mixed convex MIQP could still be quadratic after linearization, and hence solved with a NLP B&B.

As it is reported in [7], the linearization approach does not dominate in theory the non-linearization one, and as it is pointed out in [16], the decision to linearize is not clear-cut when one considers a wide variety of problems. In this sense, the question *linearize vs. not linearize* qualifies as a good quest for learning techniques. Our goal is to exploit ML predictive machinery to understand, for a given MIQP instance, whether it is favorable to linearize its quadratic part or not. In particular, we aim at learning an offline classifier to answer the question *linearize vs. not linearize*, predicting in an instance-specific way the most suited resolution approach within the CPLEX context. We restrict our focus to three problem types only (namely, *pure convex*, *mixed convex* and *pure nonconvex*). The *mixed nonconvex* case constitutes a different scenario, due to the very different setup of Spatial branch and bound, and should be treated separately. In our three cases of interest, currently, the solver provides the user the possibility to switch the linearization on or off by means of the preprocessing parameter `qtolin`. The default strategy employed by CPLEX is to always perform linearization.

We would also like to exploit the classifier to gain some methodological insights about which features of the problems influence the decision-making the most. The offline and supervised nature of the chosen paradigm seems suited for the task. Indeed, we believe that (some of) the reasons leading to an algorithmic discrimination might be linked to the formulation characteristics and the early stages of the optimization of an MIQP problem, and could hence be detected by a learning algorithm if enough relevant information is provided in terms of features. We summarize in what follows the main steps that are undertaken in the development of our method, leaving the details for the next section.

3.1 Development Overview and Contributions

The whole development process can be divided into four main phases:

1. Dataset generation: a dataset bigger than the traditional optimization ones is crucial to implement and test a meaningful learning procedure. Our first step consists in the definition of a generator of MIQP instances, spanning across various combinations of structural and optimization parameters.
2. Features design: we proceed by identifying a set of relevant features, describing the mathematical characteristics of an MIQP instance as well as its prospective behavior with respect to the alternative resolution methods (linearize and not linearize).
3. Labels definition: in order to provide for each MIQP the correct answer to the question *linearize vs. not linearize*, rigorous procedures are defined to discard inconsistent and unsolved instances; running time is the ultimate compared measure to assess the label for the problem.
4. Learning experiments: we test traditional classifiers like kernelized Support Vector Machines as well as more interpretable algorithms such as ensemble methods based on Decision Trees.

4 Methodological Details

We now go through the development steps sketched above more in details, discussing how the dataset is generated, and features and labels defined.

4.1 Dataset Generation

To build complete MIQP instances, data generation comprises two separate steps. First, real symmetric matrices Q are generated by the MATLAB function `sprandsym` [17], to which desired size n, density d and eigenvalues $\lambda_i, i = 1, \ldots, n$ are specified. The final distribution of entries q_{ij} is nonuniform. Rank and definiteness/indefiniteness of Q are indirectly specified by the spectrum definition, which becomes a crucial step to ensure the creation of heterogeneous instances. We uniformly drawn the rank $k \in [2, n]$. Maximum and minimum eigenvalues are uniformly drawn from prescribed magnitude ranges related to n, and the

Table 1: Constraint set possible composition with respect to problem types. Constraints can be chosen among cardinality (c), simplex (s) or multi-dimensional knapsack (mk) ones.

	c	**s**	**mk**	**c+s**	**c+mk**	**s+mk**	**all**
01 conv	✓		✓		✓		
01 nonc	✓		✓		✓		
mix conv	✓	✓	✓	✓	✓	✓	✓

remaining $k-2$ nonzero eigenvalues are uniformly generated in $[\lambda_{\min}, \lambda_{\max}]$. The quadratic data obtained in this way can then be completed or not with the definition of the linear vector c, again uniformly generated and with density d.

Second, the objective function is completed with optimization data: binary and continuous variables are added in various proportion, depending on Q being positive semi-definite or indefinite, and finally a constraints set is defined. This second generation block is implemented with Python and CPLEX respective API. In particular, we monitor the addition of the following types of constraints:

- a single cardinality constraint, of the form $0 \leq \sum_{j \in I} x_j \leq r$, with $r < |I|$ varying;
- a standard simplex constraint, of the form $\sum_{j \notin I} x_j = 1, x_j \geq 0$;
- a set of η multi-dimensional knapsack constraints of the form $\sum_{j \in I} w_{ij} x_j \leq f_i$, for $i = 1, \ldots, \eta$. We follow the procedure described in [18] to generate weights w_{ij} and capacity coefficients f_i, without correlating them to the objective function.

Combinations of constraints types can appear inside a generated MIQP instance, depending on the variables types, which are in turn conditioned by the definiteness/indefiniteness of Q. Table 1 summarizes the various possibilities; the empty constraint set could be established for all MIQP types, while only mixed convex instances could contain all kinds of constraints.

4.2 Features Design

A (raw) formulation like the one in (1)-(4) cannot be fed directly as input to a learning algorithm, so we need to define and extract features representing what we think are the important pieces of information that could lead the discrimination between the methods we want to compare. In this phase of features design, we reinterpret few ideas from the recent works [19] and [3], since to the best of our knowledge the learning paradigm has never been exploited before for MIQPs.

We depict an MIQP instance in its mathematical, optimization and computational properties, by means of a set of 23 hand-crafted features. The main group consists of *static features*, describing the instance in terms of variables, constraints and quadratic objective function; they are efficiently extracted via Python libraries and the solver's API before any solving (pre)process takes place,

Table 2: Overview and brief description of the complete features set.

#	Group name	Features general description
	Static features	
2	Generic problem type	Size n of the problem, proportion of binary variables on total
2	Constraints matrix composition	Density w.r.t. different types of variables, magnitudes of nnz coefficients
6	Quadratic matrix composition	Variables with nnz coefficients on diagonal w.r.t. different types, ratio of magnitudes of nnz diagonal coefficients, appearance of nnz bilinear terms (*continuous* $\cdot$ *continuous*) and (*binary* $\cdot$ *continuous*)
7	Spectrum description	Shares of positive and negative eigenvalues, value of smallest nnz eigenvalue, ratio of eigenvalues magnitudes, trace and spectral norm of Q
4	Other properties of Q	Density, rank and determinant of Q, a measure of "diagonal dominance"
	Dynamic features	
2	Root node information	Difference of lower bounds and resolution times at the root node, between methods linearize and not linearize

and are intrinsic to the problem formulation. An overview and brief description of the entire features set is given in Table 2.

4.3 Labels Definition

Given our question *linearize vs. not linearize*, we need to provide for each MIQP the answer corresponding to the better performing approach. We take into account the possibility of a *tie* between the two methods, and we therefore assign one among three labels $\{\mathsf{L}, \mathsf{NL}, \mathsf{T}\}$.

Following optimization best practices to deal with the solver's performance variability [20], each instance is run five times, with different random seeds. We enforce a timelimit of 1h for each run, and collect data on final upper and lower bounds, resolution times and solver's solution statuses. We implement three checks to monitor troublesome instances.

- Solvability check: at least one method L, NL must find an optimal solution (with tolerance) in order to assign any label; instances that cannot be solved within timelimit by any method for any seed are discarded.
- Seed consistency check: for each seed, lower and upper bounds of L and NL runs are checked for inconsistencies, and unstable instances discarded.
- Global consistency check: if every seed gave consistent results individually, we perform a global check considering the best upper and lower bounds for the two methods. Again, unstable instances are discarded.

Table 3: Composition of dataset $\mathcal{D}$. For each type and label we report the total number of instances and their percentage.

	L	NL	T	**Total**	%
01 conv	195	600	35	830	**0.36**
01 nonc	392	312	39	743	**0.32**
mix conv	11	701	15	727	**0.32**
Total	598	1613	89	**2300**	
%	**0.26**	**0.70**	**0.04**		

If an MIQP passes all these checks, we can decree the winner between L and NL or assign a tie T. When one mode is never able to solve an instance, the solving one is the clear winner. Otherwise, if both L and NL could solve the instance at least once, the performance on each seed is compared and a "seed win" assigned if 10% significantly better in terms of computing time. We assign a winner between L and NL only if seeds wins are consistent through the five runs, opting for a tie otherwise.

5 Data, Experiments and Results

We now analyze the generated dataset and report tested algorithms and settings, considering a share of CPLEX internal testbed as well. Results are interpreted from the optimization point of view by means of new measures of quality of the prediction that take into account the solver's performance.

5.1 Dataset Analysis

The generation procedure is run with MATLAB 9.1.0 (R2016b), Python 2.7 and CPLEX 12.6.3 on a Linux machine, Intel Xeon E5-2637 v4, 3.50 GHz, 16 threads and 128 GB of memory. In order to label the dataset, we used a grid of 26 machines Intel Xeon X5675, 3.07 GHz (12 threads each) and 96 GB of memory; each problem is restricted to one thread only. With our generator of instances we produce 2640 different MIQPs; the size of the problems varies as $n \in \{25, 50, \ldots, 200\}$ and the density of Q as $d \in \{0.2, 0.4, \ldots, 1\}$. For mixed convex MIQPs, the percentage of continuous variables is chosen from $\{0, 20, \ldots, 80\}$. We discard 340 instances due to solvability or consistency failures, ending up with a dataset $\mathcal{D}$ of 2300 problems.

We report in Table 3 the composition of dataset $\mathcal{D}$ with respect to problem types and assigned labels. The dataset is highly unbalanced: the majority of instances is tagged as NL, and only a small share of them as T. Moreover, the NL answer is strongly predominant for mixed convex instances, suggesting that there could be a clear winner method depending on the type of problem itself.

Table 4: Classification measures for different learning settings. The best performing classifiers are boldfaced.

(a) Multiclass - All features

	SVM	**RF**	EXT	GB
Accuracy	85.22	**88.87**	84.00	87.65
Precision	81.91	**85.51**	81.26	84.79
Recall	85.22	**88.87**	84.00	87.65
F1-score	83.16	**87.11**	82.52	86.19

(b) Binary - All features

	SVM	**RF**	EXT	GB
Accuracy	88.43	**91.50**	88.79	91.14
Precision	88.32	**91.39**	88.56	91.04
Recall	88.43	**91.50**	88.79	91.14
F1-score	87.92	**91.33**	88.58	91.07

(c) Multiclass - Static features

	SVM	RF	EXT	GB
Accuracy	**83.48**	83.13	81.91	83.13
Precision	**80.02**	79.81	78.87	79.90
Recall	**83.48**	83.13	81.91	83.13
F1-score	**81.36**	81.17	80.28	81.19

(d) Binary - Static features

	SVM	RF	EXT	GB
Accuracy	**86.80**	86.08	85.53	86.62
Precision	**86.48**	85.69	85.20	86.32
Recall	**86.80**	86.08	85.53	86.62
F1-score	**86.28**	85.53	85.30	86.03

5.2 Learning Experiments and Results

Learning experiments are implemented in Python with Scikit-learn [21], and run on a personal computer with Intel Core i5, 2.3 GHz and 8 GB of memory. We randomly split $\mathcal{D}$ into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, a training and a test sets of, respectively, 1725 (75%) and 575 (25%) instances. Each feature is normalized in $[-1, 1]$, and two static features (namely, the determinant of Q and the ratio of magnitudes between diagonal and out-of-diagonal nonzero coefficients of Q) are removed because of scaling issues. Each experiment comprises a training phase with 5-fold cross validation to grid-search hyper-parameters, and a test phase on the neutral $\mathcal{D}_{\text{test}}$. Unbalance of data is taken into account when splitting $\mathcal{D}$ and defining folds: general proportions of labels are maintained in each subset to ensure that even the less numerous case is always represented. We tested Support Vector Machine (SVM) with RBF kernel [22], together with three ensemble methods based on Decision Trees, namely, Random Forests (RF) [23], Extremely Randomized Trees (EXT) [24] and Gradient Tree Boosting (GB) [25]. We opt for this type of learning algorithms because we would like to get results that are interpretable, particularly in terms of features importance.

As a first experiment, we consider a multiclass scheme, with labels $\{\mathsf{L}, \mathsf{NL}, \mathsf{T}\}$, and employ all defined features (21, given two are removed). Table 4a reports the standard quality measures for classification in this setting: for all four classifiers we compared accuracy, precision, recall, f1-score (averaged and weighted by support among classes, to account for class unbalance).

In this setting, RF is best performing in all measures, though the other classifiers are following up rather close. Overall, and across RF, EXT and GB, the subset of features that appear more influential for the prediction comprises dy-

namic features and information on the spectrum of Q (and hence the convexity or nonconvexity of the problem). An averaged top five list of features includes: difference of lower bounds found by L and NL after the resolution of the root node, and difference of root resolution times; value of the smallest nonzero eigenvalue; a measure of "diagonal dominance", computed as $\frac{1}{n}\sum_{i=1}^{n}(|q_{ii}| - \sum_{j\neq i}|q_{ij}|)$; the spectral norm of Q, i.e., $\|Q\| = \max_i |\lambda_i|$.

We examine the classifiers' confusion matrices to better understand their predictions: a major difficulty seems to be posed by the T class, whose examples are (almost) always misclassified. We assign tie labels in order to track those problems for which methods are comparable, but ultimately we aim at providing a reliable classification of those "extreme" cases for which a change in the algorithmic mode produces a change in the instance being solved or not, so that one could question the relevance of ties with respect to the inquiry *linearize vs. not linearize*. We carry out further experiments in a binary setting, and compare them with the original multiclass one. Moreover, we also test the framework in which classifiers are trained without dynamic features: albeit this may sound counterintuitive with respect to the discussed features evaluation, it is useful to understand how the predictions change without them. Especially, from the optimization solver's point of view, a scenario in which a prediction can be cast without the need of solving twice the root MIQP, but instead directly after reading the original instance, looks appealing and indeed more convenient.

Binary and All Features Setting We remove all tie cases from $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, and we rescale the data accordingly. New binary classifiers are trained, for which we report results in Table 4b. All measures are overall improved, and again RF performs as the best algorithm.

Multiclass and Static Features Setting As reported in Table 4c, all classification measures slightly deteriorate without the information provided by dynamic features, and SVM (with RBF kernel) is the best performing algorithm now. To understand the differences between these "static" classifiers and their original counterparts, we compute in how many cases they are both correct or wrong, equal or not, on the instances of $\mathcal{D}_{\text{test}}$. Overall, classifiers are coherent in their predictions, even in the ones they misclassify; SVM appears as the algorithm that deteriorates the least the original (all features) predictions. About features importance scores, there is no surprise: all scores are slightly bigger than those in the original case, but in a proportional way. Additional features on the spectrum of Q enter the top five list, to join the static ones that were already ranked high.

Binary and Static Features Setting We finally experiment in a setting simplified in terms of both labels and features. We use the binary dataset and discard dynamic features to retrain the classifiers, and obtain results as in Table 4d. Performance is balanced in the improvement brought by the removal of ties,

and the degradation due to the absence of dynamic features. As was in the previous static setting, SVM performs better, while features importance and predictions are in line with the other tested cases.

5.3 Complementary Optimization Measures

Until now, we discussed predictors in terms of traditional classification measures and their changes with respect to different learning scenarios. We now evaluate classifiers in optimization terms: quantifying their performance with respect to the solver's strategy is crucial to determine how effective and valuable our learned approach is for the resolution of MIQPs. Moreover, we look for measurements able to capture the optimization phenomena we guess are happening and conditioning the classification, in order to make learning results meaningful from an optimization standpoint.

We define few "optimization measures" scoring the classifiers with respect to their computational optimization gains, especially in terms of resolution running times. Specifying different values for the `qtolin` parameter, we run each instance i of $\mathcal{D}_{\text{test}}$ in three different modes - CPLEX default strategy (DEF), L and NL - to collect runtimes t^i_{DEF}, t^i_{L} and t^i_{NL}. Each problem is run on one seed only, again with timelimit of 1h, and we consider the multiclass and all features scenario. Instances that are not solved by all modes are removed, and we remain with 529 problems in $\mathcal{D}_{\text{test}}$.

For each classifier clf, we associate a vector of predicted times t_{clf} to the vector of its predicted labels y_{clf}: for every sample $i \in \mathcal{D}_{\text{test}}$, we look at the prediction y^i_{clf} and select the corresponding runtime among t^i_{L} or t^i_{NL}, choosing their average if a tie was predicted. Similarly, we build best and worst times vectors t_{best} and t_{worst}, selecting, respectively, runtimes corresponding to the correct label of the samples and their opposite. Note that we do not have a vector of labels for DEF, but its times vector corresponds to the t_{DEF} already collected.

Sum of Predicted Runtimes We get a first sense of how much faster a predicted resolution method would be compared to the default one by summing over t_{clf} and looking at the ratios between times sums of clf and DEF with respect to *best* and *worst* strategies. Formally, we compare

$$\sigma_{\text{clf}} \coloneqq \sum_{i \in \mathcal{D}_{\text{test}}} t^i_{\text{clf}} \tag{7}$$

for $clf \in \{\text{SVM, RF, EXT, GB}\}$ with σ_{best}, σ_{worst} and σ_{DEF}. Results are reported in Table 5a: RF proves as the closest to the *best* strategy and the farthest from *worst*. The CPLEX default strategy DEF could take up to 4 times more time to run MIQPs in $\mathcal{D}_{\text{test}}$, compared to a trained classifier that selects the most suitable approach. Note that the real gain in time could be even bigger than what showed by Table 5a, given that we stopped the test runs at 3600 seconds and did not let them run indefinitely.

Table 5: Complementary optimization measures. The best performing classifiers are boldfaced.

(a) Multiclass - All features

	SVM	**RF**	EXT	GB	DEF
$\sigma_{\text{clf}}/\sigma_{\text{best}}$	1.49	**1.31**	1.43	1.35	5.77
$\sigma_{\text{worst}}/\sigma_{\text{clf}}$	7.48	**8.49**	7.81	8.23	1.93
$\sigma_{\text{DEF}}/\sigma_{\text{clf}}$	3.88	**4.40**	4.04	4.26	-
$N\sigma_{\text{clf}}$	0.98	**0.99**	0.98	0.99	0.42

(b) Binary - Static features

	SVM	RF	EXT	GB	DEF
$\sigma_{\text{clf}}/\sigma_{\text{best}}$	**1.80**	2.04	2.01	1.82	5.81
$\sigma_{\text{worst}}/\sigma_{\text{clf}}$	**6.23**	5.50	5.59	6.19	1.93
$\sigma_{\text{DEF}}/\sigma_{\text{clf}}$	**3.22**	2.85	2.89	3.20	-
$N\sigma_{\text{clf}}$	**0.98**	0.98	0.98	0.98	0.43

Normalized Time Score A more sophisticated measure takes into account the shifted geometric mean of t_{clf} over $\mathcal{D}_{\text{test}}$,

$$sgm_{\text{clf}} \coloneqq \sqrt[|\mathcal{D}_{\text{test}}|]{\prod_{i \in \mathcal{D}_{\text{test}}} (t^i_{\text{clf}} + 0.01)} - 0.01, \tag{8}$$

which is then normalized between *best* and *worst* cases as

$$N\sigma_{\text{clf}} \coloneqq \frac{sgm_{\text{worst}} - sgm_{\text{clf}}}{sgm_{\text{worst}} - sgm_{\text{best}}} \in [0, 1]. \tag{9}$$

The measure is reported in Table 5a: all predictors are very near 1 (as expected, since this score highly reflects classification performance), while CPLEX default strategy is almost halfway between *best* and *worst* case scenarios.

Going further, we examine the presence of timelimiting cases in the test runs: conferring meaning to our question, both mode L and NL do indeed reach the limit without finding a solution (38 and 55 times, respectively), remarking the fact that one method does not dominate the other one. The presence of timelimits is well reflected in the optimization scores defined above, which are better for classifiers hitting timelimits less frequently: RF reaches the limit 3 times only, while DEF 39 times. Note that due to variability even *best* hits the timelimit once; *worst*, though, does it in 92 cases.

We compute σ_{clf} and $N\sigma_{\text{clf}}$ in the Binary - Static features setting as well. Results in Table 5b are in line with the previous ones, and the slight deterioration in classification performance that we observed in Table 4d is also reflected by the scores.

5.4 Preliminary Experiments on the CPLEX Test Set

We now consider a share of CPLEX internal MIQP testbed: we select problems with size $n < 500$, extract features and compute labels. We remove troublesome instances and remain with a dataset $\mathcal{C}_{\text{test}}$ of 175 instances, which is summarized in Table 6. Note that $\mathcal{C}_{\text{test}}$ is unbalanced with respect to problem types and with respect to the assigned labels. Surprisingly, the majority class is that of ties,

Table 6: Composition of dataset $\mathcal{C}_{\text{test}}$. For each case we report the total number of instances and their percentage.

	L	NL	T	**Total**	%
01 conv	12	1	16	29	**0.16**
01 nonc	40	4	17	61	**0.35**
mix conv	10	3	72	85	**0.49**
Total	62	8	105	**175**	
%	**0.35**	**0.05**	**0.6**		

Table 7: Complementary optimization measures for $\mathcal{C}_{\text{test}}$. The best performing classifier (other than DEF) is boldfaced.

	SVM	RF	**EXT**	GB	DEF
$\sigma_{\text{clf}}/\sigma_{\text{best}}$	2.55	2.30	**1.72**	2.91	**1.22**
$\sigma_{\text{worst}}/\sigma_{\text{clf}}$	2.00	2.22	**2.96**	1.75	**4.19**
$\sigma_{\text{DEF}}/\sigma_{\text{clf}}$	0.48	0.53	**0.71**	0.42	-
$N\sigma_{\text{clf}}$	0.75	0.90	**0.91**	0.74	**0.96**

followed by L, and with very few NL cases. We study the dataset and compare it to $\mathcal{D}$, to understand differences in features distribution. The difference between the two datasets is plain if we consider the fact that while $\mathcal{D}$ is built to represent a variety of mathematical and optimization parameters, $\mathcal{C}_{\text{test}}$ is dominated by the presence of some very structured combinatorial MIQPs, like Max-Cut and Quadratic Assignment Problems.

We produce some preliminary experiments using $\mathcal{C}_{\text{test}}$ as a test set for the classifiers trained on $\mathcal{D}_{\text{train}}$, although we are aware of the many differences in the two datasets distributions. Given the number of ties, we work with $\mathcal{C}_{\text{test}}$ in the original learning setting only (Multiclass - All features). As expected, results are very bad, resembling those of a random predictor. However, looking at confusion matrices it appears clear that most often misclassification happens in form of a tie case predicted as a NL one.

Nonetheless, scores of complementary optimization measures reported in Table 7 are not discouraging, showing that even though DEF is the best strategy to tackle instances in $\mathcal{C}_{\text{test}}$, trained classifiers are not so far away. In particular, the normalized time score behaves well despite the bad classification performance: given that $\mathcal{C}_{\text{test}}$ contains mostly ties, albeit the misclassification rate is high, the loss in terms of solver's performance is not so dramatic, because the difference of runtimes between methods L and NL is not big in those comparable cases.

6 Conclusions and Ongoing Research

We propose a learning framework to investigate the question *linearize vs. not linearize* for MIQP problems. Classification results are satisfactory and interpretable in terms of features importance. We generate a dataset containing a variety of problems for which the question is significant, and we define novel scoring measures to evaluate classifiers in terms of solver's efficiency, in order to capture and interpret the optimization point of view.

In ongoing and future research, we plan to focus on three main directions.

- Analyze other benchmark datasets: the analysis of public optimization libraries containing MIQPs (e.g., [26]) is crucial to understand how representative the synthetic $\mathcal{D}$ is of commonly used instances. Together with $\mathcal{C}_{\text{test}}$, new instances can be used to form a more meaningful and comprehensive final dataset.
- Identify the best learning scenario: in order to successfully integrate the learning framework with the solver, we need to identify a proper learning setting. We already considered the simplified Binary - Static features one; it could be interesting to perform static features selection based on their correlation with dynamic ones.
- Define a custom loss function: the complementary optimization measures that we propose showed effective in capturing the optimization performance as well as the classification one. We plan to use these and other intuitions to craft a custom loss function to train the learning algorithm, this time tailored to the solver's performance on MIQPs.

References

1. Lodi, A., Zarpellon, G.: On learning and branching: a survey. TOP **25**(2) (Jul 2017) 207–236
2. Hutter, F., Hoos, H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (2010) 186–202
3. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence **206** (2014) 79–111
4. Bienstock, D.: Computational study of a family of mixed-integer quadratic programming problems. Mathematical programming **74**(2) (1996) 121–140
5. Bonami, P., Lejeune, M.A.: An exact solution approach for portfolio optimization problems under stochastic and integer constraints. Operations research **57**(3) (2009) 650–670
6. CPLEX: (2017) http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html.
7. Bliek, C., Bonami, P., Lodi, A.: Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report. In: Proceedings of the Twenty-Sixth RAMP Symposium. (2014) 16–17
8. Fourer, R.: Quadratic optimization mysteries, part 1: Two versions (2015) http://bob4er.blogspot.ca/2015/03/quadratic-optimization-mysteries-part-1.html.

9. Motzkin, T.S., Straus, E.G.: Maxima for graphs and a new proof of a theorem of Turán. Canadian Journal of Mathematics **17** (1965) 533–540
10. Bonami, P., Biegler, L.T., Conn, A.R., Cornuéjols, G., Grossmann, I.E., Laird, C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., et al.: An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization **5**(2) (2008) 186–204
11. Gupta, O.K., Ravindran, A.: Branch and bound experiments in convex nonlinear integer programming. Management science **31**(12) (1985) 1533–1546
12. Land, A., Doig, A.: An automatic method of solving discrete programming problems. Econometrica **28** (1960) 497–520
13. Duran, M.A., Grossmann, I.E.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical programming **36**(3) (1986) 307–339
14. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I — convex underestimating problems. Mathematical Programming **10**(1) (1976) 147–175
15. Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A.: Mixed-integer nonlinear optimization. Acta Numerica **22** (2013) 1–131
16. Fourer, R.: Quadratic optimization mysteries, part 2: Two formulations (2015) http://bob4er.blogspot.ca/2015/03/quadratic-optimization-mysteries-part-2.html.
17. MATLAB: Version 9.1.0 (2016) The MathWorks Inc., Natick, Massachusetts.
18. Puchinger, J., Raidl, G.R., Pferschy, U.: The multidimensional knapsack problem: Structure and algorithms. INFORMS Journal on Computing **22**(2) (2010) 250–265
19. Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., Dilkina, B.: Learning to branch in mixed integer programming. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence. (2016)
20. Lodi, A., Tramontani, A.: Performance variability in mixed-integer programming. In: Theory Driven by Influential Applications. INFORMS (2013) 1–12
21. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12** (2011) 2825–2830
22. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3) (1995) 273–297
23. Breiman, L.: Random forests. Machine Learning **45**(1) (2001) 5–32
24. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine Learning **63**(1) (2006) 3–42
25. Friedman, J.H.: Stochastic gradient boosting. Computational Statistics & Data Analysis **38**(4) (2002) 367–378
26. Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A., Gould, N., Liberti, L., Lodi, A., Misener, R., Mittelmann, H., Sahinidis, N., Vigerske, S., Wiegele, A.: QPLIB: A library of quadratic programming instances. Technical report (2017) Available at Optimization Online.