LEARNING MILP RESOLUTION OUTCOMES BEFORE REACHING TIME-LIMIT

Martina Fischetti
Andrea Lodi
Giulia Zarpellon

November 2018

DS4DM-2018-009

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 MILP Resolution Outcomes Before Reaching Time-Limit

Martina Fischetti
Vattenfall
martina.fischetti@vattenfall.com

Andrea Lodi
Polytechnique Montréal
andrea.lodi@polymtl.ca

Giulia Zarpellon
Polytechnique Montréal
giulia.zarpellon@polymtl.ca

Abstract

The resolution of some Mixed-Integer Linear Programming (MILP) problems still presents challenges for state-of-the-art optimization solvers and may require hours of computations, so that a time-limit to the resolution process is typically provided by a user. Nevertheless, it could be useful to get a sense of the optimization trends after only a fraction of the specified total time has passed, and ideally be able to tailor the use of the remaining resolution time accordingly, in a more strategic and flexible way. Looking at the evolution of a partial branch-and-bound tree for a MILP instance, developed up to a certain fraction of the time-limit, we aim to predict whether the problem will be solved to proven optimality before timing out. We exploit machine learning tools, and summarize the development and progress of a MILP resolution process to cast a prediction within a classification framework. Experiments on benchmark instances show that a valuable statistical pattern can indeed be learned during MILP resolution, with key predictive features reflecting the know-how and experience of field's practitioners.

Introduction

Within the realm of discrete optimization, we consider Mixed-Integer Linear Programming (MILP) problems, of the form

$$\min\{c^T x : Ax \geq b, x \geq 0,\ x_i \in \mathbb{Z}\ \forall i \in \mathcal{I}\}, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c, x \in \mathbb{R}^n$ and $\mathcal{I} \subseteq \{1, \ldots, n\}$ is the set of indices of variables that are required to be integral. We do not assume A, b having any special structure (as it is, e.g., for Traveling Salesman Problem instances). Models like (1) can be used to mathematically describe a number of different real-world problems, and are daily deployed across a wide spectrum of applications – network, scheduling, planning and finance, just to mention a few.

Despite being $\mathcal{NP}$-hard problems, MILPs are nowadays solved in very reliable and effective ways, ultimately based on the divide-and-conquer paradigm of Branch and Bound (B&B) (Land and Doig 1960). State-of-the-art optimization solvers, such as IBM-CPLEX (2018), experienced a dramatic performance improvement over the past decades, due to both hardware and software advances (see, e.g., Achterberg and Wunderling; Lodi). Nonetheless, the resolution of some MILPs can prove to be challenging for solvers, and may require hours of computations, so that the experimental practice of imposing a time-limit (TL) to the MILP resolution process is not only very reasonable, but well established too. However, it would be useful to get a sense of the optimization trends after only a fraction of the specified TL has passed, and ideally be able to tailor the usage of the remaining resolution time in a more strategic and flexible way.

We aim to predict whether a generic MILP instance will be solved before timing out, only relying on information from a first portion of the resolution process. More specifically, given P and a time-limit TL, we look at the partial resolution of P, up to a certain time τ, $0 < \tau < TL$, and ask whether P will be solved to proven optimality within TL. We summarize the partial resolution of P, and exploit machine learning (ML) tools to cast a prediction about it being solved or not before TL. Thus, the prediction we aim at is one that takes as input (a summary of) the evolution of a partial MILP run, up to time τ, and outputs a yes/no response, in the framework of binary classification. Note the inherent difference between our approach and the problem of directly predicting the "difficulty" of a MILP instance – e.g., in terms of runtime prediction, the latter being a common interest for both the optimization and the ML communities since the work of Knuth (1975) (a more recent approach can be found in Hutter et al.).

If, on the one hand, the sequential nature of B&B makes it natural to interpret our question as a sequence classification task, on the other hand the transformation of a stream of data from the MILP resolution process into a valid input for traditional classification algorithms cannot be performed with off-the-shelf techniques. To this end, we design specific features to describe the development and behavior of a MILP run in a quantitative way, taking into account the complex interplay between the solver's components. The broad generality of the proposed features makes them apt to be re-used every time one needs to evaluate the B&B development of a general MILP, thus conferring even more impact to this contribution, especially given that applications of ML to discrete optimization have lately been flourishing. For example, in the context of MILP, ML has been proposed to establish good solver's parametric configurations (Hutter, Hoos, and Leyton-Brown 2010); learn heuristics for B&B (see Lodi and Zarpellon for a survey); choose resolution options (Kruber, Lübbecke, and Parmentier; Khalil et al.; Bonami, Lodi, and Zarpellon), and also predict solution-related outcomes (Fischetti and Fraccaro; Larsen et al.). Our work represents a novel contribution in this thread of research: ML is employed to provide an accurate prediction on the resolution outcome of MILPs, which can readily be implemented within solvers to enable tailored optimization and enhance the comprehension of the resolution process, too often hard to unravel given the solver's complexity. In fact, despite the abundance of data and events in the MILP resolution framework, to the best of our knowledge no statistical analysis presently happens within the solver; in particular, information is not exploited in any structural way via ML algorithms to make decisions. Applying to generic MILP problems and opening new opportunities on the solvers' side, our results affect a broad audience and assume greater methodological relevance for the discrete optimization community.

Background: Solving MILPs

As already mentioned, the resolution of MILPs is fundamentally based on the B&B paradigm. In its basic version, B&B sequentially partitions the solution space of (1) into sub-MILPs, which are mapped into nodes of a binary decision tree. At each node, the integrality requirements $x_i \in \mathbb{Z}$ for variables $i \in \mathcal{I}$ are dropped, and a *linear continuous relaxation* (polynomially tractable) of the sub-problem is solved, providing a valid lower bound to the optimal solution value of the original MILP. On the other hand, feasible solutions of (1) provide upper bound values. Global lower and upper bounds (called *best bound* and *incumbent*, respectively) are maintained throughout the resolution process and smartly used to prune unpromising regions of the feasible space, so that the resulting algorithm is only implicitly enumerating the exponentially many solutions of (1). The normalized difference between global bounds (known as *gap*) allows to measure, at any point in time, the quality of a solution and the progress of the optimization: a MILP is solved when the gap is fully closed, i.e., when it reaches 0, with upper and lower bounds coinciding (up to numerical tolerances). The branching and bounding operations are combined with other solver's building blocks – the cutting planes algorithm (Gomory 1960), presolving, primal heuristics – to form a very rich and interconnected resolution framework (Lodi 2009), in which single events and data become hard to disentangle.

The ability to analyze the outputs of the B&B algorithm can help identifying causes of performance issues, and explaining instance-specific trends (Klotz and Newman 2013). In particular, many indicators interact in describing the progress of the MILP resolution process, and need to be taken into account when casting a prediction about the resolution outcome. To provide a simple example, we plot in Figure 1 basic information from the resolution log of CPLEX, for an "Easy" instance of MIPLIB2010 (Koch et al. 2011). We report the development of the global bounds and the gap, the number of *nodes left* (i.e., the leaves yet to be explored) and the *depth* of the nodes as the algorithm traverses the tree. The interconnection between these figures is, for this easy case, quite clear to observe: for example, an update of the incumbent value naturally reduces the gap, triggers a drop in the number of nodes left (due to pruning by bound), and possibly ends a (depth-first) dive in the tree traversal exploration, a common practice when looking for initial feasible solutions with primal heuristics.

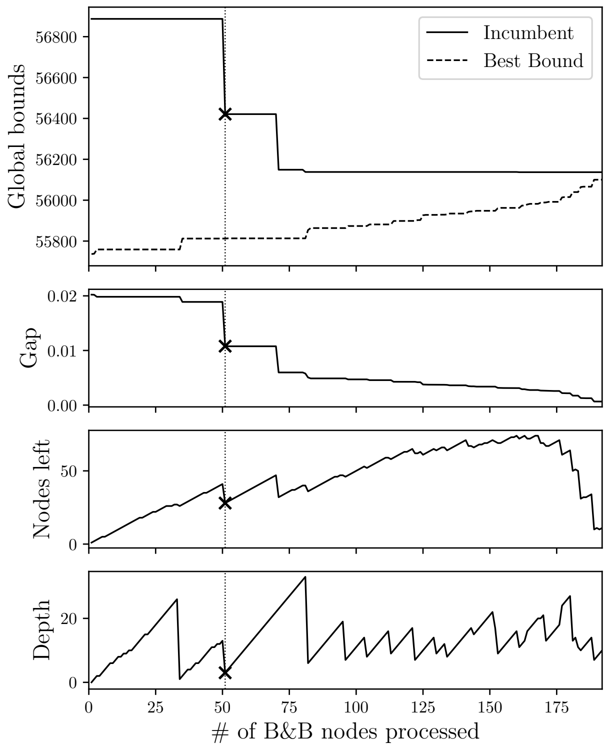


Figure 1: Basic information from the CPLEX log from the resolution of MIPLIB2010 instance `air04`. Interpreting the evolution and interaction of these indicators enables a quantitative description of the optimization process.

Problem Formalization

We can re-phrase our question more formally by considering a MILP P, a time-limit TL, and a certain percentage ratio $\rho \in [0, 1]$ yielding $\tau = \rho \cdot TL \in [0, TL]$. We solve problem P with time-limit TL and take into account the evolution of its resolution process up to time τ. We denote with t^P_{sol} the moment in which P is fully solved (to proven optimality) by the solver. We want to describe and evaluate the progress (in other words, the "work done") in solving P, given that only a share of the total available time has passed; ultimately, we aim at casting a prediction on such description. With respect to the defined parameters, we achieve 100% of work done at t^P_{sol}, and 100% of available time at TL. In practice, there is a discrepancy between t^P_{sol} and TL, the latter specified by a user, the former unknown and subject to variability.

Graphically, one could depict the advancement of the solver with a non-decreasing "progress measure", describing the proportion of work done given the proportion of time passed (Figure 2). Our classification question translates precisely into predicting whether the 100% of the work will be done before TL, i.e., whether $t^P_{sol} \leq TL$, only observing the

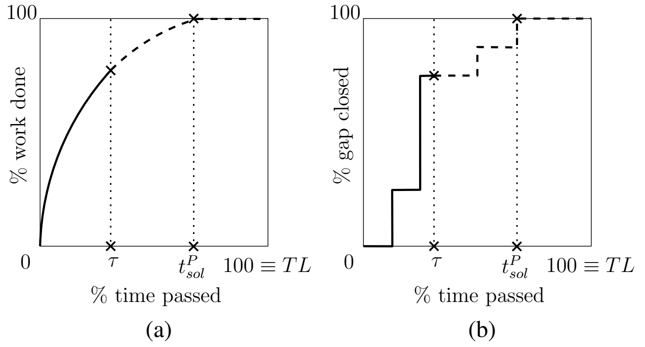


Figure 2: (a) Graphical example of "progress measure" for a triplet (TL, ρ, P); we assume a smooth behavior for drawing purposes. The observed portion of the resolution (up to time τ) is drawn in solid. (b) If we were to measure the progress by looking at the $\%$ of gap closed only, we would draw a step-wise linear function.

resolution up to time τ. The function we aim to learn is thus the indicator function $\mathbf{1}_{\{t^P_{sol} \leq TL\}}$.

The task of features design, on the other hand, aims at providing a definition of the progress measure used to represent the $\%$ of work done, given the triplet (TL, ρ, P). Instead of relying on a single feature to describe the optimization process (as could be done, e.g., using the gap), we try to capture the complexity of MILP resolution by considering heterogeneous measurements, and design a features map Φ, mapping (TL, ρ, P) to a vector in $\mathbb{R}^d$.

Sequence Classification

The sequential character of B&B makes it natural to think about the partial resolution of P as a progressive stream of information and events. In the MILP context, it appears reasonable to discretize the time dimension by considering information being retrieved at every node of the B&B tree, starting from the root and up to the last one being processed before time τ (say η). In other words, one could describe the output of a MILP run with a multivariate time series $\mathcal{S}_{TL,\rho,P}$,

$$\begin{aligned} \mathcal{S}_{TL,\rho,P} = \{ \; & (N^1, \langle v^1_1, \cdots, v^1_s \rangle), \\ & (N^2, \langle v^2_1, \cdots, v^2_s \rangle), \\ & \quad \vdots \\ & (N^\eta, \langle v^\eta_1, \cdots, v^\eta_s \rangle) \; \}, \end{aligned} \tag{2}$$

a sequence of vectors $v^k \in \mathbb{R}^s$, each carrying information about the optimization state at node N^k, up to η.

Classifying $\mathcal{S}_{TL,\rho,P}$ depending on P's optimization outcome can be seen as a *(conventional) sequence classification* task. Sequence classification is typically employed in genomic applications, anomaly-detection and information retrieval (see, e.g., Deshpande and Karypis; Lane and Brodley; Sebastiani, respectively), and generally deals with learning a *sequence classifier* for data of sequential type. Few alternatives to tackle sequence classification can be found in the literature (see the work of Xing, Pei, and Keogh for a brief survey). We opt for a feature-based approach: simply put, we transform the sequence $\mathcal{S}_{TL,\rho,P}$ into a single vector of numerical features $\Phi(TL, \rho, P) \in \mathbb{R}^d$, to which we will then apply traditional classification algorithms. In our setting, a data-point for the learning algorithm consists of a tuple $\big(\Phi(TL, \rho, P), y\big)$ with $\Phi(TL, \rho, P)$ describing the time series data $\mathcal{S}_{TL,\rho,P}$, and binary label $y \in \{0, 1\}$ assigned according to $\mathbf{1}_{\{t^P_{sol} \leq TL\}}$. Our sequence classifier can hence be written as $C : \mathbb{R}^d \longrightarrow \{0, 1\}$,

$$C(\Phi(TL, \rho, P)) = \begin{cases} 1 & \text{if } t^P_{sol} \leq TL, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

However, as pointed out in (Xing, Pei, and Keogh 2010), one of the major challenges when dealing with sequence classification resides in the fact that sequence data does not come with explicit features. Moreover, features selection is usually costly, and needs to account for an interpretable prediction. Off-the-shelf features selection methods – like k-grams or time series shapelets – do not appear suitable to capture the special temporal nature of B&B. We will present features specifically designed for the MILP resolution process after discussing the data collection methodology.

Collecting B&B Data

As we said, the B&B framework produces a lot of heterogeneous information, whose combination can provide interesting insights about the optimization status of a MILP run. Extracting data from the resolution process is allowed by means of implementing custom Callbacks in the solver's APIs, and comes with some computational overhead. From an application perspective, it seems reasonable that a user might be willing to spend some additional resources in the first part of the resolution process, say up to time τ, in order to get a prediction on the more lengthy horizon of TL. Nevertheless, especially in our setting, time is important: any appreciable overhead during the run could bias the yes/no response with respect to the fixed TL, so data collection has to be as cheap as possible.

To comply with the need of collecting non-biased data – and certain that a data collection procedure implemented internally on the solver side would incur in much less overhead than that experienced by any user dealing with its interfaces – we devise a two-step proof-of-concept implementation. We use CPLEX 12.7.1 as solver, together with its Python API. Given (TL, ρ, P), we perform

1. *Label computation*: run P with time-limit TL, and determine a label for the run by checking if $t^P_{sol} \leq TL$. During the run record η, the number of processed nodes at τ.
2. *Data collection*: run again P (the deterministic run of Step 1 can be reproduced by setting the same random seed), and actively collect data during the optimization, up to η nodes.

Having detached data collection from label computation, we do not need to worry anymore about the overhead incurred in Step 2, nor about the integrity of the labeled data;

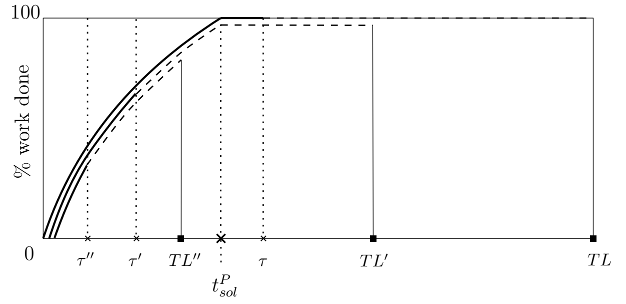


Figure 3: Graphical example of approach (*i*) to obtain multiple data-points from fixed (ρ, P), varying TL. Using additional TL', TL'' we get $\big(\Phi(TL, \rho, P), 1\big)$, $\big(\Phi(TL', \rho', P), 1\big)$ and $\big(\Phi(TL'', \rho'', P), 0\big)$. The observed portions of the resolution process are drawn in solid.

the produced sequence $\mathcal{S}_{TL,\rho,P}$ records the real work done up to the sought fraction ρ of TL.

Producing Diversification

For fixed TL and ρ, a data-point corresponds to a single run of a problem P. The need of a reasonable amount of data for applying ML thus requires many MILP instances – definitely more than those currently part of MILP libraries. Instead of resorting to random problems generation, we try to create additional data from existing benchmark instances.

A first general diversification of data from the same problem P can be produced exploiting the so-called *performance variability* of MILPs (Lodi and Tramontani 2013). Perturbations can be obtained simply by setting different random seeds in the solver, to obtain diverse runs of P. Other diversification schemes, specific to our setting, consist in varying the main parameters TL and ρ. In particular, one could (*i*) vary TL and keep ρ fixed, and/or (*ii*) vary ρ and keep TL fixed. Intuitively, approach (*i*) seems more promising at generating heterogeneous points: a change of TL allows for a sensible re-scaling of τ as well, potentially producing data labeled differently, despite coming from the same problem P. We graphically describe this intuition in Figure 3.

Having discussed how to produce and collect valuable MILP time series data, we now turn to the task of handling it, in order to craft a vector of features.

Features Design

We undertake a features-based approach for sequence classification, and transform MILP sequential data $\mathcal{S}_{TL,\rho,P}$ into a single vector of features $\Phi(TL, \rho, P) \in \mathbb{R}^d$, to be fed as input to traditional classification algorithms. As already mentioned, features selection is not a straightforward process when dealing with serial data, especially if one wants to retain a certain degree of interpretability. We rely on MILP domain-knowledge to define features that shall encompass the optimization progress encoded in $\mathcal{S}_{TL,\rho,P}$.

In practice, we extract 25 raw numerical traits from each Callback call during Step 2 of our data collection procedure, i.e., each vector v^k of $\mathcal{S}_{TL,\rho,P}$ has dimension 25. Note, however, that the length η of the series varies considerably across instances and seeds, ranging between few dozens and hundreds thousands. At each branched node of the MILP tree we collect information about the general state of the optimization (e.g., gap, value of incumbent and best bound, total number of processed nodes and count of simplex iterations performed), together with node-specific data (e.g., current node LP objective value, number of infeasibilities in the LP solution, node depth). At few points in time, we extract information about the list of nodes left (e.g., its length, the maximum and minimum objective estimates, and the number of nodes attaining them). Data traditionally reported in the solver's log are included in these 25 traits.

Few remarks on the nature of the extracted B&B data, and on the guidelines that should be observed to transform them into MILP "progress measures".

1. Some raw information already describe the *global* optimization state, and can be considered in all respects as "progress measures" for the MILP resolution. An example in this sense is provided by the *gap* measure: the last datum collected about the gap refers to the entire resolution process up to that point, and can be used directly as feature in $\Phi(TL, \rho, P)$.
2. Some other information are instead *local*, referring to a particular node LP, and need to be embedded and interpreted within a more broad and global context. For example, a single datum about the *depth* of a node is not informative of the tree evolution, but combined depth data can provide indications about the tree profile, as well as describe dives and leaps in the traversal.
3. Some traits are global (in the sense that they refer to the totality of the optimization process), but are not significant if taken individually. This is the case, for example, of data about the global bounds values, which present themselves as a crude sequence of decreasing (or increasing) scalar values. Measuring their development and changes, instead, can be more informative of the optimization progress.
4. Finally, the wide range of MILP benchmark instances requires features to be comparable across the dataset. For example, exact values linked to parameters $(c, A, b, |\mathcal{I}|)$ and solutions should be avoided. Global counters, e.g., the number of processed nodes, should be used to rescale other indicators, in order not to affect the learning process (and subsequent data normalizations) with data of different magnitudes.

With these guidelines in mind, by means of combining different raw indicators with each other and interpreting them from a development perspective, we design (and select) 37 features to represent the MILP progress. We report an overview and brief description of the features set in Table 1.

Besides the canonical use of statistical functions (like max, min, average) to synthesize some serial information, and the use of *throughputs* measures (e.g., to infer the rates at which nodes are processed and pruned), we apply our domain-knowledge to summarize the optimization progress.

Count	**Group name**	**Features general description**
7	Last observed global measures	Gap, global bounds ratio, fraction of nodes left attaining max/min objective estimate, comparison of max/min estimates with incumbent, primal-dual integral
4	Nodes left and pruned, iterations count	Throughput of pruned nodes, comparison with nodes left, trend w.r.t. max observed # of nodes left, simplex iterations throughput
4	Node LP integer infeasibilities (iinf)	Max/min/avg number of observed iinf, fraction of nodes with iinf below 5% quantile value
5	Incumbent	Throughput of incumbent updates, average frequency and improvement of updates (normalized), distance from last observed update (normalized), was an incumbent found before an integer feasible node (boolean)?
4	Best bound	Throughput of best bound updates, average frequency and improvement of updates (normalized), distance from last observed update (normalized)
3	Node LP objective	Fraction of nodes with objective above the 95% quantile value, normalized differences between quantile threshold and global bounds
4	Node LP fixed variables	Fraction of max/min observed # of fixed variables, fraction of nodes with # of fixed variables above 95% quantile value, normalized distance from last observed peak
6	Depth and tree traversal	Comparison of max observed depth with # of processed nodes, normalized height of last full level and waist of the tree, average length of dives (normalized), frequency of leaps in the traversal

Table 1: Overview and brief description of the 37 features employed for learning experiments.

For example, we tackle measures that can vary significantly even between consecutive nodes in the B&B tree, but for which we are interested in localizing extreme behaviors only, by employing quantile values as statistically meaningful thresholds. We use them to track peaks for values of node LP *objective*, number of *integer infeasibilities* and number of *fixed variables*. Instead, for data that is updating throughout the optimization process (e.g., for *incumbent* and *best bound* values), we focus on interpreting their changes in time, deduce how often and how distant are updates happening, and what is their average improvement.

Experimental Results

Dataset Composition and Setup We employ instances of MIPLIB2010 (Koch et al. 2011) and (Mittelmann 2018) for our experiments. An assessment of the distribution of solving times seemed necessary in order to produce a balanced and meaningful dataset. Evaluation runs with 10 different seeds on the MIPLIB2010 *Benchmark* set suggested the use of $TL \in \{3600, 2400, 1200\}$ seconds. A projection of the resulting labels distribution was performed, to select $\rho = 0.2$ (i.e., we stop the observation after 20% of TL).

To build our dataset, we collect B&B data from the following MILP problems:

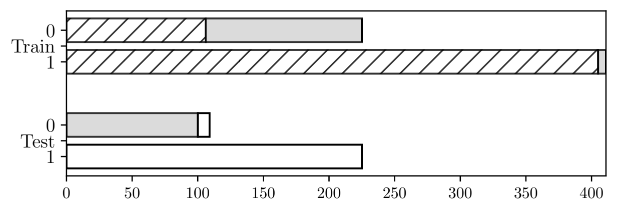
- `Benchmark78`: 78 instances from MIPLIB2010 *Benchmark* set (problems belonging to *Infeasible* and *Primal* subsets are removed, since they do not appear meaningful for our question);
- `Challenge160`: 160 problems from MIPLIB2010 *Challenge* set (with *Infeasible* and *Primal* removed);
- `Mittelmann48`: 48 instances from H. Mittelmann *MILPlib* collection (Mittelmann 2018).

Problems in `Benchmark78` and `Mittelmann48` are solved with 3 different random seeds, while those in `Challenge160` with a single one. As expected, `Mittelmann48` runs are very short, with few cases of time-limiting problems. Counterbalancing this effect, the majority of instances in `Challenge160` cannot be solved within 1h time-limit; `Benchmark78` run times are distributed more evenly. All MILP runs were performed on a cluster of 640 48-cores machines, each equipped with a 2.1GHz Intel Platinum 8160F "Skylake" processor and 192 GB of RAM. Apart from time-limit specifications, we do not modify the solver's default setting; in particular, we leave in place CPLEX default presolve, cuts and primal heuristics.

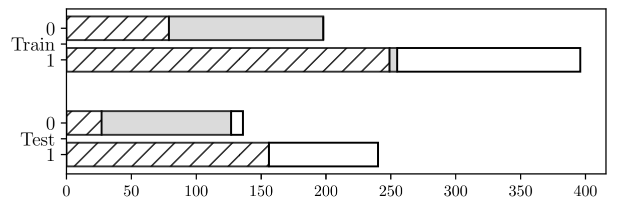
	Class 0	Class 1	**Total (%)**
`Benchmark78`	106	405	511 (52.7)
`Challenge160`	219	6	225 (23.2)
`Mittelmann48`	9	225	234 (24.1)
Total (%)	334 (34.4)	636 (65.6)	**970**

Table 2: Dataset composition in terms of labels and original MILP libraries.

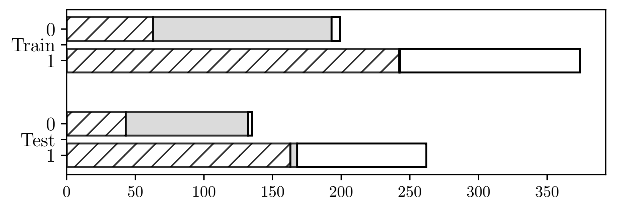
The heterogeneity of the collected time series data makes necessary a thorough phase of data cleaning and scaling. We discard troublesome runs to get 1315 data-points, which then reduce to 970 after computing the hand-crafted features and performing basic data cleaning (data with missing values are removed). Note that a single MILP problem can generate up to 9 different data-points, given the variations in seeds and time-limits used. In the final dataset of 970 points, Class 1 (Class 0) represents the 65.6% (34.4%) of the total; a snapshot of the dataset composition is provided in Table 2.



(a) Non-homogeneous split



(b) Homogeneous split



(c) Random split

Figure 4: Training and test set composition with respect to different labels and MILP libraries, reported for the three considered train-test splits.

Train and Test Splits In order to account for the different composition of MILP libraries and the role of performance variability, we define and try three different ways of splitting our data into training and test set.

1. *Non-homogeneous* split: data-points from `Benchmark78` are used for training, while those from `Mittelmann48` for test; data from `Challenge160` is divided between train and test, taking care of keeping together points arising from the same MILP instance.
2. *Homogeneous* split: both training and test sets are built using a share of each dataset. Again, points arising from the same instance are kept together.
3. *Random* split: data from all runs are mixed together and randomly split. In this case, points that originated from the same MILP instance can appear in both training and test sets.

Proportions between training and test set are roughly maintained around a 60%-40% repartition, with slight variations across splits. Figure 4 illustrates the datasets composition in more detail.

Learning Experiments

We train and test five different learning models, namely, Logistic Regression (LR), Support Vector Machines (SVM) with RBF kernel (Cortes and Vapnik 1995), Random Forest (RF) (Breiman 2001), Extremely Randomized Trees (ExT) (Geurts, Ernst, and Wehenkel 2006), and Multi-Layer Perceptron (MLP). All algorithms are compared against a dummy classifier (dum) following a stratified strategy. The learning phase is implemented entirely in Python with Scikit-learn (Pedregosa et al. 2011), and run on a PC with Intel Core i5, 2.3 GHz and 8 GB of memory. Each feature is normalized to have a mean of 0 and a standard deviation of 1, and each experiment comprises a training phase with 3-fold cross validation to grid-search hyper-parameters, and a test phase on the neutral test set.

	Dum	LR	SVM	**RF**	ExT	MLP
Accuracy	0.55	0.94	0.94	**0.96**	0.96	0.91
Precision	0.56	0.94	0.94	**0.96**	0.96	0.92
Recall	0.55	0.94	0.94	0.96	**0.96**	0.91
F1-score	0.56	0.94	0.94	**0.96**	0.96	0.90

(a) Non-homogeneous split

	Dum	LR	SVM	RF	**ExT**	MLP
Accuracy	0.59	0.90	0.91	0.94	**0.95**	0.86
Precision	0.58	0.91	0.91	0.94	**0.95**	0.86
Recall	0.59	0.90	0.91	0.94	**0.95**	0.86
F1-score	0.59	0.90	0.91	0.94	**0.95**	0.85

(b) Homogeneous split

	Dum	LR	SVM	**RF**	ExT	MLP
Accuracy	0.57	0.93	0.94	**0.94**	0.93	0.93
Precision	0.57	0.93	0.94	**0.95**	0.94	0.93
Recall	0.57	0.93	0.94	**0.94**	0.93	0.93
F1-score	0.57	0.93	0.94	**0.94**	0.93	0.93

(c) Random split

Table 3: Classification results for the three considered train-test split settings. Best scores and classifiers are bold-faced.

		Predicted							
		0	1		0	1		0	1
True	0	100	9	0	125	11	0	130	5
	1	3	222	1	12	228	1	18	244

(a) Non-homog. (b) Homogeneous (c) Random

Table 4: Confusion matrices (w/o normalization) for RF in different split settings. Note that support sizes are varying.

Results Table 3 reports the standard performance measures for binary classification: for all classifiers we compare accuracy, precision, recall and f1-score, the last three metrics averaged between classes and weighted by supports. Overall, RF and ExT are the best performing models, with SVM

Rank	Score (avg)		Feature description
1	0.1856	$*$	Throughput of pruned nodes (over number of processed nodes)
2	0.1839	$*$	Ratio of pruned nodes over last measured number of nodes left
3	0.0805	$*$	Last measured number of nodes left over maximum number of nodes left observed
4	0.0758	$*$	Fraction of nodes attaining max objective in the list of nodes left
5	0.0632	$*$	Fraction of nodes attaining min objective in the list of nodes left
6	0.0622	$*$	Frequency of leaps (i.e., changes in depth with absolute value > 1)
7	0.0453	$*$	Frequency of best bound updates
8	0.0324	$*$	Last measured gap
9	0.0196		Ratio between last measured best bound value and best integer value
10	0.0181		Maximum length of observed dives, normalized
11	0.0165		Normalized difference between last measured best bound and value of objective 5% quantile
12	0.0164		Normalized distance from last measured best bound update

Table 5: Subset of features appearing in the top-10s for RF: scores are averaged among split cases; features marked with $*$ appear in the top-10 of each setting.

following close behind. We report confusion matrices for RF in Table 4; selected hyper-parameters (`n_estimators`, `max_depth`) are $(1500, 10), (100, 10), (100, 5)$, respectively. The high accuracy scores obtained in all three train-test settings attest that there is indeed a statistical pattern to be learned during MILP resolution, and that the designed features are capturing it.

Taking a closer look at class-specific precision and recall scores, we note distinct behaviors with respect to different train-test splits. In particular, models in the *Non-homogeneous* case present a sensitivity (i.e., recall for Class 1) being higher than specificity, accompanied by high precision for Class 0. The trend is much less accentuated in the *Homogeneous* setting, and blurs completely (if not reverses itself) in the *Random* one. An explanation of these behaviors could be linked to the intrinsic difference in composition of the MILP libraries employed for the experiments. In fact, instances in `Benchmark78` do not exhibit clear-cut behaviors as those in `Challenge160` and `Mittelmann48`. Finally, the fact of *Random* being the setting in which MLP is best performing might be a sign of the model being able to recognize akin data-points arising from the same instance (now scattered in both training and test set), and thus linked to the presence of problems with low variability scores.

Features Analysis Our best performing methods, RF and ExT, have the advantage of interpretability. We employ features scores returned by Scikit-learn to provide a first evaluation of those factors that proved valuable for the predictions. We look at the sets of top-10 scoring features for RF, for each train-test split case, and note a very stable scoring pattern: 8 features appear in the top-10 of each setting, and a total of 12 different features covers the three top rankings. We report them in Table 5, where scores have been averaged among cases. In particular, throughputs and trends of nodes pruned, processed and left seem to be crucial for proper classification. Information on the proportions of nodes attaining max and min objective estimates within the list of nodes left are also valuable. Indeed, such estimates at the frontier of the B&B tree are somehow quantifying the amount of work to be done to close the upper and lower bounds in the remaining subtrees, and hence measuring the "difficulty" of what is yet to be explored. Together with the gap, few top-ranked features focus on dives and leaps happened during the traversal, while few others on best bound updates. Note that, despite having provided the same set of features to capture updates of incumbent and best bound, only those relative to the latter are top-ranked by the algorithm. This is in line with the composition of MILP benchmarking libraries and the experience of MILP practitioners, who often witness slow B&B searches due to difficulty in improving the LP bound.

Conclusions and Outlook

We propose a learning approach to predict the outcome of a general MILP problem after only a share of the available computing time has passed. We summarize the sequential MILP resolution process with hand-crafted features, and successfully classify it with traditional learning models. In particular, our novel features can be applied to any type of MILP instance, and hence used in future application of ML for B&B studies, making this work of interest for a wide audience. Our positive results show that there is indeed a pattern to be learned across MILP instances, and represent (to the best of our knowledge) the first structural statistical use of the data provided by the solver throughout the resolution. The proposed framework could be readily implemented internally on the solver side, in order to strategically specialize the optimization process on the fly, before timing out, providing better options for the user. In other words, an early detection of a potential time out can trigger algorithmic changes that, in turn, could prevent such a time out to happen. The developed setting can be extended in a number of different directions. We plan to deepen data analysis – possibly augmenting our dataset – and frame the role of performance variability in the learning process. It would be interesting to consider other ways to tackle sequence classification, e.g., by following a pattern-based approach.

References

Achterberg, T., and Wunderling, R. 2013. *Mixed Integer Programming: Analyzing 12 Years of Progress*. Berlin, Heidelberg: Springer Berlin Heidelberg. 449–481.

Bonami, P.; Lodi, A.; and Zarpellon, G. 2018. Learning a classification of mixed-integer quadratic programming problems. In van Hoeve, W.-J., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 595–604. Springer International Publishing.

Breiman, L. 2001. Random forests. *Machine Learning* 45(1):5–32.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20(3):273–297.

CPLEX. 2018. `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`. Online; accessed 2018.

Deshpande, M., and Karypis, G. 2002. Evaluation of techniques for classifying biological sequences. In Chen, M.-S.; Yu, P. S.; and Liu, B., eds., *Advances in Knowledge Discovery and Data Mining*, 417–431. Springer Berlin Heidelberg.

Fischetti, M., and Fraccaro, M. 2017. Using OR + AI to predict the optimal production of offshore wind parks: A preliminary study. In Sforza, A., and Sterle, C., eds., *Optimization and Decision Science: Methodologies and Applications*, 203–211. Springer International Publishing.

Geurts, P.; Ernst, D.; and Wehenkel, L. 2006. Extremely randomized trees. *Machine Learning* 63(1):3–42.

Gomory, R. 1960. An algorithm for the mixed integer problem. Technical Report RM-2597, The Rand Corporation.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206:79–111.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2010. Automated configuration of mixed integer programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* 186–202.

Khalil, E. B.; Dilkina, B.; Nemhauser, G.; Ahmed, S.; and Shao, Y. 2017. Learning to run heuristics in tree search. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*.

Klotz, E., and Newman, A. M. 2013. Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science* 18(1):18–32.

Knuth, D. E. 1975. Estimating the efficiency of backtrack programs. *Mathematics of Computation* 29(129):122–136.

Koch, T.; Achterberg, T.; Andersen, E.; Bastert, O.; Berthold, T.; Bixby, R. E.; Danna, E.; Gamrath, G.; Gleixner, A. M.; Heinz, S.; Lodi, A.; Mittelmann, H.; Ralphs, T.; Salvagnin, D.; Steffy, D. E.; and Wolter, K. 2011. MIPLIB 2010. *Mathematical Programming Computation* 3(2):103–163.

Kruber, M.; Lübbecke, M. E.; and Parmentier, A. 2017. Learning when to use a decomposition. In Salvagnin, D., and Lombardi, M., eds., *Integration of AI and OR Techniques in Constraint Programming*, 202–210. Springer International Publishing.

Land, A., and Doig, A. 1960. An automatic method of solving discrete programming problems. *Econometrica* 28:497–520.

Lane, T., and Brodley, C. E. 1999. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security* 2(3):295–331.

Larsen, E.; Lachapelle, S.; Bengio, Y.; Frejinger, E.; Lacoste-Julien, S.; and Lodi, A. 2018. Predicting solution summaries to integer linear programs under imperfect information with machine learning. *Preprint arXiv:1807.11876*.

Lodi, A., and Tramontani, A. 2013. *Performance Variability in Mixed-Integer Programming*. INFORMS. chapter 1, 1–12.

Lodi, A., and Zarpellon, G. 2017. On learning and branching: a survey. *TOP* 25(2):207–236.

Lodi, A. 2009. Mixed integer programming computation. In Jünger, M.; Liebling, T.; Naddef, D.; Nemhauser, G.; Pulleyblank, W.; Reinelt, G.; Rinaldi, G.; and Wolsey, L., eds., *50 Years of Integer Programming 1958-2008*. Springer Berlin Heidelberg. 619–645.

Mittelmann, H. D. 2018. MILPlib. `http://plato.asu.edu/ftp/milp/`. Online; accessed 2018.

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.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Sebastiani, F. 2002. Machine learning in automated text categorization. *ACM Computing Surveys* 34(1):1–47.

Xing, Z.; Pei, J.; and Keogh, E. 2010. A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter* 12(1):40–48.