Random Sampling and Machine Learning to Understand Good Decompositions

S. Basso and A. Ceselli [*]

Università degli Studi di Milano, Dipartimento di Informatica
{*svr.basso@gmail.com, alberto.ceselli@unimi.it*}

A. Tettamanzi
Université Côte d'Azur, Inria, CNRS, I3S
andrea.tettamanzi@unice.fr

March 25, 2017

Abstract

Motivated by its implications in the development of general purpose solvers for decomposable Mixed Integer Programs (MIP), we address a fundamental research question, that is to assess if good decomposition patterns can be consistently found by looking only at static properties of MIP input instances, or not. We adopt a data driven approach, devising a random sampling algorithm, considering a set of generic MIP base instances, and generating a large, balanced and well diversified set of decomposition patterns, that we analyze with machine learning tools. The use of both supervised and unsupervised techniques highlights interesting structures of random decompositions, as well as suggesting (under certain conditions) a positive answer to the initial question, triggering at the same time perspectives for future research.
Keywords: Dantzig-Wolfe Decomposition, Machine Learning, Random Sampling

1 Introduction

General purpose Mixed Integer Programming (MIP) solvers have been developed for decades from both theoretical, algorithmic and software engineering points of view. There is indeed a huge interest in making such tools more

[*]The work has been partially funded by Regione Lombardia – Fondazione Cariplo, grant n. 2015-0717, project `REDNEAT`, and partially undertook when the second author was visiting INRIA Sophia Antipolis - I3S CNRS Université Côte d'Azur.

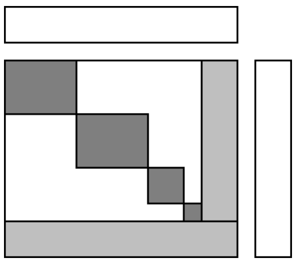


Figure 1: Structure of a MIP with diagonal blocks (black) and border (grey).

and more effective, the primary one being their ease of integration in real world decision support systems [4, 5, 6].

All state-of-the-art solvers currently rely on complex methods mounted on branch-and-cut frameworks, where Linear Programming (LP) relaxations are used to produce initial dual bounds, that are subsequently strengthened by cutting planes [2]. On the one hand such a paradigm offers efficiency and stability, coming from a deep computational understanding built over the years; on the other hand, such a structure yields to drawbacks that show to be very hard to overcome by simply plugging in additional techniques.

A key one is related to the structure of models: while on the majority of the cases generic cutting procedures are very effective, on a few classes of combinatorial optimization problems the initial LP dual bounds are too poor to drive the full method. A second weakness is related to the structure of algorithms, preventing them to scale effectively as the amount of available computing resource increases, for instance thanks to multi-core or distributed computing capabilities.

To overcome these drawbacks, at least when well-structured combinatorial optimization problems need to be optimized, researchers have started to investigate alternative paradigms. One of them is Dantzig-Wolfe decomposition applied to integer programs (DWD): on applications like vehicle routing, crew scheduling, cutting stock, facility location, generalized assignment, and many others [13], DWD-based techniques proved to be far superior to branch-and-cut. In fact, models for these applications exist, having as common structure a constraint matrix in a so-called bordered block-diagonal form (see Figure 1). DWD is able to disaggregate these problems in a master entity, involving only constraints and variables in the border, and one or more subproblems, one for each diagonal block. DWD can then be paired with column generation techniques [1] to obtain dual bounds that are potentially much tighter than the LP relaxation of the original model in reasonable computing time. Furthermore, these column generation techniques prove to scale very well as the amount of computing resources increases [22].

DWD has however, one fundamental issue preventing it to be applied as generically as cutting planes: decomposition patterns must be provided by the modeler, and such a task requires deep mathematical programming skills. That is why, although many positive results have been obtained by problem-tailored DWD approaches, very few attempts have been made in the literature to tackle MIPs with *generic* DWD methods.

All of them aim at analyzing a MIP, understanding whether a suitable structure can be detected, and automatically performing DWD and column generation. A few research threads can be surveyed in the literature. The first one considers to explicitly enrich the input, asking the user both a MIP instance (as in current branch-and-cut based general purpose solvers), and a decomposition pattern describing the block structure to be exploited. DWD is then applied automatically. Approaches along this line include [14], [16] and [15]. In [9] a full generic column generation framework is introduced, which is currently engineered and released with the SCIP framework [10]. Since finding suitable decomposition patterns requires very specific technical expertise, a second research thread is to ask only a MIP instance as input, and to detect if in that instance specific *previously known* structures occur, which are known to be suitable for DWD. Successful contributions along this axis are indeed embedded in [9]. A third, more ambitious, thread is to assume that *previously unknown* decomposable structure may be found (or even *enforced*) in a generic MIP instance provided as input, that can be made evident through a suitable algorithmic search for good decomposition patterns. Very recently, the authors of [12] propose to preprocess the input MIP with machine learning tools, to detect first of all if it is amenable for a successful decomposition approach or not. In [11] and [3] fully automatic frameworks are proposed, obtaining better results than CPLEX [4] on a noticeable set of MIP instances. A common issue of these approaches is to find suitable *optimization metrics*, exploiting MIP instance details to score decomposition candidates.

Indeed a fundamental research question remains open, that is to assess if good decomposition patterns can consistently be found at all by looking only at static properties of the instance, or not, and possibly to understand which properties are worth considering.

In this paper we try to step towards the clarification of this issue with an agnostic approach. That is, we consider generic MIP instances and we generate a large, balanced and well diversified set of decomposition patterns. Then we analyze them with statistical and machine learning tools, with the main aim of understanding if good ones can be distinguished from others by automatic methods looking only at the static properties of the corresponding matrices.

We formally employ both supervised and unsupervised learning tools [19] although, whenever possible, we present our methods with an operations research language and perspective. Indeed using Machine Learning for Integer Programming [24] or vice versa [23], and in general integrating both fields [25], is a recent and extremely promising trend.

The paper is organized as follows. First we face the key issue of producing a significant dataset, proposing a randomized sampling algorithm (Section 2); we then perform a preliminary assessment on the properties of such a dataset (Section 3). Second, we check if simple correlation patterns arise, linking single properties of the MIP instance to the computational behavior of decompositions (Section 4). Since no particular pattern is found by this check, we perform more advanced analyses for finding out if *combinations of MIP instance properties* can be used to effectively predict how good a decomposition is (Section 5). We finally estimate which is the impact of exploiting previous knowledge of good and bad decompositions, both on the same and on different MIP instances, in correctly understanding new decompositions (Section 6). Our research leads to some perspectives, which are collected in a final discussion (Section 7).

2 Dataset generation

Our first target was the creation of a significant and well diversified dataset, as a key requirement for our investigation was to have both *good and bad* decomposition to compare. This task turned out to be difficult: on one hand, drawing a random decomposition for an arbitrary MIP is very likely to yield unpromising results; on the other hand, focusing on specific good decomposition patterns for known decomposable MIPs would not allow to produce both diversified positive and negative examples. Indeed, many of our initial attempts to produce diverse decompositions failed in either sense, as it was very hard to produce a sufficiently large and spread sampling, and make at the same time block diagonal structures appear in the coefficients matrix; in turn, no block structure basically means a predictably useless decomposition approach.

We finally found the following approach to produce meaningful results. We considered a set of *generic base MIP instances* (base instances in the remainder), selecting them from *unstructured* problems in MIPLIB 2003 [7] and MIPLIB 2010 [8] that, according to [3], might potentially be suitable for a decomposition approach. The list of our base instances is reported in Section A.1 of the appendix. These are all stated as minimization problems. For each base instance we generated 1000 random decompositions with the randomized sampling algorithm reported in Figure 2. It builds blocks of constraints, that in turn induce blocks of corresponding *covered* variables, that is variables having nonzero coefficients in the constraints of the block.

The randomized algorithm produces decompositions with a random number of blocks; we fixed a minimum blocks threshold: when a decomposition was produced, containing fewer blocks than the threshold we simply repeated the generation process, until that criterion was met. Such a threshold was fixed to 3 in our experiments.

```
• Fix the initial set of blocks to be empty.

• Consider a special *border* block, which is initially empty.

• At each iteration, assume that a set of blocks has already been created; a new
  constraint is chosen at random, with a probability directly proportional to
  its *sparsity*, defined as the number of zeros in the constraint matrix

    – if the constraint covers no variable that is covered by other
      constraints belonging to existing blocks, then a new block is created,
      and the constraint is assigned to that block
    – if the constraint covers any variable that is covered by other
      constraints belonging to more than one existing block, then the
      constraint is put in the border
    – if the constraint covers variables that are covered by constraints in a
      single existing block, and possibly additional variables that are still
      uncovered, then the constraint is added to that block.
```

Figure 2: Randomized algorithm for sampling decompositions.

Then, each decomposition was analyzed, and a set of *features* was extracted by measuring standard properties of both the constraint matrix, the constraints right hand sides and the objective function coefficients of variables in each block. A full list of the features is reported in Section A.2 of the appendix. These include also the quality measures that were employed in the decomposition optimization algorithms of [3]. We have also introduced a coefficient to estimate how similar is each block to a totally unimodular matrix: intuitively total unimodular blocks mean decomposition subproblems possessing the integrality property, and therefore yielding no bound improvement in a DWD-like decomposition process. Sets of conditions being sufficient for proving total unimodularity require, for instance, to find a suitable bi-partition of the set of rows [26]: in an effort to obtain a numerical feature, we approximated such a check by means of a randomized greedy algorithm, whose pseudo-code is reported in Figure 3.

The randomized greedy algorithm was iterated ten times, and the highest value was retained as final coefficient. The coefficient represents the fraction of variables that have been considered before stopping in a failure status. That is, in the extreme cases, 1.0 means that the corresponding matrix has been proved to be totally unimodular (that is undesirable for the decomposition process), while 0.0 means it has been proved not to be totally unimodular.

Afterwards, we ran a simulation on each decomposition, experimentally solving the continuous relaxation of the corresponding extended formulation, that is computing the corresponding dual bound at the root node of a branching tree. Such a simulation was carried our by means of the generic column generation procedure of GCG [9] from the SCIP framework [10],

```
For each block independently

  • if any variable of the block contains more than two non-zero entries, return
    $0.0$

  • if any entry in the block is different from $0$, $1$ or $-1$, return $0.0$

  • initialize a left and right partition to be empty sets

  • initialize a counter $k = 0$

  • iteratively, consider each variable of the block in the order in which they
    appear in the base instance:

      – if the number of its non-zeros entries in the block is either zero or
        one, increase $k$ and iterate
      – if the variable has two different entries in the block, corresponding to
        constraints assigned to different partitions, increase $k$ and iterate
      – if the variable has two equal entries in the block, corresponding to
        constraints assigned to the same partition, increase $k$ and iterate
      – if the variable has two different (resp. equal) entries in the block,
        one of which in a constraint assigned to a certain partition, and the
        other in a constraint assigned to no partition yet, assign the second
        constraint to the opposite (resp. the same) partition, increase $k$ and
        iterate
      – if the variable has two different (resp. equal) entries in the block,
        none of which in a constraint assigned to a partition, choose a random
        partition, assign the first constraint to it, and the second constraint
        to the other (resp. the same) partition, increase $k$ and iterate
      – otherwise simply iterate without increasing $k$

  • return $k$ divided by the total number of variables appearing in the block.
```

Figure 3: Computation of a Total Unimodularity coefficient.

excluding all preprocessing, cut generation and problem reduction procedures, and activating IBM ILOG CPLEX 12.6.3 [4] to solve LPs. Bash and Python scripts were used to manage the whole process. The result of each simulation was a dual bound value and a corresponding computing time for each decomposition. No time limit was given to the simulations, but a very limited number of runs did not terminate after weeks of computing. Those were marked as “timeout”; obtaining a valid dual bound was always possible, even on timeout instances.

After preliminary experiments we decided to remap time values in a base 10 logarithm scale. Both time and bound were then transformed into scores in the range $[0, 1]$ through a feature scaling normalization, using as limits the maximum and minimum values on the decompositions of the same base instance. That is, we assigned higher scores to high bounds and low computing times: decompositions that reached the time limit received score 0 in computing time whereas those that had maximum (resp. minimum) values in bound (resp. time) received score 1. After all decompositions related to a certain base instance were processed, each of them was marked as positive or negative according to the following relaxed Pareto optimality definition.

Let A and B be two decompositions for the same base instance, let b_A and b_B be the dual bound scores they yield, and let t_A and t_B be the computing time scores required to achieve them. We define decomposition A to be *dominated* by decomposition B if and only if $b_A \cdot (1+\beta) \leq b_B$ and $t_A \cdot (1+\tau) \leq t_B$, where β and τ are two tolerance parameters that we use to control how many decompositions can be considered comparably good.

In particular, for the time tolerance τ we evaluated 10 logarithmic spaced values from 1% to 100%, while for the bound we evaluated 10 linearly spaced values from 0.1% to 10%. After fixing β and τ, each decomposition that was dominated by no other was marked as *positive*, all the remaining ones as *negative*. These boolean marks were added to the dataset as class labels. Bounds and computing time were recorded, but excluded from the set of features, as they are not statically observable, and therefore unsuitable as input in a hypothetical decomposition generation process.

In Figure 4 (resp. 5) we report the average number of positive decompositions for every value of τ (resp. β) over every possible value of β (resp. τ). Variations on τ have a sharp effect on the number of Pareto optimal decompositions whereas variations on β have a more limited impact.

We performed experiments on several tolerance configurations, finding the following two to be more representative of the general behavior of our methods:

- Tolerance L (Low): with 13% tolerance τ on time and 0.77% tolerance β on bound,
- Tolerance H (High): with 21% tolerance τ on time and 10.00% tolerance β on bound.

For the analysis performed in the following sections we therefore report results considering both settings.

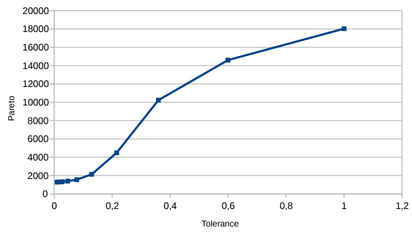


Figure 4: Average number of positive decompositions (time)

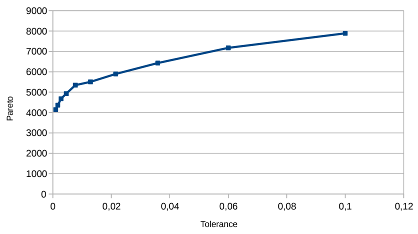


Figure 5: Average number of positive decompositions (bound)

3 A preliminary analysis of the dataset

As reported, the initial dataset was composed by 39 base instances, and 1000 decompositions for each of them. Three base instances were then excluded, as their decompositions went systematically timeout. Duplicate records (about 4%) were discarded as well. The relevant features of the dataset were normalized with a simple criterion: measures on the number of variables of each decomposition were divided by the number of variables of the corresponding base instance; the same was done for the constraints. The dataset obtained after these operations consisted of 34565 records and 117 features. About 2% and 14% of decompositions were classified as positive respectively with tolerance L and tolerance H.

As far as the dimensionality of the dataset is concerned, we found through a Principal Component Analysis (PCA) 22 components to be significant according to the eigenvalues criterion [20]. However, all these components had a correlation near zero to both time and bound scores. Indeed, we performed many of the subsequent experiments also by considering this set of 22 components instead of the full set of original features. No quality increase was obtained, and at the same time we encountered no particular computational bottleneck. Therefore we present only results on the dataset using the full set of features.

As a double check for the initial sampling process we compared the trade-off between time and bound scores for decompositions of each base instance. In Figure 6 we report a scatter plot of time (y axis) and bound (x axis) scores on four representative base instances: each point represents the behavior of a single decomposition. The sampling process produced different categories of results. In base instances like `beasleyC3` the randomized sampling algorithm was very successful in uniformly sampling our search space, finding also a few decompositions proving to be simultaneously good in terms of bound and time. In others, like `timtab1` the sampling was still adequate, yielding as expected to a larger set of positive decompositions covering diverse trade-offs between bound and time. A few instances like `p2756` showed a biased outcome, including both good bound and good time decompositions, but no decomposition providing reasonable trade-offs. Finally, in instances like `enlight13`, an interesting stack-like structure appeared, suggesting the search for algorithms trying in these special cases to locally improve decompositions after an initial random selection.

4 Correlation analysis

We initially focused on a fundamental question, that is understanding if single features could be found that were predictive on the performance of decompositions. To this aim, we measured the linear correlation between

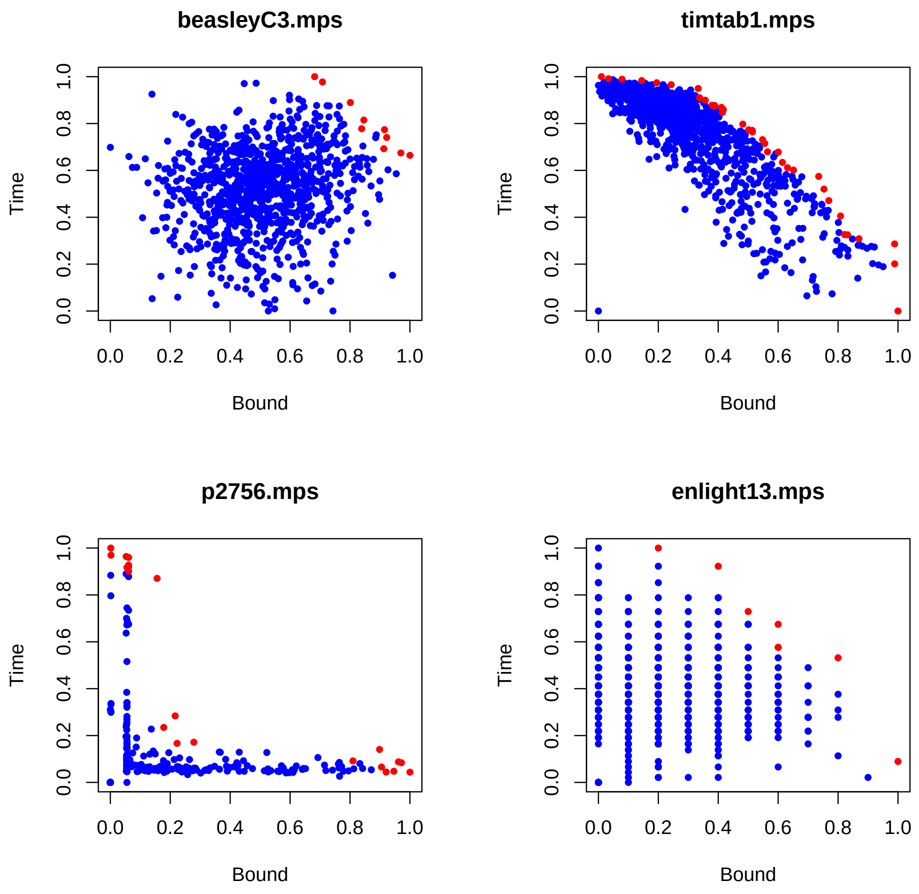


Figure 6: Bound and time scores for each decomposition on different base instances. Red (resp. blue) points indicate positive (resp. negative) decompositions.

each feature and both time and bound scores. In Figure 7 (resp. 8) we report the results of this analysis for the time (resp. bound) score, depicting for each feature (x axis) its linear correlation coefficient value (y axis) over all the decompositions of the dataset. Both analyses highlight that there is no single feature with strong correlation, not even the number of blocks as one may expect. The only features that have a non-negligible correlation (i.e., over 0.6) are the average and maximum number of binary variables in blocks with respect to the bound score; such a result is trivially related to the effect of the Dantzig-Wolfe decomposition process.

We repeated the same analysis splitting the dataset base instance by base instance, to understand if different features could be predictive of scores on different base instances. We report our results in Figure 9 (resp. Figure 10) for time score (resp. bound score) with a box plot for each feature, collecting the linear correlation distribution summary (y axis) between each

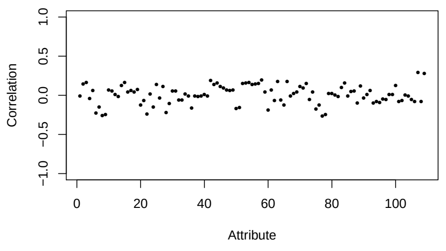


Figure 7: Corr. between feat. and time

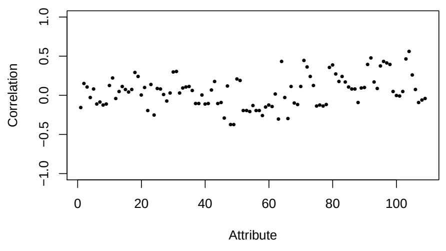


Figure 8: Corr. between feat. and bound

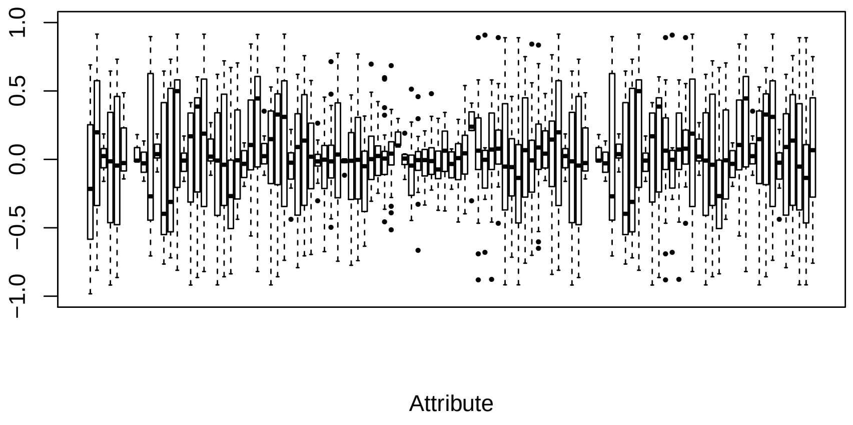


Figure 9: Correlation between features and time, distribution over base instances.

feature (x axis) over different base instances. Again, the results confirm that no single feature shows high correlation with scores on every base instance. At the same time, a set of instances show to have strong correlation with a restricted set of features. This is depicted in figures 11 for time score and 12 for bound score, in which we plot the third quartile (y axis) of the distribution of correlations between features and scores on each base instance (x axis).

The correlation analysis suggests that relationships between the features of the dataset and time and bound scores exist. This confirms that the set of features in our dataset is meaningful, but also that an algorithmic approach trying to optimize static features of decompositions might be pertinent. However, no simple link between features and scores emerges, and therefore the task of defining suitable quality measures, to be used in optimization algorithms for *creating* good decompositions, is highly non trivial.

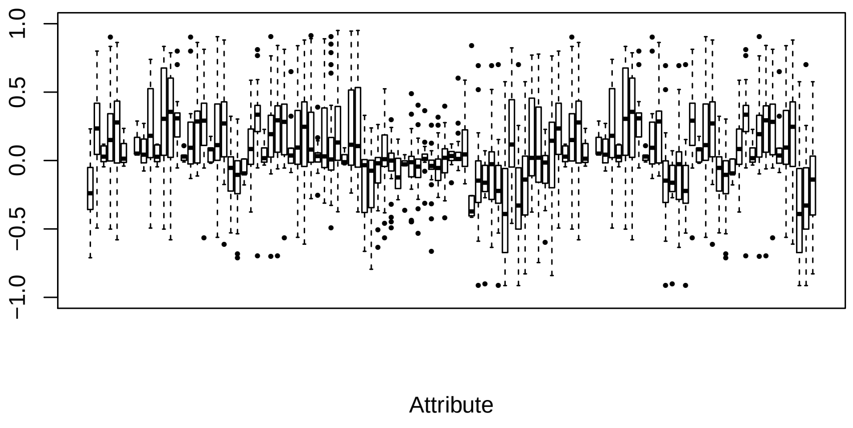


Figure 10: Correlation between features and bound, distribution over base instances.

5 Splitting positive and negative decompositions

We therefore tackled a more involved question, that is if *combinations* of features exist, that might be predictive on decompositions performance. We devised the following approach. First, we defined a *distance measure* between decompositions *in the space of time and bound scores*, and we tried to check if finding a suitable model was possible, mapping whole vectors of features to score distances. We employed regression techniques for such a check. Then, we tackled the problem of automatically detecting if a decomposition is positive (that is, Pareto optimal according to the definition of Section 2) through classification models.

5.1 Regression analysis

As sketched above, we first defined the following distance measure between decompositions in the space of time and bound scores:

$$\Delta(A, B) = \sqrt{(t_A - t_B)^2 + \alpha \cdot (b_A - b_B)^2}$$

where α is a suitable weighting parameter. Then, following the intuition that good decompositions are those having low score distance to positive ones, we computed for each decomposition A, the following Pareto-distance value:

$$\delta(A) = \min_{B \in \mathcal{P}(A)} \{\Delta(A, B)\}$$

where $\mathcal{P}(A)$ is the set of positive decompositions on the base instance of A.

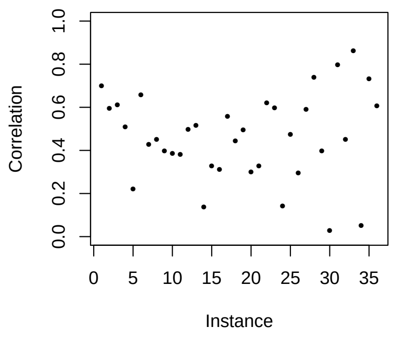


Figure 11: Third quartile of the distribution of correlation between features and time.

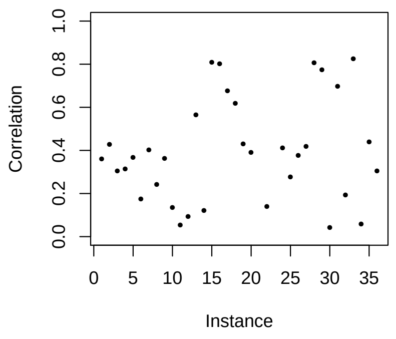


Figure 12: Third quartile of the distribution of correlation between features and bound.

For each decomposition we calculated two Pareto-distance scores, both using Tolerance L for defining $\mathcal{P}(A)$: the first one with $\alpha = 1$, that is equally weighting time and bound scores, and a second one with $\alpha = 10$ in which bound differences have higher importance. In figure 13 we report the cumulative distribution function (CDF) of the $\delta(\cdot)$ values for the case $\alpha = 1$, showing decomposition distances to be spread fairly evenly in the range $[0.0, \sqrt{2}]$.

As a double check, for both $\alpha = 1$ and $\alpha = 10$ cases, we performed a correlation analysis between $\delta(\cdot)$ and each feature in the dataset. The results are reported in Figure 14 for $\alpha = 1$ and in Figure 15 for $\alpha = 10$: linear correlation values (y axis) are reported for each attribute (x axis). In the $\alpha = 10$ case a set of features have correlation values over 0.7. These are those features related to the constraint right hand side values of both the blocks and the border: right hand side values are negatively correlated to distance, while right hand side dispersions are positively correlated to

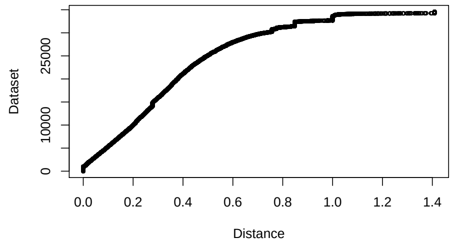


Figure 13: CDF of distance, $\alpha = 1$

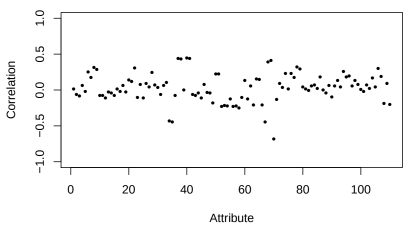


Figure 14: Correlation between every feature and $\delta(\cdot)$ for $\alpha = 1$

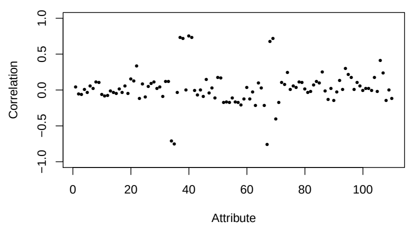


Figure 15: Correlation between every feature and $\delta(\cdot)$ for $\alpha = 10$

distance; the second phenomenon actually matches the common practice of mathematical programming experts, that whenever appropriate tend to keep homogeneous master and subproblems, while we conjecture the first one to depend on full base instances rather than on single decompositions.

Then, we tried to model $\delta(\cdot)$ as a function of the decomposition features by means of regression techniques. We trained a Support Vector Regression (SVR) model [18] to predict the values of Pareto distance $\delta(\cdot)$ given the vector of decomposition features. We split the dataset, using 75% of data for training and the remaining for testing. In particular, we chose representative training samples by categorizing the full dataset in 10 layers, based on the distance values, and by performing a stratified sampling, randomly picking 75% of data from each layer.

We experimented by training four different models using either $\alpha = 1$ or $\alpha = 10$, and either using the full set of features or only those 8 being strongly correlated with the $\delta(\cdot)$ calculated with $\alpha = 10$. The R statistical analysis framework [27] 3.2.5 was used for this experiment, and in particular the SVR implementation of the e1071 library.

Only the model with $\alpha = 1$ using the full set of features showed predictive potential. For that setting we report in Figure 16 for the training set and in Figure 17 for the test set, a scatter plot detailing the predicted values (y axis) respect to the calculated ones (x axis); that is, points on the bisector of the quadrant are perfect predictions.

We also report the CDF of absolute prediction errors in figures 18 and 19 for the training and testing phases, resp.. Similarly, we report the CDF of mean absolute percentage error (MAPE) for both training (20) and testing (21). MAPE was calculated by excluding decompositions whose $\delta(\cdot)$ values were in the first tenth percentile, since they were too small to produce significant results. About 80% of the absolute errors in both training and testing are below 0.2 confirming that most predictions fall in an area relatively close to the correct values. MAPE figures show similar results and confirm that high errors occur more often on low distances.

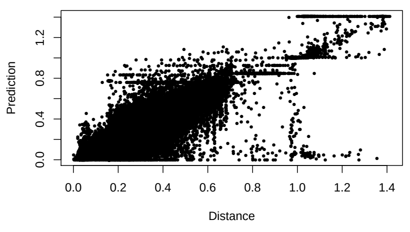


Figure 16: Training, predicted values over expected ones. $\alpha = 1$

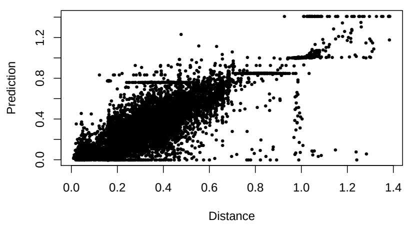


Figure 17: Testing, predicted values over expected ones. $\alpha = 1$

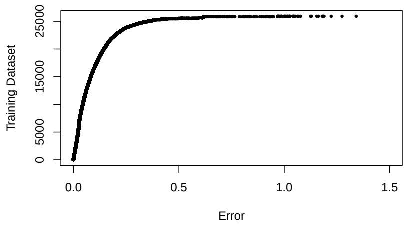


Figure 18: Training, CDF of absolute error. $\alpha = 1$

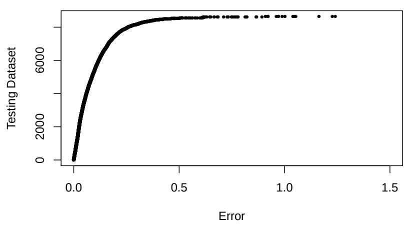


Figure 19: Testing, CDF of absolute error. $\alpha = 1$

5.2 Classification analysis

Strictly speaking, to accurately predict the score distance $\delta()$ is unnecessary in our application. In fact, all we need would be a model helping to discard any decomposition which is not Pareto optimal. Therefore, having verified that combinations of features exist, that tend to be predictive on bound and time scores, we tried a classification experiment considering each decomposition to possess a target feature taking value *true* for those decompositions being marked as positive with respect to their base instance, *false* otherwise, and a set of numerical predictors given by the decomposition features. The Support Vector Machine (SVM) [17] implementation of library e1071 of the R framework was used in this experiment.

We first report that no experiment was successful using the original dataset as it was, since the set of positive instances was always too small to impact on training. Therefore, performed a splitting of the dataset in training and test data; training data was then re-sampled by allowing repetitions, and balanced in different ways. Test data was left untouched. A 1:1 balancing between positives and negatives in the training set allowed a SVM classifier to reach good overall accuracy, but lacked precision. A finer look at the results revealed that sampling positive decompositions multiple times from the training dataset did not guarantee enough support for most base

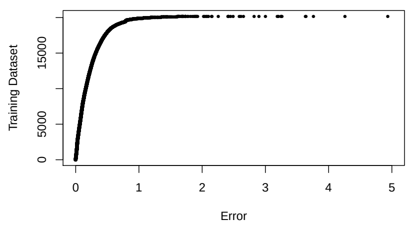


Figure 20: Training, CDF of MAPE. $\alpha = 1$

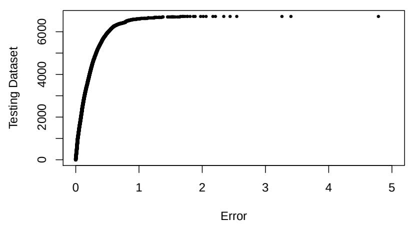


Figure 21: Testing, CDF of MAPE. $\alpha = 1$

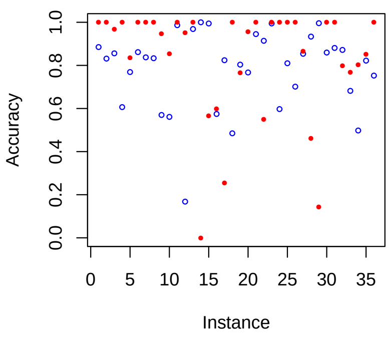


Figure 22: Training accuracy

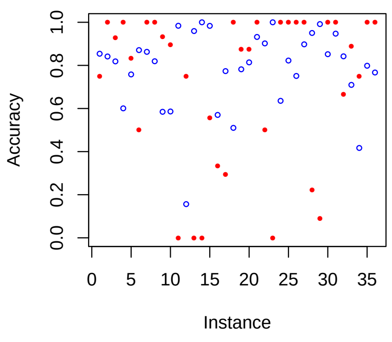


Figure 23: Testing accuracy

instances. For this reason, we further refined the creation of a representative training dataset by performing a stratified sampling of 75% of positive and negative decompositions *for every instance independently.* Then a 1:1 balancing was obtained for each of them through re-sampling; the samples were in the end merged in a single set.

Our results on the test set are detailed in Table 1 for both tolerance L and H. The table contains one row for each base instance, and is split in two blocks, one for each tolerance setting. For each setting and each base instance we report the number of True Negative (TN), False Positive (FP), False Negative (FN) and True Positive (TP) elements in the test set. We also summarize the results in Figure 23 for the tolerance L case; for reference, we also report the performance on the training set (Figure 22). For each base instance (x axis) we report the true positive rate (TPR), defined as TP/(TP+FN) (red dots) and true negative rate (TNR), defined as TN / (TN + FP) (blue circles). Overall, we obtained 77% accuracy in testing when using tolerance L to define positive decompositions. The corresponding accuracy in training was 80%, suggesting mismatches to be more due to

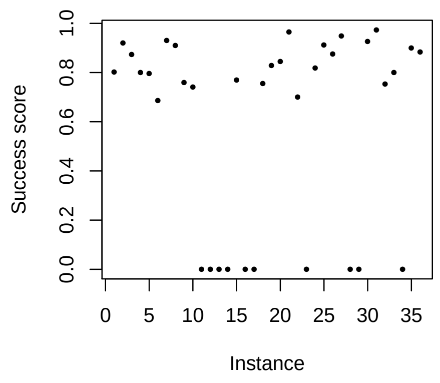


Figure 24: Testing success score

the structure of data than due to a lack of generalization power or other modeling problems. However, a closer look reveals a very heterogeneous behavior among the base instances: while on a few we obtained unacceptable accuracy, on the majority of them accurate classification of both positives and negatives was produced.

We therefore defined the following *success score* measure, to clearly split base instances that we are able to classify with good accuracy from the others:

$$success(x) = \begin{cases} 0, & \text{if } \min(TPR, TNR) < 0.5; \\ \frac{TPR+TNR}{2}, & \text{otherwise.} \end{cases}$$

Our results in terms of success score on the test set of each base instance are reported in Figure 24. We were able to classify with a good success score 26 base instances over 36. For the remaining 10: TNR was below 0.5 for `gmu-35-40` and `timtab2`, TPR was below 0.5 for `glass4`, `m100n500k4r1`, `manna81`, `mine-166-5`, `mine-90-10`, `pigeon-10`, `reblock67`, `rmine6`.

This could depend on many factors. First, features of the dataset may represent well only a subset of the available base instances. Second, the positive decompositions of an unsuccessful base instance may not have common characteristics, i.e., more decompositions are needed to catch possible similarities between them. This may depend on the structure of the base instance or on the capabilities of the random sampling algorithm described in Section 2. Third, training the classifier using every available base instance may be detrimental to some if their structure is totally different from the most common one. Although further research is needed to explain these factors, classification shows potential and could be used with good accuracy if enough data about a given base instance is present in the dataset.

	Tolerance L				Tolerance H			
Instance	TN	FP	FN	TP	TN	FP	FN	TP
10teams	164	28	1	3	133	48	0	6
aflow30a	207	39	0	3	183	45	0	14
aflow40b	167	37	1	13	136	31	3	18
beasleyC3	149	99	0	3	169	68	20	31
csched010	185	59	1	5	169	72	2	12
enlight13	216	32	1	1	215	33	3	7
fiber	214	34	0	2	199	49	0	3
fixnet6	204	45	0	2	199	48	0	3
gesa2	138	98	1	14	102	86	1	3
gesa2-o	136	96	2	17	18	161	14	49
glass4	244	4	1	0	244	3	2	70
gmu-35-40	38	205	2	6	37	188	2	0
m100n500k4r1	236	10	1	0	179	64	4	22
manna81	208	0	43	0	110	0	2	2
mcsched	238	4	4	5	99	57	140	0
mine-166-5	138	104	6	3	142	75	25	70
mine-90-10	181	53	12	5	139	40	25	9
mkc	125	120	0	6	105	55	52	20
modglob	190	53	1	7	137	85	32	59
neos-686190	197	45	1	7	177	49	2	27
noswot	233	17	0	1	239	11	3	22
p2756	221	24	3	3	160	65	1	0
pigeon-10	250	0	1	0	245	4	11	15
pp08aCUTS	157	90	0	4	115	67	1	0
pp08a	204	44	0	3	179	52	20	48
pw-myciel4	187	62	0	1	182	67	5	14
ran16x16	158	18	0	3	150	24	0	2
reblock67	230	12	7	2	204	2	0	5
rmine6	238	2	10	1	0	200	43	2
rococoC10-001000	213	37	0	1	139	108	0	51
rout	54	3	0	2	49	7	1	2
set1ch	208	39	1	2	139	39	0	3
timtab1	171	70	1	8	193	41	23	50
timtab2	103	144	1	3	99	147	11	6
tr12-30	198	50	0	3	36	110	2	3
vpm2	191	58	0	2	198	47	8	97
Total	6591	1835	102	141	5219	2248	458	745

Table 1: Testing results

		Aggregation by Instance			Aggregation by Cluster		
Tolerance	k	Support	Average	Max	Support	Average	Max
	5	2,78%	3,00	3	40,00%	17,50	33
Low	36	30,56%	2,64	9	27,78%	2,80	8
	50	44,44%	3,25	10	34,00%	2,29	4
	5	2,78%	3,00	3	40,00%	17,50	33
High	36	38,89%	2,93	8	33,33%	3,25	11
	50	52,78%	3,00	10	28,00%	2,71	5

Table 2: Clustering profiling

6 Clustering

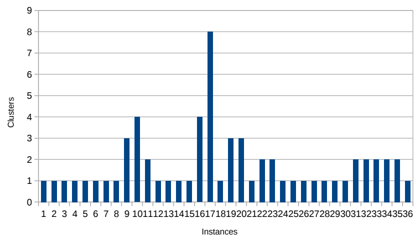
Our classification analysis highlighted that the the single base instance structure has a fundamental impact on the prediction potential. It would be therefore tempting to conclude that best results would be obtained by independently training a different model for each base instance, as similarities between different base instances would be negligible. We therefore performed experiments on a similar setting, surprisingly observing a *worsening* in the accuracy when different models were trained on each base instance independently.

In an effort for understanding such a phenomenon, we tried to assess the inter base instance relationships through cluster analysis. We performed it in two phases: cluster profiling and cluster-based classification.

6.1 Cluster profiling

In the first phase, we used a standard k-means model to find clusters of *positive decompositions only.* We first performed analyses by setting $k = 36$, that is allowing as many clusters as base instances, and then consider also $k = 5$ and $k = 50$ as representatives for aggressive and loose aggregation scenarii. The k-means heuristics implementation of the e1071 R library was used [21], performing 1000 random restarts, each involving 10000 iterations. Data was randomly shuffled before the execution of the algorithm and experiments were repeated to test both tolerance L and tolerance H.

Our results are presented in Table 2 for each tolerance setting (first column) and for each target values of clusters k. The table is composed by two blocks. In the left one we present aggregated results by base instance, including the overall percentage of base instances whose decompositions are assigned to more than one cluster (Support), the average number of clusters to which such split base instances are assigned (Average) and the number of clusters to which the most split base instance is assigned (Max). In the right block we report similar details, but aggregated by cluster: the overall percentage of clusters containing decompositions of more than a single base



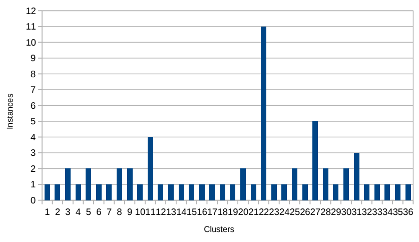


Figure 25: Clustering with k = 36, tolerance H. Aggregation by base instance (left) and by cluster (right).

instance (Support), the average number of base instances having decompositions which are assigned to a heterogeneous cluster (Average) or to the most heterogeneous one (Max). The case $k = 5$ is representative of a general phenomenon: independently on the chosen tolerance, a single base instance has decompositions assigned to different clusters, and only two clusters contain decompositions of more than a single instance, one of which collecting decompositions of 33 base instances; the average number of base instances for each cluster is therefore high only because of that single high value.

The case $k = 36$ is a key test. In addition to the results in the table, in Figure 25 we report the details for the test with Tolerance H. In the left (resp. right) figure for each base instance (resp. each cluster) we report the number of clusters containing decompositions of that base instance (resp. the number of base instances having decompositions in that cluster).

Remarkably, even when k is set equal to the number of base instances, the best clustering option is not to always put decompositions of the same base instances in the same cluster. More in details, decompositions of a same base instance are often assigned to different clusters (left figure): complete intra base instance similarity is not always guaranteed. In these cases, only subsets of decompositions are similar. Results also confirm (right figure) that often clusters aggregate decompositions of more than one base instance. This means that inter base instance similarity is non negligible.

The same happens allowing for more clusters ($k = 50$), proving such a phenomenon not to disappear even when intra base class splits are possible without the need of balancing by mixing decompositions of different base instances.

Clustering behaves similarly for tolerance L and tolerance H. With 5 clusters most of the decompositions are aggregated into a single cluster. When the number of clusters increases, aggregation is discouraged but is still present in about one third of the clusters. When aggregation on base instances is analyzed, support and max values grow linearly with the number of clusters; the average value is however independent from it and is always close to 3.

		Median		Third Quartile	
Tolerance	k	TNR	TPR	TNR	TPR
	5	0,39	0,51	0,22	0,77
Low	36	0,66	0,53	0,48	0,74
	50	0,61	0,52	0,42	0,74
	5	0,48	0,50	0,23	0,76
High	36	0,74	0,52	0,61	0,73
	50	0,63	0,50	0,43	0,74

Table 3: Testing classifier accuracy

6.2 Cluster-based classification

Following the rationale that negative decompositions should be distant from the positive ones, in the second part of the experiment we used the clusters found in the previous step in a nearest-neighbor based classifier.

That is, our classification model is a collection I of hyperspheres in the decompositions feature space, representing regions where *positive* decompositions cluster. Training consists in finding the centers c_i and radii t_i of these hyperspheres in the decompositions feature space.

Then classification of an arbitrary decomposition works as follows: if the decomposition falls within any hypersphere i in I, that is if its distance in the feature space with respect to the corresponding center c_i is less than or equal to the corresponding threshold t_i, then the decomposition is classified as positive. Otherwise, the decomposition is classified as negative.

In our experiments, training was performed by sampling 75% of the *positive* decompositions of every base instance, building clusters and finding cluster centers with them, and finally computing thresholds. Two possible thresholds were experimented, setting each t_i to either the median distance from decompositions in the training set assigned to cluster i, or the third quartile of the corresponding distribution. Testing was then performed using the remaining 25% of positive decompositions, and all the negative ones. The Table 3 reports the TNR and TPR for each tolerance setting, each value of k and each threshold selection policy.

The true negative rate mostly depends on the number k of clusters: 5 are not enough to guarantee good accuracy as aggregating a high number of instances means negative decompositions to fall more easily (and inappropriately) within the thresholds. Allowing a higher number of clusters guarantees better performance. In our tests, the classifier worked best when using 36 clusters and thresholds equal to the third quartile for each cluster. Changing the values of the thresholds has a direct impact on the true positive and true negative rates. Positive samples and negative ones are not completely disjunct and choosing the threshold values resorts in a trade-off between TNR and TPR.

To further estimate the impact of intra-base instance relationships, we repeated our experiments by keeping the training phase identical, but artificially forcing during the test phase each decomposition to be either assigned to one of the clusters containing training elements of its base instance, or classified as negative. The corresponding TNR for each base instance is reported in figures 27-30 for tolerance H with k equal to 5, 36, and 50 clusters. The TPR is instead implicitly fixed when choosing the threshold.

We first observed that negative decompositions are seldom misclassified by clusters to which no positive decomposition of the corresponding base instance was assigned during training, that is they are far from their centers. Strictly speaking, the TNR cannot decrease, as we are comparing each decomposition with a subset of I, thereby decreasing the cases allowing it to be marked as positive. It could increase, as decompositions previously falling into hyperspheres of alternative clusters are instead classified as negative. However, we experimentally observed it to increase only marginally (less than 1%), suggesting inter cluster dissimilarity to be high, and therefore confirming a inter base instance cluster tendency in our dataset. That also suggests, in order to improve our understanding of positive and negative decompositions, to focus more on improving intra-cluster classification, as different clusters already encode well distinct structures.

Second, the TNR improvements obtained by using a lower threshold are different from instance to instance and better results could be obtained in every test by calibrating the threshold differently for each cluster, eventually reaching even perfect classification for some instances. In these cases, it may be possible to increase the threshold to classify better positive samples without losing accuracy.

Third, some base instances that had TNR equal to 0 in one experiment have higher accuracy in others in which the number of clusters and their structure is different. That confirms that aggregation between different base instances helps classification. A remarkable, yet counter-intuitive, example is instance 6 (`enlight13`), that on the $k = 50$ test had TNR equal to 0, while by lowering $k = 36$ *improves* its TNR to 1 (i.e., by forcing more aggregation we improve accuracy). Of course, a deeper analysis would be needed to understand if such a property is structural, or just a coincidence, or even an artifact of our random sampling algorithm.

Finally, aggregation allows to classify with good accuracy negative samples that could not be classified at all with a SVM model. To support such a claim, we highlight the case with tolerance H, $k = 36$, t_i set to the median of distances: SVM had ten base instances with either TNR or TPR below 0.5 (see Figure 24); the nearest-neighbor classifier, instead, has only two base instances with TNR below 0.5, and TPR always about 0.5 by definition (see Figure 28).

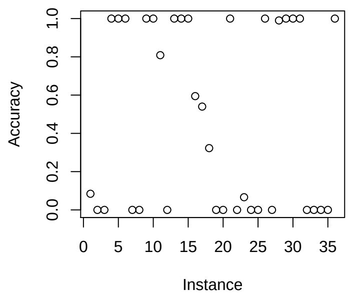


Figure 26: Testing TNR, 5 clusters, t = median, tolerance H

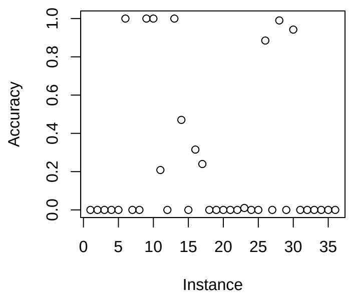


Figure 27: Testing TNR, 5 clusters, t = 3Q, tolerance H

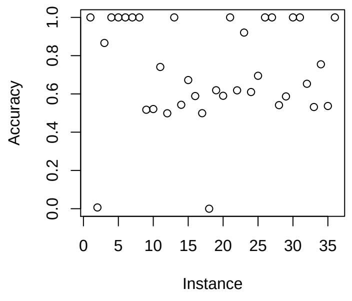


Figure 28: Testing TNR, 36 clusters, t = median, tolerance H

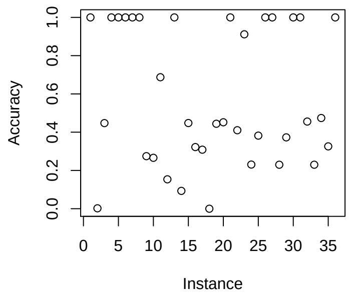


Figure 29: Testing TNR, 36 clusters, t = 3Q, tolerance H

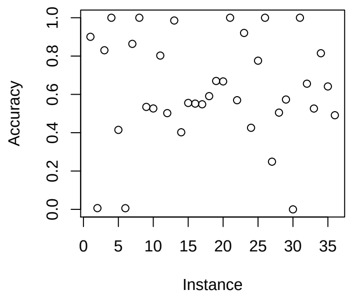


Figure 30: Testing TNR, 50 clusters, t = median, tolerance H

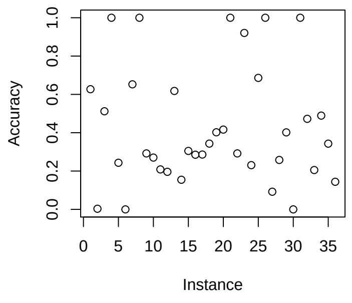


Figure 31: Testing TNR, 50 clusters, t = 3Q, tolerance H

7 Perspectives

In this paper we faced the ambitious task of understanding which features make a decomposition pattern for an arbitrary MIP appealing. We employed a machine learning approach, creating a large, balanced and heterogeneous dataset of random decompositions, explicitly solving the corresponding relaxations, computing suitable *a-posteriori* scores to be used as ground truth, and then trying to distinguish automatically good decompositions from bad ones by considering *a-priori* features of the MIP instances as predictors.

As expected, the task turned out to be rather complex: we found no model that was able to generalize completely, providing reasonable accuracy on decompositions for a certain base instance, if such a base instance was never seen before. Instead, machine learning models show potential when used to detect good decompositions for base instances that are *similar* to previously seen ones, which is indeed the normal working condition of a general purpose solver, where the user often optimizes several instances of very few, often similar, problems.

Furthermore, our experiments proved that there is no single feature, not even those proposed in the literature, that shows enough predictive power when used alone. At the same time, by combining different features we obtained encouraging results: in 72% of the base instances we considered, it was possible to automatically detect if a decomposition was promising or not with good accuracy and precision. It was even possible to observe, although only as a qualitative proof-of-concept, that a-posteriori scores can be estimated as a function of a-priori features by means of regression techniques. Understanding the reasons behind the misclassification in the remaining 28% of the base instances is still an open question and possibly requires more representative features, more data and more experiments, with focus more on base instance classification than on decomposition classification.

Another promising result is the assessment of cluster tendency among decompositions related to different base instances. That is, our experiments highlighted that good and bad decompositions clearly share similarities and dissimilarities *across base instances*, thereby suggesting the search through automatic tools for *classes* of good decomposition patterns which are *independent* on the particular base instance.

Besides clarifying these aspects, the next research step would be to extract the parameters of our predictive models, and use them in *optimization* algorithms for finding good decompositions. We finally remark that, although we tried through all our experiments to keep unbiased input (that is, promising and unpromising base instances, good and bad decompositions), in our vision the generic decompositions search methods found by machine learning are meant to be integrated with existing alternative ones from the literature. A cascade approach might be appropriate, such that whenever a new base instance is provided, the solver tries first of all to detect previously known structures, then to detect if the decomposition approach might be pertinent or not, and only as a final step to automatically generate a good decomposition pattern.

References

[1] G. Desaulniers, J. Desrosiers, M.M. Solomon (Eds.) "Column Generation", Springer, Berlin (2005)

[2] L. Wolsey "Integer Programming", Wiley (1998)

[3] M. Bergner, A. Caprara, A. Ceselli, F. Furini, M. Lübbecke, E. Malaguti, E. Traversi "Automatic Dantzig—Wolfe reformulation of mixed integer programs", Mathematical Programming A, 149(1-2): 391–424 (2015)

[4] IBM Cplex webpage: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html, (last access Aug. 2016)

[5] GUROBI webpage: http://www.gurobi.com (last access Mar. 2017)

[6] FICO xpress webpage: http://www.fico.com/en/products/fico-xpress-optimization-suite (last access Mar. 2017)

[7] T. Achterberg, T. Koch, A. Martin "MIPLIB 2003" Operations Research Letters, 34(4):361-372 (2006)

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

[9] G. Gamrath and M.E. Lübbecke "Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs", LNCS 6049:239-252 (2010)

[10] T. Achterberg "SCIP: solving constraint integer programs", Mathematical Programming Computation 1(1): 1–41 (2009)

[11] J. Wang and T. Ralphs "Computational experience with hypergraph-based methods for automatic decomposition in discrete optimization" C. Gomes and M. Sellmann (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. LNCS 7874:394–402 (2013)

[12] M. Kruber, M.E. Luebbecke and A. Parmentier "Learning when to use a decomposition", RWTH technical report 2016-037 (2016).

[13] F. Vanderbeck and L. Wolsey "Reformulation and decomposition of integer programs", in M. Jünger, Th.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey (eds), 50 Years of Integer Programming 1958–2008. Springer, Berlin (2010).

[14] F. Vanderbeck "BaPCod – a generic branch-and-price code", `https://wiki.bordeaux.inria.fr/realopt/pmwiki.php/Project/BaPCod`, (last access Mar. 2017)

[15] T.K. Ralphs and M.V. Galati "DIP – decomposition for integer programming" `https: //projects.coin-or.org/Dip`, (last access Mar. 2017)

[16] J. Puchinger, P.J. Stuckey, M.G. Wallace, and S. Brand "Dantzig-Wolfe decomposition and branch-and-price solving in G12", Constraints 16(1):77–99 (2011).

[17] C. Burges "A Tutorial on Support Vector Machines for Pattern Recognition", Data Mining and Knowledge Discovery 2(2): 121–167 (1998)

[18] A.J. Smola and B. Scholkopf "A tutorial on support vector regression", Statistics and Computing 14:199–222 (2004)

[19] D.T. Larose, C.D. Larose "Data Mining and Predictive Analytics", Wiley (2015)

[20] H. Abdi, L.J. Williams "Principal Component Analysis", Wiley Interdisciplinary Reviews: Computational Statistics, 2(4): 433-459 (2010)

[21] J. A. Hartigan and M. A. Wong "Algorithm AS 136: A K-Means Clustering Algorithm", Journal of the Royal Statistical Society. Series C, 28(1):100-108 (1979)

[22] S. Basso, A. Ceselli "Asynchronous Column Generation", Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX), 197-206 (2017)

[23] J.P. Brooks and E.K. Lee "Analysis of the consistency of a mixed integer programming-based multi-category constrained discriminant model", Annals of Operations Research 174(1):147–168 (2010)

[24] E. B. Khalil "Machine Learning for Integer Programming", Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (2016)

[25] F. Hutter, L. Xu, H.H. Hoos, K. Leyton-Brown "Algorithm runtime prediction: Methods & evaluation", Artificial Intelligence 206(1):79-111 (2014)

[26] A. Schrijver “Theory of Linear and Integer Programming” John Wiley & Sons (1998)

[27] R Core Team “R: A Language and Environment for Statistical Computing”, R Foundation for Statistical Computing, `https://www.R-project.org/` (2016)

A Appendix

A.1 Base instances of the Dataset

The dataset is composed of the following base optimization problem instances:

1. 10teams
2. aflow30a
3. aflow40b
4. beasleyC3
5. csched010
6. enlight13
7. fiber
8. fixnet6
9. gesa2
10. gesa2-o
11. glass4
12. gmu-35-40
13. m100n500k4r1
14. manna81
15. mcsched
16. mine-166-5
17. mine-90-10
18. mkc
19. modglob
20. neos-686190
21. noswot
22. p2756
23. pigeon-10

24. pp08aCUTS
25. pp08a
26. pw-myciel4
27. ran16x16
28. reblock67
29. rmine6
30. rococoC10-001000
31. rout
32. set1ch
33. timtab1
34. timtab2
35. tr12-30
36. vpm2

Initially, further base optimization instances (`macrophage`, `harp2`, `opt1217`) were considered but they were discarded during preprocessing operations.

A.2 Features of the Dataset

For each base optimization problem instance the following features were measured:

- number of variables
- number of generic integer variables
- number of binary variables
- number of continuous variables
- total number of constraints
- number of equality constraints
- number of inequality constraints

For each decomposition we instead measured:

- number of blocks
- average, min, max, standard deviation on the number of variables in blocks
- average, min, max, standard deviation on the number of generic integer variables in blocks
- average, min, max, standard deviation on the number of binary variables in blocks
- average, min, max, standard deviation on the number of continuous variables in blocks
- average, min, max, standard deviation on the number of constraints in blocks
- average, min, max, standard deviation on the density of blocks (fraction of nonzero coefficients)
- average, min, max number of equality constraints in blocks
- average, min, max number of inequality constraints in blocks
- average, min, max standard deviation of mean constraints right hand side coefficients (rhs) in blocks
- average, min, max standard deviation of rhs ranges (max rhs - min rhs) in blocks

- average, min, max, standard deviation of blocks “shape” (number of variables divided by the number of constraints in each block)
- average, min, max, standard deviation of “Total Unimodularity Coefficient” of blocks
- average, min, max, standard deviation of mean objective function coefficients in each block
- average, min, max, standard deviation of the objective function coefficients range (maximum coefficient - minimum coefficient) in each block
- number of blocks with both positive and negative coefficients in the objective function
- number of variables in the border
- number of generic integer variables in the border
- number of binary variables in border
- number of continuous variables in border
- number of constraints in border
- density of border (fraction of nonzero coefficients)
- number of equality constraints in border
- number of inequality constraints in border
- average, stdandard deviation and range of rhs in the border