Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Improving the Efficiency of Dynamic Programming on Tree Decompositions via Machine Learning

Michael Abseher, Frederico Dusberger, Nysret Musliu and Stefan Woltran
Institute of Information Systems 184/2
Vienna University of Technology
Favoritenstraße 9–11, 1040 Vienna, Austria

Abstract

Dynamic Programming (DP) over tree decompositions is a well-established method to solve problems – that are in general NP-hard – efficiently for instances of small treewidth. Experience shows that (i) heuristically computing a tree decomposition has negligible runtime compared to the DP step; (ii) DP algorithms exhibit a high variance in runtime when using different tree decompositions; in fact, given an instance of the problem at hand, even decompositions of the same width might yield extremely diverging runtimes. We thus propose here a novel and general method that is based on a selection of the best decomposition from an available pool of heuristically generated ones. For this purpose, we require machine learning techniques based on features of the decomposition rather than on the actual problem instance. We report on extensive experiments in different problem domains which show a significant speedup when choosing the tree decomposition according to this concept over simply using an arbitrary one of the same width.

1 Introduction

The notion of treewidth – and, as basic underlying concept, tree decompositions – was introduced by Bertelè and Brioschi [1973], Halin [1976] and Robertson and Seymour [1984]. Many NP-hard problems become tractable for instances whose treewidth is bounded by some constant k [Arnborg and Proskurowski, 1989; Niedermeier, 2006; Bodlaender and Koster, 2008]. A problem exhibiting tractability by bounding some problem-inherent constant is also called fixed-parameter tractable (FPT) [Downey and Fellows, 1999]. While constructing an optimal tree decomposition, i.e. a decomposition with minimal width, is intractable [Arnborg *et al.*, 1987], researchers proposed several exact methods for small graphs and efficient heuristic approaches that usually construct tree decompositions of almost optimal width for larger graphs (see recent surveys, e.g. [Bodlaender and Koster, 2010; Hammerl *et al.*, 2015]). Tree decompositions have been used for several applications including inference problems in probabilistic networks [Lauritzen and Spiegelhalter, 1988], frequency assignment [Koster *et al.*, 1999], computational biology [Xu *et al.*, 2005], Answer-Set Programming [Morak *et al.*, 2012], etc.

A promising technique for solving problems using this concept is the computation of a tree decomposition followed by a dynamic programming (DP) algorithm that traverses the nodes of the decomposition and consecutively solves the respective sub-problems [Niedermeier, 2006]. The general runtime of these algorithms for an instance of size n is $f(k) \cdot n^{\mathcal{O}(1)}$, where f is an arbitrary function of width k of the used tree decomposition. However, experience shows that even decompositions of the same width lead to significant differences in the runtime of DP algorithms. Recent research results confirm that the width is indeed not the only important parameter that has a significant influence on the runtime. Morak *et al.* [2012], for instance, suggested that the consideration of further properties of tree decompositions is important for the runtime of DP algorithms for answer set programming. In another paper, Jégou and Terrioux [2014] observed that the existence of multiple connected components in the same tree node (bag) may have a negative impact on the efficiency of solving constraint satisfaction problems.

In this paper we therefore want to gain a deeper understanding of the impact of tree decompositions on the runtime of DP algorithms by conducting extensive experiments on different problem domains, instances and tree decompositions. Moreover, we run our experiments on two inherently different prominent systems that apply DP algorithms on tree decompositions. We propose a new set of tree decomposition features that allows for a reliable prediction of the influence of a given tree decomposition on the performance of DP algorithms. To select the most promising tree decomposition from a pool of generated ones using those features we consider the application of machine learning techniques. Using fast constructive heuristics the overhead for generating the different tree decompositions is negligible in comparison to the gain in overall runtime. Our experiments show a significant benefit of selecting a decomposition that is promising according to our prediction in contrast to simply choosing an arbitrary one. Furthermore, the results confirm that our approach is generally applicable, independent from the particular solver and problem domain. Finally, the results provide valuable insights for laying the foundation to construct customized decompositions optimizing the relevant features.

Recently, researchers have used successfully machine learning for runtime prediction and algorithm selection on several problem domains. Such problems include SAT [Xu *et al.*, 2008; Hutter *et al.*, 2014], combinatorial auctions [Leyton-Brown *et al.*, 2009], TSP [Hutter *et al.*, 2014], Graph Coloring [Musliu and Schwengerer, 2013], etc. For surveys on this domain, see e.g. [Smith-Miles, 2008; Hutter *et al.*, 2014]. Our research gives new contributions in this area, as to the best of our knowledge, this is the first application of machine learning techniques towards the optimization of tree decompositions in DP algorithms.

2 Background

In the following we give a formal definition of tree decompositions and treewidth and illustrate the principle of DP on such decompositions. Furthermore, we provide a short overview on D-FLAT [Abseher *et al.*, 2014] and SEQUOIA [Kneis *et al.*, 2011], highlighting how each of them realizes DP on tree decompositions.

Tree decomposition is a technique often applied for solving NP-hard problems. The underlying intuition is to obtain a tree from a (potentially cyclic) graph by subsuming multiple vertices in one node and thereby isolating the parts responsible for the cyclicity. Formally, the notions of tree decomposition and treewidth are defined as follows [Robertson and Seymour, 1984; Bodlaender and Koster, 2010].

Definition 1 *Given a graph* $G = (V, E)$*, a* tree decomposition *of* G *is a pair* (T, χ) *where* $T = (N, F)$ *is a tree and* $\chi : N \to 2^V$ *assigns to each node a set of vertices (called the node's* bag*), such that the following conditions hold:*

1. *For each vertex* $v \in V$*, there exists a node* $i \in N$ *such that* $v \in \chi(i)$*.*
2. *For each edge* $(v, w) \in E$*, there exists an* $i \in N$ *with* $v \in \chi(i)$ *and* $w \in \chi(i)$*.*
3. *For each* $i, j, k \in N$*: If* j *lies on the path between* i *and* k *then* $\chi(i) \cap \chi(k) \subseteq \chi(j)$*.*

The width *of a given tree decomposition is defined as* $\max_{i \in N} |\chi(i)| - 1$ *and the* treewidth *of a graph is the minimum width over all its tree decompositions.*

Note that the tree decomposition of a graph is in general not unique. In the following we consider rooted tree decompositions, for which additionally a root $r \in N$ is defined.

Definition 2 *Given a graph* $G = (V, E)$*, a* normalized *(or* nice*) tree decomposition of* G *is a rooted tree decomposition* T *where each node* $i \in N$ *is of one of the following types:*

1. Leaf*:* i *has no child nodes.*
2. Introduce Node*:* i *has one child* j *with* $\chi(j) \subset \chi(i)$ *and* $|\chi(i)| = |\chi(j)| + 1$
3. Forget Node*:* i *has one child* j *with* $\chi(j) \supset \chi(i)$ *and* $|\chi(i)| = |\chi(j)| - 1$
4. Join Node*:* i *has two children* j, k *with* $\chi(i) = \chi(j) = \chi(k)$

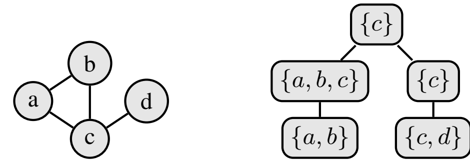
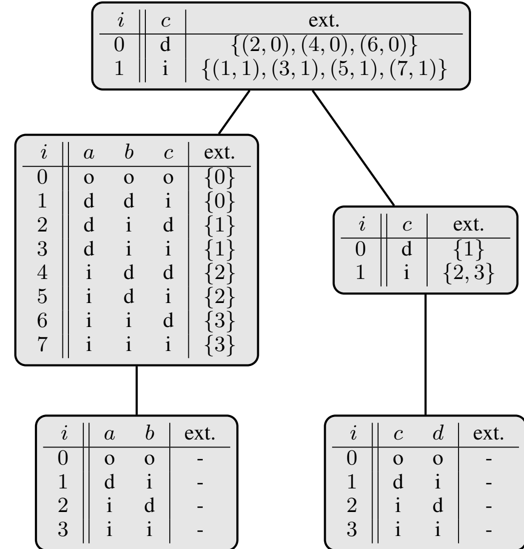


Figure 1: Example graph and a possible tree decomposition.



i	c	ext.
0	d	$\{(2,0),(4,0),(6,0)\}$
1	i	$\{(1,1),(3,1),(5,1),(7,1)\}$

i	a	b	c	ext.
0	o	o	o	{0}
1	d	d	i	{0}
2	d	i	d	{1}
3	d	i	i	{1}
4	i	d	d	{2}
5	i	d	i	{2}
6	i	i	d	{3}
7	i	i	i	{3}

i	c	ext.
0	d	{1}
1	i	{2, 3}

i	a	b	ext.
0	o	o	-
1	d	i	-
2	i	d	-
3	i	i	-

i	c	d	ext.
0	o	o	-
1	d	i	-
2	i	d	-
3	i	i	-

Figure 2: DP tables for the problem instance in Figure 1.

Each tree decomposition can be transformed into a normalized one in linear time without increasing the width [Kloks, 1994]. Figure 1 gives an example of a graph and a possible (non-normalized) tree decomposition of it.

For graph problems and problems that can be formulated on a graph, tree decompositions permit a natural way of applying DP by traversing the tree from the leaf nodes to its root. For each node $i \in N$ solutions for the subgraph of the instance graph induced by the vertices in $\chi(i)$ are computed. When traversing to the next node the (partial) solutions computed for its children are taken into account, such that only consistent solutions are computed. Thus the partial solutions computed in the root node are consistent to the solutions for the whole graph. Finally, the complete solutions are obtained in a reverse traversal from the root node to the leaves combining the computed partial solutions. Figure 2 shows the tables computed in a DP algorithm for solving the DOMINATING SET problem on the graph given in Figure 1. The central columns of each table list the possible values for the vertices in the node constituting the partial solutions. Here, *i*, *d* and *o* stand for the respective vertex being *in* the selected set, *dominated* by being adjacent to a vertex from the set or simply *out*. The last column stores the possible partial solutions from the child node(s) that can consistently be extended to these values. Note that infeasibilities in the partial solutions can already lead to an early removal of solution candidates. The partial solution $\{c = \text{o}, d = \text{o}\}$ can, for instance, already be discarded when d is forgotten

during the traversal to the next node, since d cannot appear again in any further node and can thus not become dominated or a part of the solution set anymore. For the sake of simplicity it is common practice to define DP algorithms on normalized tree decompositions [Bodlaender and Koster, 2008; Kneis *et al.*, 2011]. Thus, in the following, we refer to normalized decompositions unless stated otherwise.

In our experiments we use two systems that apply DP on tree decompositions, D-FLAT and SEQUOIA.

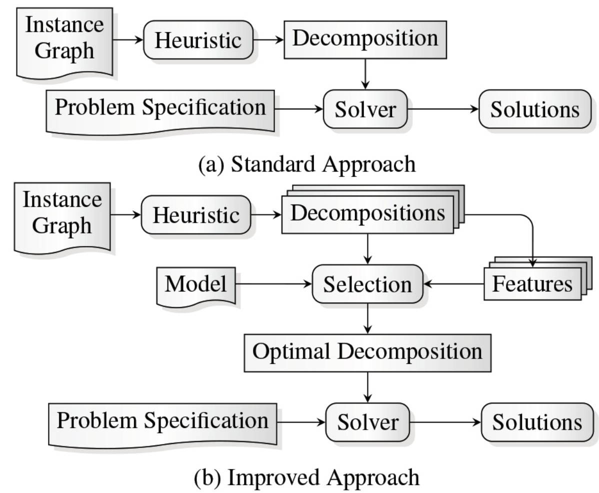
D-FLAT is a general framework capable of solving any problem expressible in monadic second-order logic (MSO) in FPT time w.r.t. the parameter treewidth. The D-FLAT system combines DP on tree decompositions with answer set programming (ASP) [Brewka *et al.*, 2011]. Given an ASP encoding Π of the DP algorithm and an instance graph G, D-FLAT first constructs a tree decomposition from G in polynomial time using internal heuristics. If desired, this decomposition can be left untouched or be normalized in a preprocessing step. The decomposition is then traversed bottom-up in the manner described above, i.e. at each node the program specified by Π and the currently known facts for the vertices in that node's bag is solved. Notice that due to its internal structure D-FLAT always stores all partial solutions in its DP tables and thus generally solves the problem of enumerating all solutions to a given instance.

An alternative approach also based on DP on normalized tree decompositions is realized in the SEQUOIA framework. Similarly to D-FLAT, SEQUOIA is a general solver that can be applied to any problem expressible in MSO. One general difference to D-FLAT is that it expects the problem definition directly in terms of an MSO formula, rather than an ASP program and that this formula has to describe the problem in a monolithic manner. In other words, the "decomposition" of the formula into the parts relevant to the separate tree decomposition nodes, as well as the special considerations on how partial solutions from child nodes are considered, are done internally. The system is based on the model checking game in which the verifier tries to prove that a given formula holds on the input graph while the falsifier tries to refute it [Hintikka, 1973]. Another major difference is that the dynamic programming algorithm relies on lazy evaluation and visits the nodes in the tree decomposition only on demand. SEQUOIA always outputs one (optimal) solution and does not enumerate all possibilities.

3 Improving the Efficiency of DP Algorithms

Systems, such as SEQUOIA and D-FLAT, follow a straightforward approach for using tree decompositions: a single decomposition is generated heuristically and then fed into the DP algorithm used in the system (see Figure 3a). However, experiments have shown that the "quality" of such tree decompositions varies, leading to differing runtimes for the same problem instance. Most interestingly, "quality" does not necessarily mean low width. Even tree decompositions of the same width lead to huge differences in the runtime of a DP algorithm when applied to the same problem instance.

The approach we propose in this work is illustrated in Figure 3b. The main idea is to generate a pool of tree decompositions for the given input instance and then to select, based on features of the decomposition, the one which promises best performance. The key aspects of the approach are as follows:



(a) Standard Approach

(b) Improved Approach

Figure 3: Comparison of Approaches

- Generating dozens of tree decompositions can be usually done within seconds by effective heuristics for tree decompositions (e.g. min-fill heuristic), thus the runtime overhead will be negligible in almost any case.
- Models allowing to predict the runtime behavior of a tree decomposition for a given DP algorithm are required. These models can be obtained in an off-line training-phase by running several instances with different tree decompositions and by storing the runtime and the feature values which are then processed by machine learning algorithms. For our purposes, machine learning techniques need to predict a good *ranking* of tree decompositions based on the predicted runtime for these decompositions. We note that a very accurate prediction of runtime is not crucial in our case, but rather predicting a correct order of tree decompositions. For example if the actual runtime of the DP algorithm using tree decomposition $TD1$ is faster than with $TD2$, it is important that machine learning algorithms predict that the runtime for $TD1$ is shorter than the runtime for $TD2$. In that case the order of tree decompositions is correct even if the runtime prediction for both tree decompositions is not completely exact.
- The main challenge and novel aspect of the approach is given by the fact that the features used to obtain these rankings need to be defined on the tree-decomposition not on the problem instance. To successfully exhibit machine learning techniques we need to find powerful features that characterize well the quality of tree decompositions. Moreover, the computation of these features needs to be done efficiently.

In other words, our approach works as follows: First, a number (which can be arbitrarily large) of decompositions of the given problem instance is computed and stored in a pool; in our concrete evaluation we shall generate ten tree

decompositions for the input problem. Second, the features (acting as explanatory variables) of these decompositions are extracted and used to predict the runtime as the response variable. This gives us a ranking from which we select the decomposition with minimal predicted runtime (in case of ties, we choose randomly one of the decompositions with minimal predicted runtime). We apply regression algorithms, such as linear regression, regression trees, nearest-neighbor, multilayer perceptrons and support-vector machines to generate the models for prediction. Finally, the selected decomposition is handed over to the actual system to run the DP algorithm.

In what follows we address one of the main contributions of the work, namely the identification of 70 tree decomposition features. We group them in six main categories: *decomposition size*, *decomposition structure*, as well as features of *introduce nodes*, *forget nodes*, *join nodes* and *leaf nodes*.

The features dealing with *decomposition size* measure the general complexity of the tree decomposition, i.e. statistics on the bag size of all nodes and the non-leaf nodes, the number of nodes a vertex appears in and the total number of vertices over all nodes.

The node type specific features contain statistics on the depth in the tree and the bag size for each respective node type. Additionally, the count and the percentage of those nodes among all nodes of the decomposition are proposed.

Finally, to characterize the structure of the decomposition we propose several new *structural features*. In particular these are statistics on the distance between consecutive join nodes, the number of consecutive bags an item appears in and the total number of children of a node. Furthermore the BalancednessFactor is a measure of how balanced the tree is, i.e. it is equal to 1 if the tree is perfectly balanced and approaches 0 the less balanced it is. In this category we also gather statistics on three more sophisticated features of nodes in a decomposition, namely:

- AdjacencyRatio: The ratio of the number of pairs of vertices in the bag that are adjacent in the original graph to the total number of vertex pairs.
- BagConnectednessRatio: The ratio of the number of pairs of vertices in the bag that are connected in the original graph to the total number of vertex pairs.
- NeighborCoverageRatio: For each vertex in the bag the ratio between the number of neighbors in the bag to the number of neighbors in the original graph is computed. Per bag, the mean of these ratios is considered.

Most of the presented features are computed independently per bag. To aggregate the per-bag outcome of a feature and in order to obtain the final set of more than 70 features, we consider the statistical functions minimum, maximum, median, 75%-percentile, mean as well as the standard deviation. A detailed description of all features can be found under the following link:
`www.dbai.tuwien.ac.at/research/project/dflat/features/features0415.pdf`

4 Experimental Evaluation

In this section, we experimentally evaluate the proposed method. All our experiments were performed on a single core of an AMD Opteron 6308@3.5GHz processor running Debian GNU/Linux 7 (kernel 3.2.0-4-amd64) and each test run was limited to a runtime of at most one hour.

We evaluate our approach using two recently developed DP solvers, D-FLAT (v. 1.0.1) and SEQUOIA (v. 0.9). The subsequent machine learning tasks were carried out with WEKA 3.6.11 [Hall *et al.*, 2009].

4.1 Methodology

Algorithms and Problems In our analysis we considered the following set of problems defined on an undirected graph $G = (V, E)$:

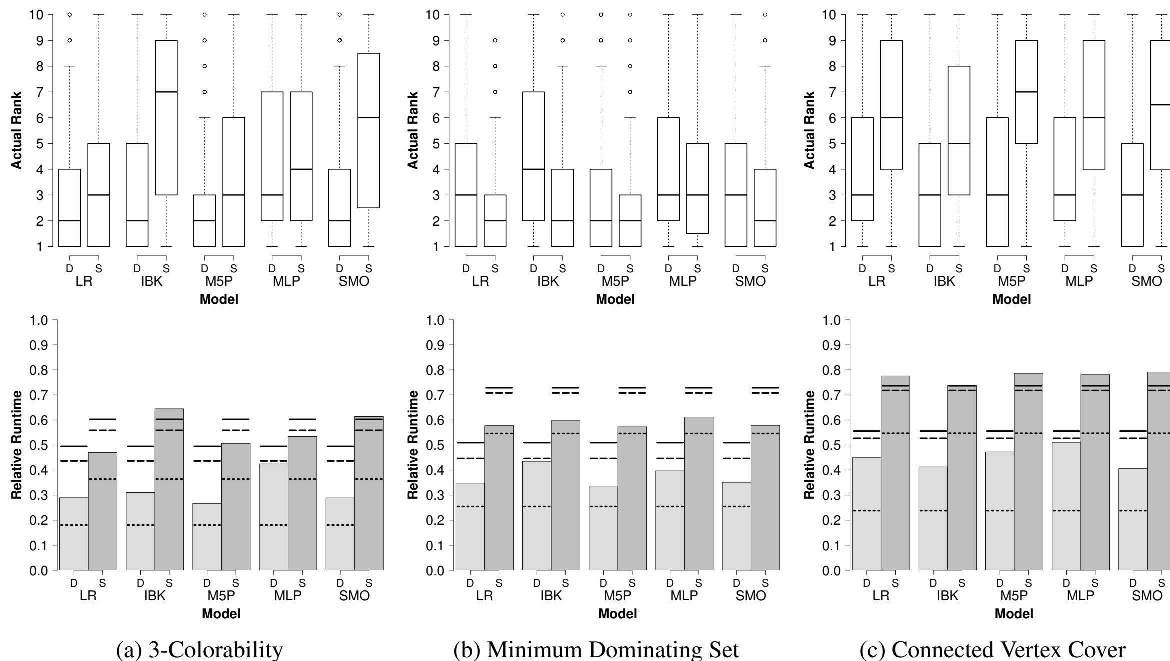
1. 3-COLORABILITY (3-COL): Is G 3-colorable?
2. MINIMUM DOMINATING SET (MDS): Find all sets $S \subseteq V$, s.t. $\forall u \in V : u \in S \vee (\exists(u, v) \in E \wedge v \in S)$ of minimal cardinality.
3. CONNECTED VERTEX COVER (CVC): Find all sets $S \subseteq V$, s.t. $\forall(u, v) \in E : u \in S \vee v \in S$) and the vertices in S form a connected subgraph of G.

Training Data We performed extensive experiments in order to collect the training data. The training dataset for each problem and solver consists of 900 tree decompositions which are created from 90 satisfiable[1] instances of different size and edge probability (three sizes, three edge probabilities with ten instances for each combination) generated based on the Erdős-Rényi graph model. The actual input data for computing the models is obtained as follows:

1. Generate problem instances: The actual sizes and edge probabilities may differ for solvers and problems as we aim for the goal that the runtime for the DP algorithm using the worst tree decomposition (= longest runtime) is close to the timeout limit of one hour.
2. Compute tree decompositions and extract all features
3. Execute DP algorithm to retrieve runtime and remove invalid measurements caused by violations of timeout or memory limit.
4. Group the feature measurements and the corresponding runtimes by instance and then standardize these values X feature-wise based on the formula $(X - \mu)/\sigma$.

As graphs of larger size and/or higher density in general lead to an increase in the time consumption of overlying DP algorithms, using different sizes and edge probabilities allows us to gain an unbiased view on the applicability of our proposed approach. In this paper we have chosen a random graph model as basis for our experiments, in order to cover a broad range of possible graphs. The full training dataset

[1]Not all generated instances are satisfiable for 3-COL or CVC. In our experiments for these problems, we consider only those that are, because for unsatisfiable instances, a large part of the tree decomposition might not even be visited by a DP algorithm. This is due to the fact that the algorithm terminates as soon as it is evident that no solution exists for the instance at hand. Therefore, unsatisfiable problem instances do not allow us to investigate the effect of decomposition selection on the runtime of DP algorithms and we thus omit them in our comparisons.



(a) 3-Colorability (b) Minimum Dominating Set (c) Connected Vertex Cover

Figure 4: Rank and Relative Runtime for Selected Tree Decompositions (Random Instances)

used for our experiments is available under the following link: `www.dbai.tuwien.ac.at/research/project/dflat/features/training.zip`

For each of the training instances we computed ten normalized tree decompositions on the basis of the min-fill-heuristic [Dechter, 2003] (we used the min-fill implementation from [Dermaku *et al.*, 2008]). Note that this is a randomized heuristic ensuring the diversity of the generated tree decompositions. Then, together with the problem specification – a D-FLAT encoding or, in case of SEQUOIA, an MSO-formulation – these decompositions were given to D-FLAT and SEQUOIA in order to solve the corresponding problem.

Afterwards, the outcome of these test runs in terms of their features and runtime has to be filtered. This was done by ignoring decompositions leading to the violation of time or memory constraints (about 5% per problem and solver). In a final step, all results were standardized (after grouping them by problem instance) to avoid a bias introduced by differing ranges of feature measurements between different instances.

Model Details For each problem domain and every solver we computed regression models for linear regression (LR), k-nearest neighbors (IBK), regression tree (M5P), multi-layer perceptron (MLP) and support vector machines (SMO). The results that we present in this paper are obtained with parameter settings that were chosen based on several of our previous experiments using 10-fold cross validation. To allow reasoning about the potential effect of model selection and/or parameter tuning, we used the same algorithm parameters for the different problems.

At this point we want to recall that the crucial property for a model is the capability of giving a correct ranking of the decompositions. This is because after a decomposition is selected, the DP algorithm still has to be run and it does not give much benefit to know the time this will take, as long as the selected decomposition is the fastest way to go.

This means that our approach also works for all models where only the correct order of the decompositions (based on their implied runtime) is preserved. This is in contrast to most "classical" regression tasks, where it is crucial to have a high correlation between the actual and predicted values.

4.2 Experimental Results

For evaluating the trained models we proceed as follows:

1. For each combination of problem and solver we generate 50 new, satisfiable instances just as described before.
2. For each instance ten normalized tree decompositions are computed, forming the pool of decompositions for the respective instance.
3. For each instance we employ each model separately to select the optimal decomposition from the pool for this instance.
4. We retrieve the runtime implied by each decomposition in the pool by running the DP algorithm.

To increase the significance of our results, ten iterations of this scheme are considered for each instance. Based on these steps, we obtain a ranking by sorting the decompositions in an ascending order based on the runtime they lead to when using them in the DP algorithm. Indeed, in practice, the DP algorithm will be executed only for the decomposition which is predicted as the best one.

Figure 4 as the central figure of this section summarizes all aspects of our evaluation graphically. Each of the sub-figures

deals with a separate problem type (3-Colorability, Minimum Dominating Set and Connected Vertex Cover) and shows the results for the solvers D-FLAT and SEQUOIA (abbreviated by the letters D and S) for each of the models (LR, IBK, M5P, MLP as well as SMO).

The first row of the figure uses box-plots to illustrate the rank the selected decomposition actually achieves within the ten decompositions in the pool for a problem instance. To get this actual rank, we use the ranking we obtained by running all ten decompositions. The decomposition which leads to the lowest runtime has rank 1 and the remaining decompositions are ranked ascendingly according to the runtime they lead to. Basically, we want the median (bold line within boxes) as close to the optimal rank 1 as possible and the inter-quartile-range (boxes) should be as small as possible.

Furthermore, our setup allows us to investigate the concrete impact of our approach on the runtime compared to average and median runtime of the ten decompositions.

The second row of Figure 4 shows the average runtime the decompositions selected by a model lead to via a bar-plot. Due to different runtimes of the DP algorithm for different instances, the range of the runtime is normalized per problem instance between 0 (0 seconds) and 1 (maximum runtime for instance). The height of the bars in the diagram corresponds to the complete runtime using our approach (runtime of the DP algorithm + time needed to generate the ten decompositions + the time for feature extraction and applying the model). The following limits are also provided and represent the average over the 50 problem instances:

- The dotted line stands for the minimum runtime within ten decompositions.
- The dashed line stands for the median runtime over the ten decompositions. Whenever the bar is below this limit, our approach works better than selecting a random tree decomposition.
- The full line stands for the average runtime over the ten decompositions.

With this information at hand, the big influence of the tree decomposition on the performance of the DP algorithm becomes immediately clear. We observe that the relative minimum runtime does not exceed 0.55, even in the worst case, which means that in most of the cases, the maximum runtime is at least twice as high as the minimum runtime.

4.3 Discussion

As the evaluation underlines, our machine learning approach shows great potential for improving the performance of DP algorithms. We can see that in general there is no "perfect" model which performs best in every case and that there exist differences between the problems. Below we give a list of main observations from our experiments:

- When looking at the two solvers separately, we see that for D-FLAT our approach leads to improved selection and an acceleration of the DP algorithm in any case. In relative values, we save about 20% of the runtime in the average case compared to the median runtime and in some cases (e.g. for 3-COL/M5P) more than 40% which we think is quite impressive. For single instances we sometimes saved more than 25 minutes of computation time and we saved about eight minutes in the average case.
- For SEQUOIA, we observe that for MDS some models are actually quite close to an oracle selecting always the best decomposition and also for 3-COL at least three of the five models showed noteworthy improvements. Unfortunately, for the problem of CVC we could not achieve improvements, which needs further investigation.
- The differences between minimum and median runtime are quite big and so is the potential for improvement. This justifies the use of tree decomposition selection to improve the efficiency of DP algorithms.
- The median runtime (dashed lines in Figure 4) is always lower than the average computation time (full lines) which is a strong hint that decompositions which lead to high runtimes are often much worse than "average" decompositions. Detecting such "bad" decompositions is an interesting classification task for future work.
- Comparing the performance of the used regression techniques we can conclude that no algorithm is outperforming all others. Although all used prediction algorithms could be used for improvement of our DP algorithms, in general our experiments indicate that LR (linear regression) and M5P (regression tree) give slightly better results when considering all problems and solvers.

Problem	Model				
	LR	IBK	M5P	MLP	SMO
3-COL (D)	0.9995	0.9995	0.9995	0.9750	0.9995
MDS (D)	0.9995	0.9995	0.9995	0.9995	0.9995
CVC (D)	0.9995	0.9995	0.9990	0.9750	0.9995
3-COL (S)	0.9995	—	0.9995	0.9750	—
MDS (S)	0.9995	0.9995	0.9995	0.9995	0.9995
CVC (S)	—	—	—	—	—

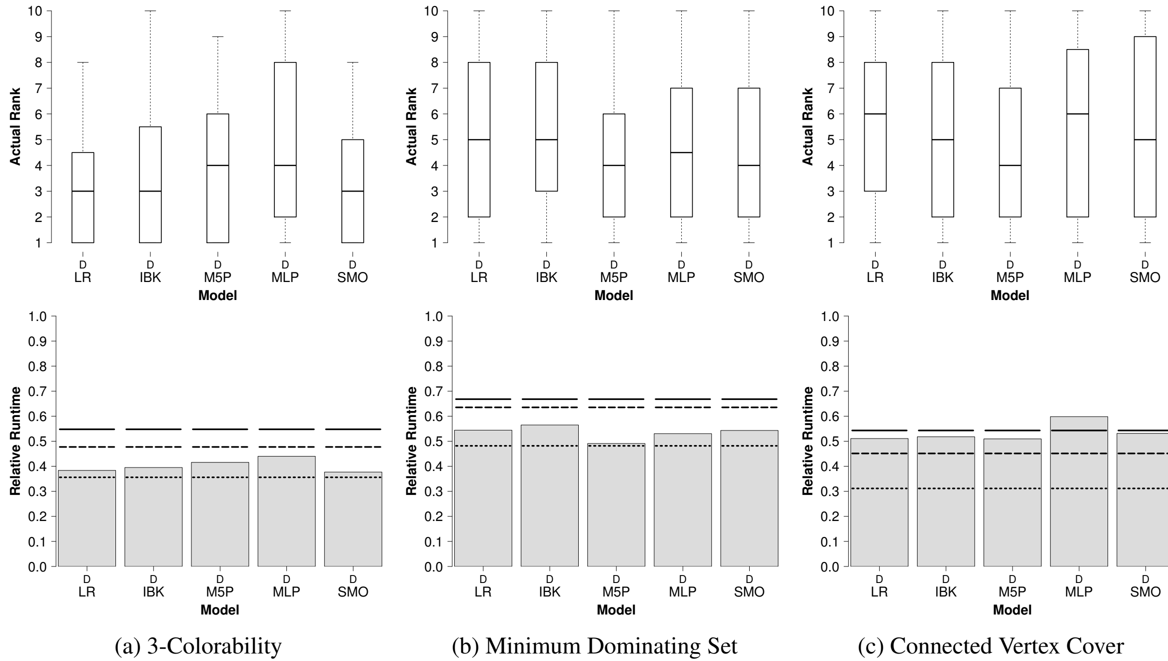
Table 1: Confidence Levels for Improvement
(D ... D-FLAT, S ... SEQUOIA)

Finally, we mention that most of the improvements we gained are statistically (highly) significant. Table 1 contains for each combination of problem and solver the confidence level for the hypothesis that our approach leads to a significantly better selected rank. The actual values for the table were obtained via an one-sided, paired t-test.

4.4 Experiments on Real-World Instances

Until now, we considered only randomly generated instances. With the goal of strengthening our findings, we additionally conducted a series of tests on real-world graphs. The dataset we used for these tests contains 21 instances of tree-width up to 8 from different domains.

Eight instances of our dataset represent metro and urban train networks of a selection of prominent cities: Beijing, Berlin, London, Munich, Shanghai, Singapore and Vienna,



(a) 3-Colorability (b) Minimum Dominating Set (c) Connected Vertex Cover

Figure 5: Rank and Relative Runtime for Selected Tree Decompositions (Real-World Instances)

where for the last city we also considered the tram network. The remainder of our dataset is formed by a subset of the graph collection [Batagelj and Mrvar, 2006] provided by the University of Ljubljana by considering real-world instances having up to 674 vertices and a tree-width up to 8. One important detail of the dataset is the fact that not all instances are actually 3-colorable or connected. Considering also these graphs allows us to investigate the behavior on unseen real-world problems accurately. The complete dataset of tested real-world instances is provided under the following link:
`www.dbai.tuwien.ac.at/research/project/dflat/features/realworld.zip`

The actual results of our tests are presented in Figure 5 which follows exactly the same scheme as the figure for the random graph instances (Figure 4). However, we had to omit SEQUOIA here, since this system in its current state did not support handling such large tree decompositions.

At first glance, we can see that our proposed approach gives good results also when applied to real-world instances, at least for the problems 3-COL and MDS. In the former case SMO achieves the best results and in case of MDS we obtain an almost perfect runtime behavior using the M5P-model.

Note that the results presented in Figure 5 are obtained using the models that were learned on the training set consisting of random graphs, which are of much smaller size. This not only demonstrates the versatility of the proposed method, but also leaves room for further possible improvements when adapting the training data. In particular, we expect to improve the runtime also in case of the CVC by using training data of similar size and structure.

5 Conclusion

In this work we studied the applicability of machine learning techniques to improve the runtime behavior of DP algorithms based on tree decompositions. To this end we identified a variety of tree decomposition features, beside the width, that strongly influence the runtime behavior. The models using those features for the selection of the optimal decomposition have been validated by means of an extensive experimental analysis including real-world instances from different domains. Hereby, we considered three different problems and two inherently different frameworks that employ DP algorithms on tree decomposition, namely the systems SEQUOIA and D-FLAT. Our approach showed a remarkable, positive effect on the performance with a high statistical significance. We thus conclude that turning the huge body of theoretical work on tree decompositions and dynamic programing into efficient systems highly depends on the quality of the chosen tree decomposition, and that advanced selection mechanisms for finding good decompositions are indeed crucial.

The models we obtained give us further insights about those features of tree decompositions that are most influential in order to reduce the runtime of DP algorithms. In upcoming work we thus plan to focus on implementations of new heuristics for constructing tree decompositions that optimize the relevant features. Therefore, the ultimate goal of this research perspective is to achieve the potential speed-up we have observed in our experiments by directly obtaining tree decompositions of higher quality and thus *without* the initial training step our method requires.

Acknowledgments

The work was supported by the Austrian Science Fund (FWF): P25607-N23, P24814-N23, Y698-N23.

References

[Abseher *et al.*, 2014] Michael Abseher, Bernhard Bliem, Günther Charwat, Frederico Dusberger, Markus Hecher, and Stefan Woltran. The D-FLAT System for Dynamic Programming on Tree Decompositions. In *JELIA*, volume 8761 of *LNAI*, pages 558–572. Springer, 2014.

[Arnborg and Proskurowski, 1989] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

[Arnborg *et al.*, 1987] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[Batagelj and Mrvar, 2006] Vladimir Batagelj and Andrej Mrvar. Pajek datasets, 2006. http://vlado.fmf.uni-lj.si/pub/networks/data/.

[Bertelè and Brioschi, 1973] Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. *Journal of Combinatorial Theory, Series A*, 14(2):137–148, 1973.

[Bodlaender and Koster, 2008] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[Bodlaender and Koster, 2010] Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208(3):259–275, 2010.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Dechter, 2003] Rina Dechter. *Constraint Processing*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, 2003.

[Dermaku *et al.*, 2008] Artan Dermaku, Tobias Ganzow, Georg Gottlob, Benjamin J. McMahan, Nysret Musliu, and Marko Samer. Heuristic Methods for Hypertree Decomposition. In *MICAI*, volume 5317 of *LNCS*, pages 1–11. Springer, 2008.

[Downey and Fellows, 1999] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[Halin, 1976] Rudolf Halin. S-functions for graphs. *J. Geometry*, 8:171–186, 1976.

[Hall *et al.*, 2009] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[Hammerl *et al.*, 2015] Thomas Hammerl, Nysret Musliu, and Werner Schafhauser. Metaheuristic Algorithms and Tree Decomposition. In *Handbook of Computational Intelligence*. Springer, 2015.

[Hintikka, 1973] Jaakko Hintikka. *Logic, Language-games and Information: Kantian Themes In the Philosophy of Logic*. Oxford: Clarendon Press, 1973.

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

[Jégou and Terrioux, 2014] Philippe Jégou and Cyril Terrioux. Bag-Connected Tree-Width: A New Parameter for Graph Decomposition. In *ISAIM*, pages 12–28, 2014.

[Kloks, 1994] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994.

[Kneis *et al.*, 2011] Joachim Kneis, Alexander Langer, and Peter Rossmanith. Courcelle's Theorem - A Game-Theoretic Approach. *Discrete Optimization*, 8(4):568–594, 2011.

[Koster *et al.*, 1999] Arie M. C. A. Koster, Stan P. M. van Hoesel, and Antoon W. J. Kolen. Solving Frequency Assignment Problems via Tree-Decomposition 1. *Electronic Notes in Discrete Mathematics*, 3:102–105, 1999.

[Lauritzen and Spiegelhalter, 1988] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.

[Leyton-Brown *et al.*, 2009] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical Hardness Models: Methodology and a Case Study on Combinatorial Auctions. *J. ACM*, 56(4):1–52, 2009.

[Morak *et al.*, 2012] Michael Morak, Nysret Musliu, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Evaluating Tree-Decomposition Based Algorithms for Answer Set Programming. In *LION*, volume 7219 of *LNCS*, pages 130–144. Springer, 2012.

[Musliu and Schwengerer, 2013] Nysret Musliu and Martin Schwengerer. Algorithm Selection for the Graph Coloring Problem. In *LION*, volume 7997 of *LNCS*, pages 389–403. Springer, 2013.

[Niedermeier, 2006] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics And Its Applications. Oxford University Press, 2006.

[Robertson and Seymour, 1984] Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[Smith-Miles, 2008] Kate Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1):6:1–6:25, 2008.

[Xu *et al.*, 2005] Jinbo Xu, Feng Jiao, and Bonnie Berger. A tree-decomposition approach to protein structure prediction. In *CSB 2005*, pages 247–256, 2005.

[Xu *et al.*, 2008] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *J. AI Research*, 32:565–606, 2008.