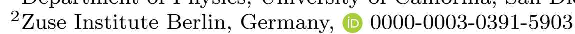
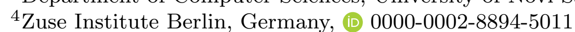
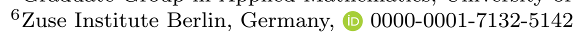
Zuse Institute Berlin

ZIB

Takustr. 7
14195 Berlin
Germany

Alexander Georges [1]
Ambros Gleixner [2]
Gorana Gojić [3]
Robert Lion Gottwald [4]
David Haley [5]
Gregor Hendel [6]
Bartłomiej Matejczyk [7]

Feature-Based Algorithm Selection for Mixed Integer Programming

[1]Department of Physics, University of California, San Diego, USA

[2]Zuse Institute Berlin, Germany, 0000-0003-0391-5903
[3]Department of Computer Sciences, University of Novi Sad, Serbia

[4]Zuse Institute Berlin, Germany, 0000-0002-8894-5011
[5]Graduate Group in Applied Mathematics, University of California, Davis, USA

[6]Zuse Institute Berlin, Germany, 0000-0001-7132-5142
[7]Department of Mathematics, University of Warwick, United Kingdom

ZIB Report 18-17 (April 2018)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Feature-Based Algorithm Selection for Mixed Integer Programming

Alexander Georges, Ambros Gleixner[1], Gorana Gojić, Robert Lion Gottwald[2], David Haley, Gregor Hendel[3], Bartlomiej Matejczyk

April 5, 2018

Abstract

Mixed integer programming is a versatile and valuable optimization tool. However, solving specific problem instances can be computationally demanding even for cutting-edge solvers. Such long running times are often significantly reduced by an appropriate change of the solver's parameters. In this paper we investigate "algorithm selection", the task of choosing among a set of algorithms the ones that are likely to perform best for a particular instance.

In our case, we treat different parameter settings of the MIP solver SCIP as different algorithms to choose from. Two peculiarities of the MIP solving process have our special attention. We address the well-known problem of performance variability by using multiple random seeds. Besides solving time, primal dual integrals are recorded as a second performance measure in order to distinguish solvers that timed out.

We collected feature and performance data for a large set of publicly available MIP instances. The algorithm selection problem is addressed by several popular, feature-based methods, which have been partly extended for our purpose. Finally, an analysis of the feature space and performance results of the selected algorithms are presented.

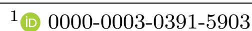
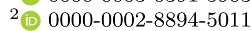
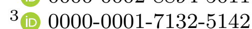
1 Introduction

1.1 Mixed Integer Programming

Throughout this paper we present results derived from instances of mixed integer programming (MIP). For our purposes, a mixed integer program can be represented as:

$$\underset{x}{\operatorname{argmin}} \left\{ c^T x \;\middle|\; b_1 \leq Ax \leq b_2, l \leq x \leq u, x_i \in \mathbb{Z}\ \forall i \in \mathcal{I} \right\}. \tag{1}$$

Here, the variables x_i are constrained to take integer values for all i in the variable subset $\mathcal{I} \neq \emptyset$. An integer variable is called a *binary variable* if, $l_i = 0$ and $u_i = 1$.

[1]  0000-0003-0391-5903
[2]  0000-0002-8894-5011
[3]  0000-0001-7132-5142

Solving a MIP instance (or even determining if a solution exists) is $\mathcal{NP}$-hard in general, and there are a variety of competitive solvers available that use different methods and approaches. These solvers are generally used as stand-alone applications and permit the user to specify a large number of parameters which influence the method/approach that the solver uses. For instance, parameters may affect the time spent in presolving, the aggressiveness with which cutting planes are used or the amount of reliance on branch and bound techniques. In their work, Hutter et al [1] describe one of the leading commercial solvers (CPLEX) as having more than 10^{45} distinct combinations of parameter settings possible (this number has likely changed since the paper was released).

This leads to a number of questions, perhaps the most important of which is what combination of parameters should be used to solve a MIP instance. Arguably, one should use the parameter settings that have the best mean performance on some benchmark set. However, the results that one can achieve in this manner have a limit to their effectiveness, and to overcome this limit one must pay more attention to the properties (features) of an individual instance and allow these features to guide the parameter selection, which is commonly addressed as *Algorithm Selection*.

1.2 Algorithm Selection

The underlying nature of MIPs remains an area of open research, and so is not fully understood in terms of complexity. There is no solver that is optimal for a wide class of problems. While it is possible to choose an algorithm with the best average performance on a representative benchmark set, being able to select from a portfolio of algorithms has been shown to be better in cases where complementary, i.e. fundamentally different strategies are available [2]. In algorithm selection, this idea is taken one step further by constructing custom portfolios using instance-specific features [3]. To this end, we use features specific to the problem instance to select an algorithm (or a portfolio of algorithms) that are likely to perform well on the given instance.

Algorithm selection has been gaining attention in the artificial intelligence community. To aid in this task, recent libraries have been established that compile and organize data for a wide range of $\mathcal{NP}$-hard tasks [4]. While approaches to the algorithm selection problem have been effective for the satisfiability problem, algorithm selection for mixed integer programming has still produced only modest results [3]. It is on this last point that our research is focused.

In the present work, we focus on the algorithm selection problem using various algorithms contained in the SCIP Optimization Suite [5], which is one of the leading non-commercial software packages that can solve MIP instances. We use 14 different combinations of parameters supplied by SCIP. These parameter combinations serve as the algorithms that we will be selecting from. While we restrict ourselves to the collection of parameter settings suggested by SCIP, we note that due to the highly parameterizable nature of the solver, the number of considered algorithms can be easily increased.

We test the different algorithms on two different benchmark sets, one homogeneous and one heterogeneous. General features of the benchmark instances are obtained from the MIP representation (1) to inform our selections and compare the predictive results of different machine learning engines trained on the benchmark runtime data.

The MIP instances within our benchmark sets are supplied in MPS format. We use the SCIP interface to load them and then extract the features, which will be described in Section 2 and in full detail in Appendix A of this document.

1.3 Related Work

Algorithm configuration of MIP solvers has been widely discussed and researched (see [1, 6] and references therein), the main task being to produce solvers that have good average performance on a particular benchmark set. These solvers should then be good predictors of algorithms to use for instances that are similar to those within the benchmark.

Leyton et al. [2] demonstrate the effectiveness of using a portfolio approach in which several algorithms are used either by a schedule or in parallel rather than restricting the solving process to only a single algorithm. This approach has since become commonplace as seen in [3, 7, 8]. We will also use a portfolio approach in our work, selecting more than one algorithm in order to improve performance when the average-best solver does not perform well on a specific instance.

Algorithm selection has proven effective for SAT instances (in particular, for SAT competitions), but has not yet had the same success for MIP instances [8]. Why this is the case is still an open question.

In [6, 9], combinatorial auction problems are used as the benchmark set for testing solver efficiency. Some previous literature has explored the feature space [1] as well as an attempt to use features to select algorithms. However, we desire to understand the problem in the context of a wider benchmark set with a wider representation of possible features. We also implement the algorithm described in [7] together with some variants of the approach.

2 Method

2.1 Description of Data

Every MIP is characterized by a vector of numeric properties, so-called features. The complete feature data is represented as an $n \times m$ matrix $\mathcal{F}$, where n is the number of MIP instances under consideration and m is the total number of features extracted from each problem instance. The performance data (referred to as $\mathcal{P}$) consists of an $5n \times s$ matrix, where s is the total number of settings used. The factor 5 here comes from running each solver on every instance 5 times, each with a different random seed specified on the run. The random seed serves as a final tie-breaker for numerous algorithmic components within SCIP. The intention behind seeded runs is to make the performance evaluation between settings more robust in the light of the phenomenon of performance variability [10, 11]. We take the arithmetic mean of results over these random seeds to smooth out statistical fluctuations produced by each run.

What is more, the data comes from two distinct sets: `Regions` which is a collection of 2000 combinatorial auction instances [9], and a general MIP testset called `M&C` with 713 instances compiled from the publicly available libraries MIPLIB 3 [12], MIPLIB 2003 [13], MIPLIB 2010 [14], and COR@L [15]. Each

of these sets has their own performance data which was computed on an HPC cluster, and we compute feature data independently for each of these sets.

While each instance from the `Regions` set was feasible and reasonably sized, it was necessary to exclude some instances from the `M&C` set. For our pourpose, and in particular for our desired experiments with the primal-dual integral (which does not have substantial qualitative meaning for infeasible instances), we excluded the 93 infeasible instances from the `M&C` library. We also excluded eight instances for which features could not be extracted without exceeding the amount of memory available on a desktop computer. This leaves 612 feasible instances in the `M&C` benchmark set with which we conduct our experiments.

The `M&C` library seems to be more challenging for the machine learning methods as it is a hetoregenous set of problems and in addition, it contains fewer instances than the `Regions` library.

2.1.1 Feature Data

The idea of using features from MIPs for algorithm selection was first explored in [3]. As an example, features can be statistical summaries of the coefficients of the objective function, "right-hand side" vectors and the constraint matrix itself. Features extracted in such a way are generally called 'static' features since they can be extracted without invoking the operant functions of the solver. We use some static features introduced in [3] as a basis for our static feature calculation.

We extract 133 static features for each problem instance. The list of all static features that we extract can be found in Appendix A along with a description. Since presolving can significantly improve the performance of the solving process, we also extract static features after presolving the problem instance with SCIP's default and fast presolve settings, which leads to a total number of 399 static features for each problem instance.

With our benchmark instances, we also extract "dynamic" features which include information extracted from the log files of running SCIP with default settings on a particular instance. We limit such extraction to features that are available up to and including the solving of the root node relaxation. In the larger context of algorithm selection, it is our expectation that this dynamic feature extraction at the root involves only a minimal time commitment relative to the time involved to solve a difficult instance to completion.

2.1.2 Performance Data

For each instance in our benchmark sets, and for each of the 14 different algorithms, we have the following data listed below. To increase robustness, we have averaged the data over five runs for each instance/algorithm. If any run for a particular instance/algorithm takes longer than 600 seconds, we terminate the run early and report the information accumulated up to that point.

- **Timing Data**
  The solving time is the most important performance data for instances on which the solver did not run into the time limit. For the homogeneous `Regions` benchmark set, with few exceptions, SCIP was able to solve the instances within the given time limit. For the more heterogeneous set

M&C, the timing data is much more varied, with a much larger number of timeouts, and in fact, some instances timed out regardless of the algorithm chosen.

- **PD-Integral**
  As a second measure of performance, the integral of the primal-dual gap was used. Herein referred to as the primal-dual integral, this measures jointly the quality of solutions found during the solving process as well as the efficiency with which these solutions are found. Particularly it is the percent gap between the best known upper and lower bounds on the true solution found in the branch and bound tree, integrated over time. This measure was introduced in [16] and does not require any prior knowledge of the ground truth solution for an instance. In our research, we investigated this value and its usefulness as a suitable proxy for runtime when the runtime information was not available (for instance, if a particular run was cut off early by timing out). For further information on how this integral is explicitly calculated, we refer the reader to [16].

- **Completion Codes**
  As a final categorical measure of performance, we have the exit codes provided by SCIP corresponding to various forms of success/failure of the solver on a particular instance/algorithm attempt. We have aggregated these completion codes into the three coarser categories of 'ok', 'timelimit', and 'fail' (details of this aggregation are given in Appendix B). 'Ok' corresponds to a successful run of the algorithm and no data adjustment is necessary. 'Timelimit' corresponds to an algorithm that went beyond its allotted time, in which case we take the execution time to be 600s. 'Fail' corresponds to a situation in which either the solver failed to complete (e.g. due to memory limits), or delivered an answer that was not satisfactory (e.g. claiming optimality of a suboptimal solution). If the solver failed, we take the execution time to be 600s and the PD Integral to being 60,000 (corresponding to 600s of 100% gap).

2.2 Upper Bounds/Well-Defined Features

Within the benchmark set, there can be instances for which numeric data is not well-defined. For instance, if an instance has no left bounds in its constraints, then no average value of the left bounds can be found. In such cases, we use zero.

Additionally, within the timing data, it is possible for an algorithm to complete an instance early but incorrectly (e.g. exceeding the available memory). In such cases, we set the numeric time to completion as 600 seconds (which is our time limit) and the PDI to 60,000 (which is the maximum possible PDI). We note that this presents a challenge for our regression engines because such values are not chosen quantitatively but qualitatively, and it is likely that this choice favors certain classes of machine learning techniques that utilize branching and decision boundaries over continuous techniques that function primarily on regression.

3 Feature Analysis

As stated previously, we currently extract 399 static features and 103 dynamic features. The static and dynamic features can be considered independently from one another or simultaneously. Since the dimensionality of the feature space is relatively large in either case, a few natural questions that arise are:

Q1 Which features are the most important?

Q2 Which features are the least important?

Q3 Is there a lower dimensional representation of the feature space that accurately encapsulates most of the information.[1]

We devote the majority of our attention in answering **Q1** and **Q3** by using Pearson correlation coefficients, principal component analysis, and multidimensional scaling.

3.1 Pearson Correlation Coefficients

Pearson correlation coefficients are normalized values of the covariance matrix, and they take on values between $+1$ and -1. A score of $+1$ indicates that two variables are positively linearly correlated, and a score of -1 indicates they are negatively linearly correlated. We compute this via:

$$\rho(x, y) = \frac{cov(x, y)}{\sigma_x \sigma_y}$$

Here, $cov(x, y)$ denotes the covariance between x and y, which are features in $\mathcal{F}$. Also, σ_x and σ_y denote the standard deviation in x and y, respectively. We plot a heatmap of these correlation coefficients, which allows us to qualitatively answer a few questions:

- Are there any features which can be removed from the feature space? Features which have many correlations, whether they are positive or negative, have their information more or less encoded in other features. These highly correlated features can be removed from the space because of this, thus answering **Q2**.
- What features are important and why? This is most useful when used in conjunction with another approach. For instance, we utilize various techniques that predict which features are the most important. The heatmap can be a useful tool in answering why are they the most important, thus answering **Q1**.
- Are there any issues in the features themselves? For instance, does a particular feature fail to vary within our benchmark set?

As an explicit example, we produce the correlation heatmap using the first 20 dimensions in the feature space (Figure 1). In this case, the feature with zero correlation turns out to be a feature which can be removed from the features space as it renders no useful information. Specifically, this feature was zero across all instances.

[1] This is defined in more detail in a later section

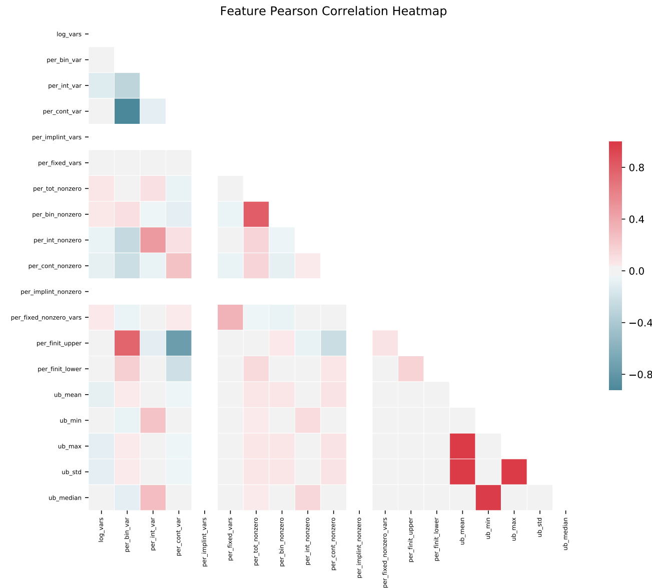


Figure 1: A Pearson correlation coefficient heatmap computed using the first 20 dimensions in the feature space. The diagonal and upper triangular elements have been removed since this heatmap is a symmetric plot.

3.2 Principal Component Analysis

Principal Component Analysis ("PCA") is a method that constructs a new coordinate system to represent data. This new coordinate system is constructed to maximize the variance in data amongst the axes, under the constraint that each axis is orthogonal. The data represented in the first principal component, or first new axis in the coordinate system, has the largest variance. Each additional axis describes less variance of the data. PCA can be used as a dimensional reduction tool owing to this decrease in descriptive power.

We use PCA to map from the original feature space to the new PCA space: $\mathcal{F} \to \hat{\mathcal{F}}$. We vary the dimension of $\hat{\mathcal{F}}$ and calculate the percentage of variance described by this space to study the information dynamic of projecting to lower dimensional spaces. We compute the "information" retained by $\hat{\mathcal{F}}$ as:

$$I(d) = \sum_{i=1}^{d} \hat{\sigma}_i^2 / \sum_{i=1}^{m} \hat{\sigma}_i^2 \tag{2}$$

Where m is the dimension of $\mathcal{F}$, d is the dimension of $\hat{\mathcal{F}}$, and $\hat{\sigma}_i^2$ is the variance of the data represented by the i-th axis in $\hat{\mathcal{F}}$. Since this equation represents a percentage value, it is in the range of $[0, 1]$ and monotonically increases with d. Additionally, $I(0) = 0$ and $I(m) = 1$. See Figure 2 for a plot of the information as a function of d.

PCA additionally allows us to answer the question of "what features are the most important?" The components of $\hat{\mathcal{F}}$ are constructed through linear combinations of features in $\mathcal{F}$. By determining which features are the highest

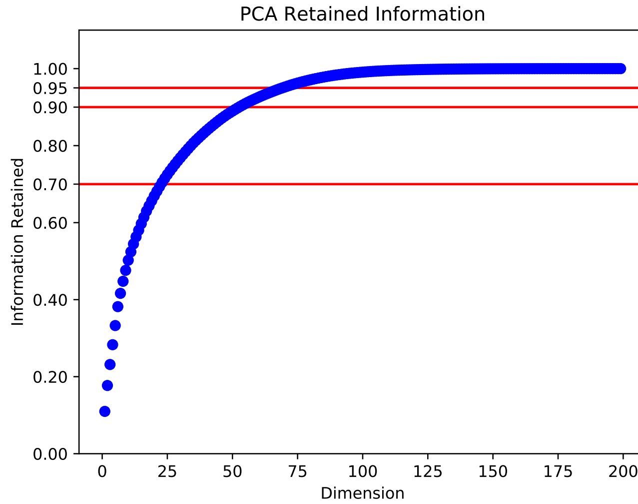


Figure 2: The PCA retained information as a function of dimension given by Equation (2). The input data is the static+dynamic feature space. The various horizontal lines are 70%, 90% and 95% thresholds. The 95% threshold occurs at $d = 71$.

weighted across all linear combinations, we are able to select a list of individual features which PCA relied most heavily on to construct $\hat{\mathcal{F}}$. By keeping 95% of the PCA information[2], we select 71 features.[3] Thus, we answer both **Q1** and **Q3** using PCA.

We note a peculiar clustering that occurs in $d = 2$ when we consider just the dynamic features (refer to Figure 3). When we consider all features, or just the static features, we obtain no similar clustering (Figure 4).

3.3 Multidimensional Scaling

Similar to PCA, multidimensional scaling ("MDS") is a dimensional reduction technique, but it fundamentally differs in that it does so nonlinearly. Rather than attempting to maximize the variances as PCA does, MDS tries to preserve pairwise distances as well as possible in a lower dimensional space. How well the method does can be quantified through an error[4] function which depends on the dimension being mapped to:

$$err(d) = \sum_{i \neq j} |d_{ij} - \hat{d}_{ij}(d)|^2$$

Here, d_{ij} is the pairwise distance between the i-th and j-th points in the original space, $\hat{d}_{ij}$ is the pairwise distance in the lower dimensional space and d is the dimension being mapped to. The goal in MDS is to minimize $err(d)$ by

[2] i.e. setting $I(d) = 0.95$

[3] We point out that although $d = 71$ refers to 71 dimensions in PCA space, and *not* in the original feature space, we nonetheless use this as motivation to select 71 features from the original space.

[4] The loss function used in MDS is typically referred to as the "stress"

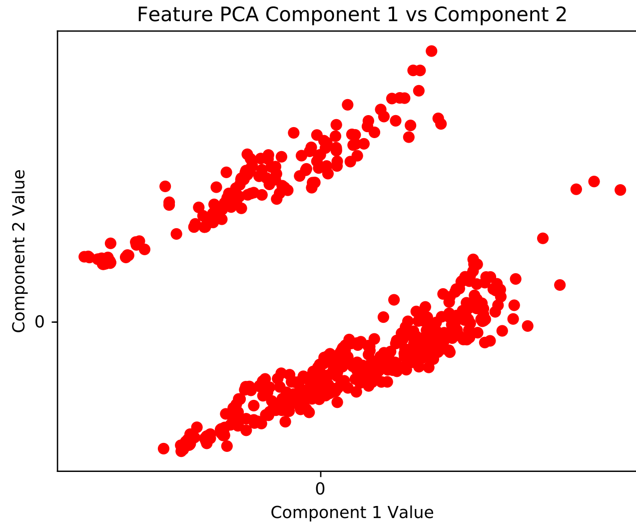


Figure 3: PCA components 1 and 2, for just the dynamic features for `M&C`. Notice the distinct 2 clusters that emerge.

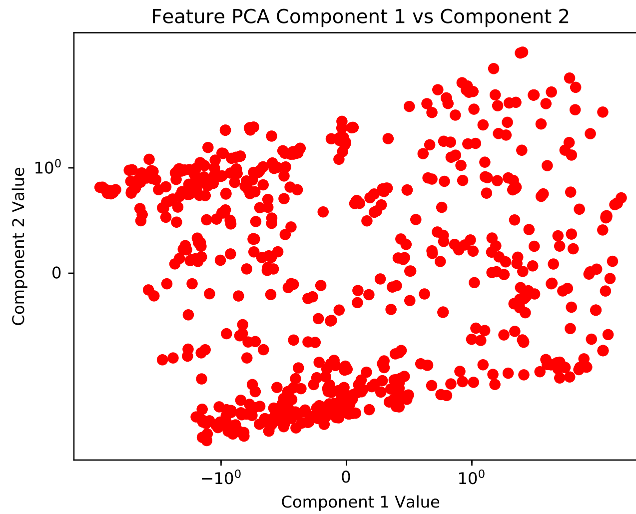


Figure 4: PCA components 1 and 2, for static and dynamic features for `M&C`. Notice, the cluster information is washed away.

choosing an appropriate $\hat{d}_{ij}$. Similar to PCA, we also define a measure of the retained information when mapping to a lower dimension:

$$I(d) = 1 - err(d)/err(2) \tag{3}$$

This form was chosen so that the information is constrained to $[0, 1]$ and so that it monotonically increases with respect to the dimension. Additionally, $I(2) = 0$ and $I(m) = 1$. Setting a 95% threshold on the retained information

results in $d = 19$ (refer to Figure 5). Using this technique we are also able to answer **Q3**.

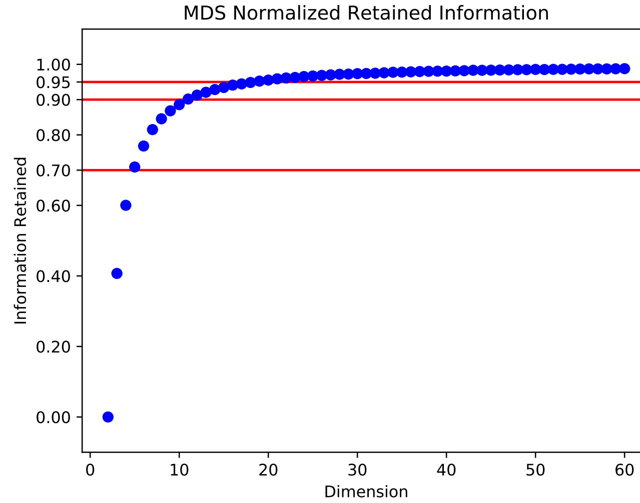


Figure 5: The MDS retained information as a function of dimension given by equation (3). The input data is the static+dynamic feature space. The 95% threshold occurs at $d = 19$.

Similarly, we note a peculiar clustering that occurs for just the dynamic features when the MDS projection is done for $d = 2$. See Figures 6 and 7. We aim to identify the source of these clusters in future work. Currently, we posit that each cluster will correspond roughly to two classes of problem difficulties.

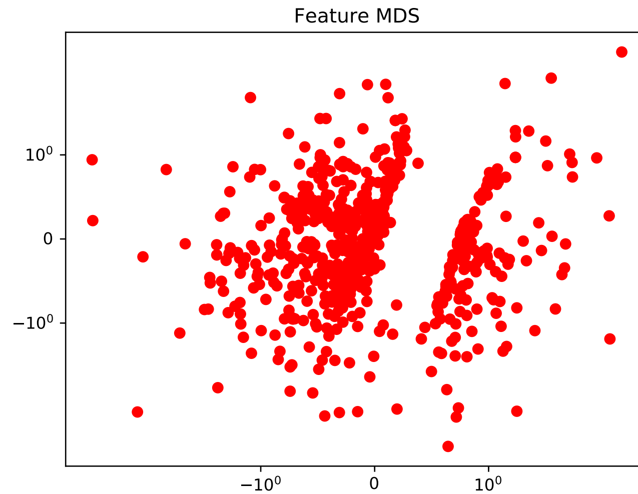


Figure 6: MDS projection to $d = 2$, for just the dynamic features for `M&C`. Notice the distinct 2 clusters that emerge.

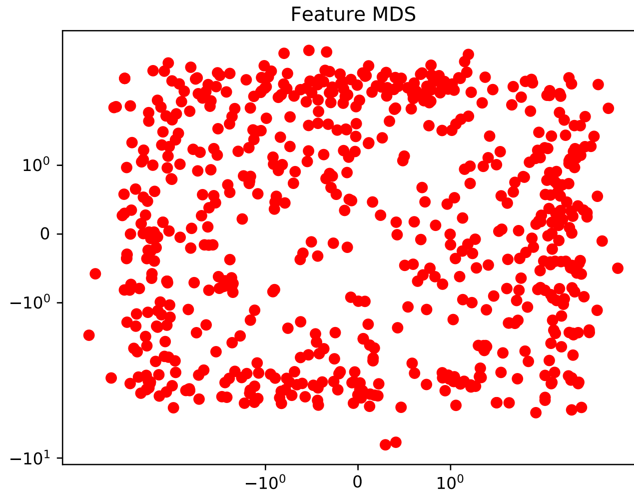


Figure 7: MDS projection to $d = 2$, for static and dynamic features for M&C. Notice, the cluster information is washed away.

3.4 Feature Investigation Conclusions

We summarize a few key observations and results to answer our original questions about the feature space.

We break up our first question (**Q1**) about the feature space into two questions: **Q1a** which features are the most important to a specific model and **Q1b** which features are the most important in general? The answer to **Q1a** simply depends on the model. For instance, the heuristic for selecting the top features for PCA has already been addressed. The heuristic for selecting the top features for a Random Forest Regressor or Classifier depends on which features result in the best tree splits. More specifically, each feature results in a split with a certain amount of information gain (in the case of a decision tree) or error reduction (in the case of a regression tree). The splits with the most[5] information gain or the least error over the entire forest become the top features.

To answer question **Q1b**, we take the intersection of all the model dependent results to construct a more model independent result:

$$\text{Top Features } = \bigcap_i \text{Top Features}_{\text{model}_i}$$

The models we currently use that go into this intersection are: a random forest regressor, random forest classifier that utilizes Hydra (see below), decision tree classifier that utilizes Hydra (see below), and PCA. The full list of model independent top features for the M&C library is available in Appendix A.

To answer **Q3** (i.e. the question of the true dimensionality of the feature space), we look at the number of each model dependent top features list and the information content of PCA and MDS (refer to Figures 2, 5). The true

[5]Here, "most" can be set in various ways, but it more or less means higher than some multiple of the average

dimensionality of the feature space appears to be, on average, around 15% of the dimensionality of the total feature space.

4 Algorithm Selection Methods

Three distinct methods have been used extensively in the algorithm selection community to predict which algorithms might be the most successful: those based on classification, regression, and clustering. We consider machine learning methods that are based on all these approaches. However, we note the success of random forest regressors for the task of algorithm selection [8]. More specifically, we apply random forest regressors, random forest classifiers, support vector machine classifiers, k-nearest neighbours, and logistic regressors. In Section 5, we present the best performing methods only.

For some of the learning methods, we choose to implement additional boosting methods: Adaptive Boosting (AdaBoost) [17] or Hydra [7]. Each of these methods trains on a set of weak learners, the output of which is a weighted sum of these weak learners. Additionally, we tune various parameters independently for each method in order to minimize the testing error. To reach maximal performance, a future investigation may be to tune these parameters without this independence assumption. Again, we only present methods that were the highest performing.

4.1 AdaBoost

AdaBoost [17] learns a weighted combination of weak learners in multiple iterations. In each iteration, it selects a weak learner that minimizes the weighted error on the training data. The weights of misclassified samples in the training data increase in the next iteration. In the end, weights of the selected learners themselves are adjusted to minimize the total error of the ensemble's prediction. We use AdaBoost in conjunction with random forest regressors, and we compare these boosted random forests to the non-boosted versions.

4.2 Hydra

Hydra is a learner voting approach which was originally described by Xu et al. [7]. Their approach uses cost-sensitive decision forests as a decision mechanism and a voting schema that summarizes the results. The experiments here were conducted using a reimplementation of the algorithm as described in the original article. Additionally the algorithm was extended to use other decision-making mechanisms within the general voting part. Particularly decision trees, k-nearest neighbours, support vector classifiers, and logistic regression algorithms were implemented and evaluated.

For the set of algorithms $\{s_1, ..., s_m\}$ Hydra constructs a set of $m(m-1)/2$ pairwise decision-making mechanisms. For a pair of algorithms i, j a mechanism $DM(i, j)$ is trained to decide whether algorithm i or j is ranked higher on a given instance. To perform the algorithm selection for an instance, each $DM(i, j)$ decides which algorithm is better and grants one vote to it, so in the end, $m(m-1)/2$ votes are distributed. Since a larger number of learners is trained for each run of Hydra, the computational effort is higher than for the other

approaches presented here. Also Hydra does not predict actual running times but only a ranking of the algorithms.

4.3 Selecting a Portfolio vs selecting an algorithm

Often, there exist subsets of MIP instances for which a single algorithm performs particularly good or bad. It has been shown in [2, 7, 8] that identifying several algorithms (called a 'portfolio') which are used to solve the instance simultaneously, increases the likelihood to conclude in a short amount of time. Following this trend, we also predict not only one best algorithm but a portfolio of algorithms that covers the entire test bed better than any single algorithm.

4.4 Performance Metric

With our performance data, we want to compare a series of values which can be somewhat disparate or irregular in their distribution. Using the arithmetic mean would over-emphasize outliers, and using the geometric mean would over-emphasize the ratios between small numbers. In [18], Achterberg identified the shifted geometric mean as a suitable compromise between the two approaches, and so it provides a more robust measure of performance for algorithm selection. For a vector $\vec{x} = (x_1, x_2, ..., x_n)$, the shifted geometric mean is given by

$$\sigma_s(x) = \left(\sqrt[n]{\prod_{i=1}^{n} (x_i + s)} \right) - s$$

where the *shift value* s reduces the influence of observations close to 0.

In order to calculate the performance of the algorithm selection mechanism on a test set, we calculate the shifted geometric mean of the best performing algorithm in the portfolio predicted by the mechanism. The mean is taken over all of the test instances. The values of x_i are taken from the real performance data (Primal-Dual integral or solving time as it is described further) and then compared with other algorithm selection mechanisms. In our work we take $s = 10$ for time data and $s = 1000$ for PD-Integral data.

Additionally, since we are also making predictions on portfolios of algorithms, in the case that more than one algorithm is selected, we use the value that corresponds to the best of the portfolio. The best number for each instance is taken to be the value of the portfolio, and then the shifted geometric mean over the instances is calculated. As a basis for comparison, we use a portfolio which is constructed without using the feature data. Using the timing data available, we place algorithms into the portfolio preferentially based upon their shifted geometric means over the instances. We refer to this as *featureless algorithm selection.*

In order to compare the performance of two algorithms to each other, we calculate the shifted geometric mean over the instances separately for each algorithm, and then compute the following performance metric[6]:

$$p_s(f, g) = 1 - \frac{\sigma_s(f(X))}{\sigma_s(g(X))} \tag{4}$$

[6]The performance metric, which we use to measure the quality of our predictions, is not to be confused with performance data!

This formula gives us the relative improvement of portfolio producer f versus portfolio producer g. In our results, we take f to be predictions made from our various machine learning methods (i.e. predictions based on both performance and feature data) and g to be the portfolio produced by featureless algorithm selection.

5 Results

5.1 Performance

We present the highest performing prediction engines we found for the `M&C` benchmark. Recall that this benchmark set is heterogeneous, and while this makes prediction more difficult, we feel that this makes the results more readily generalizable, unlike the `Regions` library which is a homogeneous benchmark and should not be expected to represent the vastly larger MIP instance space. The diagrams we present here are generated using the `M&C` set, as the results from the `Regions` set were quite similar and so for conciseness we present the sole benchmark.

Going into each of these methods is a choice in: the input data, the machine learning method, and whether to boost the learning or not. Random forests were the highest performing methods and inherent to these is a random seed that is used for training. Additionally, a random seed is used for the test/training split (20%/80% in our case). Since there are these sources of randomness, we submit 100 independent runs, where each run is trained on a different seed (i.e. we produce 100 different portfolio predictions for each learning method we utilize). For each run, we compute the performance $p_s(\text{random forest prediction, default})$ of our prediction, cf. Equation (4).

We compare the portfolio predictions produced from the random forest to the portfolio produced by always selecting the default solver. We also produce a portfolio from a featureless approach in which we rank each solver by the shifted geometric mean, and iteratively select the best performing solvers (i.e. with the lowest shifted geometric mean). This featureless approach is also compared to always selecting the default solver. In either case, the quantity p_s can then be thought of as the percent improvement over always choosing the default solver. For example, a value of $p_s = 0.05$ means that the random forest is choosing a portfolio, the solvers of which will run 5% faster than choosing the default solver.

We present our results by taking the arithmetic mean of the 100 different p_s values coming from the independent runs over seeds, and we plot the standard deviation to show the variance of the performance. Using our machine learning methods, we are able to select algorithms that outperform the choice of always selecting the default solver and outperform the featureless approach (refer to Figures 8 and 9). We note that when the size of the portfolio is 14, which is the total number of solvers to choose from, the value of the performance is equal to the value of the virtual best performing portfolio selector (i.e. no selection method will be able to exceed this value).

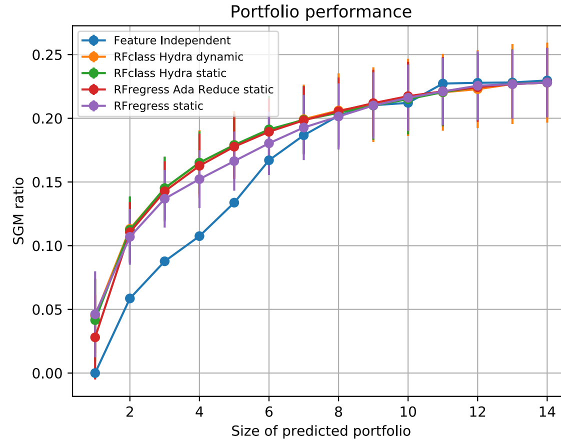


Figure 8: Portfolio performance on the `M&C` data set. Selectors were trained and tested on Primal-Dual integral values.

5.2 Primal-Dual Integrals as a Proxy for Time

One of the questions we sought to answer was whether or not the PDI values will act as a proxy for time information in training, since PDI values provide more information about the progress of the solver than does the timing data, especially in the case that many (or all) algorithms simply reach the time limit.

From Figure 9 we show that in circumstances where time data is not available for training, it is still possible to meaningfully train using primal-dual integral data. The reader will note that this is not automatic – some of our predictor engines performed worse than the featureless approaches, notably for portfolios of size one (attempting to predict only the best algorithm).

Since we are predicting not just a single algorithm but a portfolio of many algorithms, Figure 9 also highlights an important point: competitive methods that work well for choosing a portfolio may not appear competitive when they are used to select only a single algorithm. This itself provides strong evidence that some of the previous research on single algorithm selection could behave very differently if revisited in the context of portfolio selection.

5.3 Software package

We developed a Python software package named "Algorithm Predictor" to support our methodology. All results and plots in this work were produced with this package. Our package supports:

- **Feature Investigation** using various techniques (currently Pearson correlations, MDS and PCA) as well as machine learning methods (currently random regression forest). Each of these software modules produces plots and depending on the method, a list of features that it finds to be the most important.

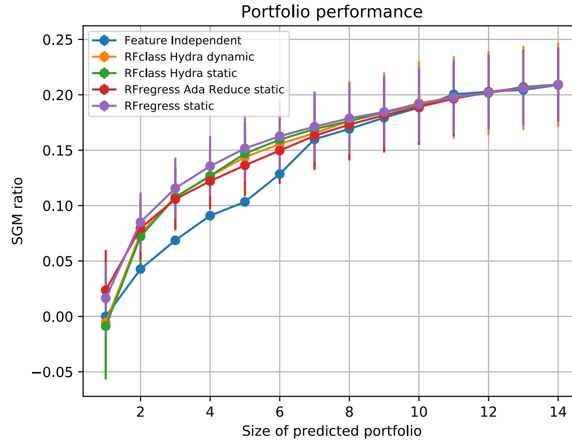


Figure 9: Portfolio performance on the `M&C` data set. Selectors were trained on Primal-Dual integral values and tested on their time performance.

- **Algorithm Selection** such as random regression forests, random forest classifiers and others combined with AdaBoost and Hydra boosting techniques. Currently, the Algorithm Predictor package supports training models based on input features and performance data for single or multiple seeds by splitting training data into 80:20 train to test ratio. Trained models can be serialized to a file, but using trained models for predicting on a single problem instance is not implemented yet.

- **Performance Measurement** for a single model or set of models using shifted geometric mean (SGM) explained in Section 4.4. Additionally, these results can be plotted with variance and/or standard error overlaid.

- **Centralized Configuration System** that controls many parameters of our package.

We also produced documentation where the reader can find more details about our package, as well as tutorials for each point described above. To access the documentation, navigate to the `docs/build` directory and open `index.html` in any web browser. The package has been made publicly available under `https://github.com/GregorCH/algoselection/`.

In addition to this package, the feature computation was implemented using SCIP as a callable library. We also created various Python scripts used for data cleaning and curation that we didn't include in our package because we found them very specific to our problem.

6 Conclusion

Random forests seem to be the clear winner, which is a result consistent with the findings in [8]. We believe random forests somewhat get around this difficulty owing to their ability to locally segment off subspaces, and train on these spaces.

Other methods seem to be more affected globally by various subspaces, which throws off the general predictive power of the machine learning method. Additionally, this is an indication of how homogeneous the `M&C` dataset actually is. We have demonstrated that accurate predictions can be made with our various methods and that our approaches outperform the featureless approach.

In each of our methods, we tune various parameters independently to minimize the testing error. In a future analysis, we would like to remove this assumption and scan the parameter space more effectively to find the global minimum of the testing error.

In the course of our experiments, we also identified a promising avenue for future research. In our experiments we used machine learning techniques and predictive engines to rank algorithms and produced portfolios based upon this ranking, in effect choosing the first best, second best, etc. in that order, demonstrating the usefulness of instance-specific features in this process. However, in the case that one is predicting a nontrivial portfolio (size more than one), it is possible to improve upon the results by focusing on choosing portfolios that focus on including algorithms that complement the existing ones. It is the belief of the authors that one can use the features of instances to intelligently predict such a portfolio, but how to use the features to identify the correct portfolio has not yet been explored. We believe that research in this direction will be valuable in the application, for instance, by identifying the best set of algorithms to run in parallel.

Acknowledgment

The work for this article has been partly conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM) within the program "Graduate-Level Research in Industrial Projects for Students" (GRIPS) 2017. The described research activities have been partly funded by the Federal Ministry for Economic Affairs and Energy within the project BEAM-ME (ID: 03ET4023A-F). The authors would like to thank the Zuse Institute Berlin, the Institute for Pure and Applied Mathematics and the National Science Foundation for their contributions in resources and finances to this project. Special thanks to Daniel Hulme and Karsten Lehmann from Satalia for many fruitful discussions about the topic.

References

[1] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 186–202, 2010.

[2] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. A portfolio approach to algorithm selection. In *IJCAI*, volume 3, pages 1542–1543, 2003.

[3] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC-

Instance-specific algorithm configuration. In *ECAI*, volume 215, pages 751–756, 2010.

[4] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, et al. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

[5] Stephen J. Maher, Tobias Fischer, Tristan Gally, Gerald Gamrath, Ambros Gleixner, Robert Lion Gottwald, Gregor Hendel, Thorsten Koch, Marco E. Lübbecke, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 4.0. Technical Report 17-12, ZIB, Takustr.7, 14195 Berlin, 2017.

[6] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[7] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydramip: Automated algorithm configuration and selection for mixed integer programming.

[8] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[9] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 66–76. ACM, 2000.

[10] Emilie Danna. Performance variability in mixed integer programming, 2008. Presentation at Workshop on Mixed Integer Programming.

[11] Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. *Tutorials in Operations Research*, pages 1–12, 2013. doi:10.1287/educ.2013.0112.

[12] Robert E. Bixby, Sebastiàn Ceria, Cassandra M. McZeal, and Martin W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[13] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006.

[14] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011. doi:10.1007/s12532-011-0025-9.

[15] Computational Optimization Research at Lehigh Laboratory. MIP instances. `https://coral.ise.lehigh.edu/data-sets/mixed-integer-instances/`. Visited 12/2017.

[16] Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013.

[17] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. doi:https://doi.org/10.1006/jcss.1997.1504.

[18] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, 2007.

A Extracted Features

This section gives a detailed description of the features extracted from both the `M&C` and `Regions` datasets. For all instances where the value of some feature is undefined, we use a zero value as replacement. Further in this section, whenever we refer to nonzero or finite values, we do that in the context of the corresponding SCIP functions for checking if a value is zero or infinity. More details can be found in the SCIP documentation[7].

A.1 Static features

Static features are extracted based on the problem instance objective function, the constraint matrix and the right hand side vector. It follows a list of static features based on the objective function of an instance:

- Logarithm of the number of all variables in the objective function.
- Percentage of all binary, integer, implicit integer, continuous, and fixed variables in the objective function, where fixed variables are variables with equal upper and lower bounds.
- Percentage of all nonzero binary, integer, implicit integer, continuous, and fixed variables in the objective function.
- Minimum, maximum, mean, standard deviation and median values (further called summary statistics) for each variable type (binary, integer, implicit integer, continuous, and fixed) and for all types together.
- Summary statistics of finite upper and lower bounds for all variables and amount of finite entries.
- Objective dynamism calculated as the logarithm of the ratio of the maximal and minimal element in the vector of all absolute nonzero objective coefficients.

Before feature extraction, every constraint (all coefficients from the constraint matrix, left and right hand sides) was normalized such that the maximal absolute coefficent value of is one. It follows a list of the features extracted from the constraint matrix, left and right hand sides of the problem instance:

[7] `http://scip.zib.de/doc/html/`

- Logarithm of number of all constraints.
- Percentage of each linear constraint type in problem instance. All 16 constraint types used for classification are listed MIPLIB2010 webpage[8].
- Summary statistic of finite right and left hand sides vectors (separately and together), including vectors density.
- Percentage of finite right and left hand sides.
- Summary statistics of number of nonzeros of each constraint.
- Summary statistics for vectors of mean, minimum, maximum and standard deviation values of linear constraint coefficients and also summary statistics for vector of ratio of maximum and minimum value for each linear constraint in problem instance.
- Constraint matrix density.
- Summary statistics for clique vector.
- Dynamism statistics which is calculated on vector of elements calculated for every linear constraint. Each value is calculated as logarithm of ratio of maximum and minimum nonzero coefficients for particular linear constraint.

A.2 Dynamic features

In order to extract dynamic features, instances were given to the SCIP solver using the default settings, and the solving process was allowed to continue up until the completion of the solution at the root node. No data was extracted from the time after the branching process began. It is our belief that this partial solving process can be done efficiently for most instances since in practice most of the computational effort on hard instances is spent in branching.

More precisely, dynamic features used in this paper are extracted from SCIP execution logs based on the SCIP statistics output which gives performance information, for instance, resolving, separating and LP statistics at the root node. More details about SCIP statistics output can be found in [5]. In the case that one of the features extracted in this manner has a value of zero for every instance in the dataset, we drop that feature for the purposes of our experiments. Also, we do not use information that is provided in units of time, since such information is highly machine dependent; however, it is conceivable that one could use this information in future study if one also took the standpoint that it is acceptable to demand that an end-user benchmark and train the predictor on the machine that will be later making the predictions. In our experiments, we have the expectation that the training and prediction will be completed on different machines, and so we do not incorporate dynamic time information into our feature set.

[8] `http://miplib.zib.de/miplib2010.php`

A.3 Top features for `M&C`

The list of the important features obtained using the procedure described in Section 3.4 for the `M&C` data set:
Separators Cuts clique
log constr with off presolve
nvar all max with off presolve
clique max with fast presolve
log vars with off presolve
Separators Calls strongcg
clique max with default presolve
rh constr ratio with default presolve
RootNode FinalRootIters

A.4 Top features for `Regions`

The list of the important features obtained using the procedure described in Section 3.4 for the `Regions` data set:
constr matrix density with default presolve
coeff bin mean with default presolve
coeff bin std with fast presolve
Separators Calls clique
Separators Calls impliedbounds
Presolvers Calls trivial
Presolvers Calls dualfix
coeff bin median with off presolve
coeff all median with off presolve

B Completion Codes

- Ok
  - `'ok'`: a typical satisfactory completion code.
  - `'fail_solution_infeasible'`: the algorithm terminated and gave a solution, but that solution does not satisfy the constraints. This is usually due to rounding errors and other forms of numerical inaccuracy within the machine so can be considered 'ok'.
  - `'solved_not_verified'`: the algorithm terminated and found a solution that was better than the reported bound in the instance definition (happens either because the instance had not previously been solved or because the bound was simply missing in the instance definition). This should mean that the algorithm has correctly solved the instance, but because we do not have the ground truth we cannot confirm its correctness.
- Fail
  - `'fail_abort'`: indicates a catastrophic failure of the algorithm resulting in its early termination

  - ‘`fail_dual_bound`’: this only happened on one of our training instances. This happens when the algorithm times out but reports a lower bound on the dual which is inconsistent with the known lower bound.
  - ‘`fail_objective_value`’: the algorithm terminated and gave a feasible solution, but that solution was not optimal.

- Timelimit
  - ‘`timelimit`’: typical completion code when algorithm exceeds the time limit.
  - ‘`better`’: will happen when the algorithm times out, but it did find at least one feasible solution that was better than the reported bound in the instance definition (happens either because the instance had not previously been solved or because the bound was simply missing in the instance definition)