Optimal decision tree using dynamic programming

for the algorithm selection problem

Henwei Zeng[1]
Supervisor(s): Emir Demirović[1], Koos van der Linden[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 29, 2023

Name of the student: Henwei Zeng
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Koos van der Linden, Frans Oliehoek

Abstract

Several algorithms can often be used to solve a complex problem, such as the SAT problem or the graph coloring problem. Those algorithms differ in terms of speed based on the size or other features of the problem. Some algorithms perform much faster on a small size while others perform noticeably better on a larger instance. The optimization problem in this case is to select the best-performing algorithm based on the problem features, resulting in a much faster overall runtime. This is defined as the algorithm selection problem. Many different approaches have been used to solve this problem, such as constructing optimal decision trees. However, there is little published data on using optimal decision trees for algorithm selection and one study reveals a problem in finding a feasible solution on a large number of problem instances. We provide new insights into solving algorithm selection using a dynamic programming approach. The motivation to use this novel approach is that recent studies suggest it has lower scalability issues compared to the traditional optimal decision tree algorithms, due to several efficient techniques such as caching and frequency counting method. The investigation has shown that compared to the integer programming method, the dynamic programming approach is significantly faster and is able to solve large problem instances.

1 Introduction

Solving computationally challenging problems, such as optimization problems or search problems, optimally is a major area of interest within computer science. Many of those complex problems have different algorithms that are able to calculate the solution. Generally, none of these algorithms can perform optimally across various problem instances [1]. This makes it interesting to search for an algorithm that has the best-expected performance according to the particular instance. Rice [2] defined this as the algorithm selection problem (ASP).

The ASP tries to formulate complex problems into abstract models with features and selects the optimal algorithm based on these features. For example, you can take different routes to navigate to your destination, various features can be fuel usage, maximum speed, and how the roads are connected. Several algorithms are able to calculate to get the optimal route, but some algorithms reach the solution faster because they take the complexity of connected roads less into account. So for "simpler" roads, those are most optimal while other algorithms are likely to perform better when the complexity of the problem rises.

One way to better understand the algorithmic problems is to construct a decision tree model. The decision tree is commonly used for classification problems, the model constructs, on a given dataset, a flow-like chart that classifies the solution based on decision splits, see for example in Figure 1. The benefit of the decision tree model is that it allows for a better comprehension of the given data while maintaining a clear structure. Decision trees can be constructed in various ways, such as using a heuristic approach [3, 4, 5] or optimal decision tree approach [6, 7]. The optimal decision tree is the best possible tree based on

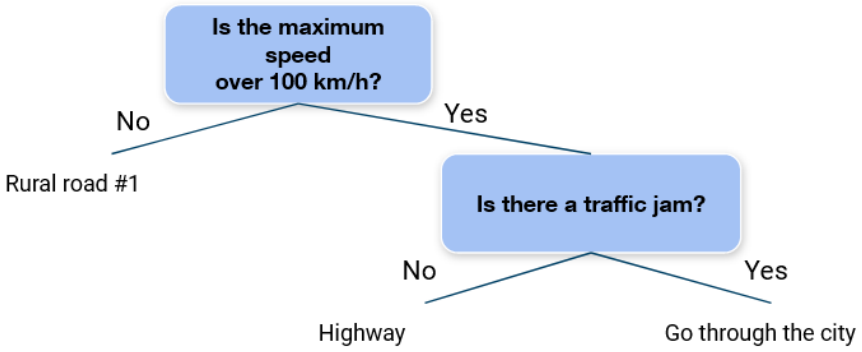


Figure 1: Example decision tree for the fastest route to reach your destination

the given data, but constructing an optimal decision tree has been proven to be an NP-hard problem [8], but it allows for a better representation of the data compared to the usual heuristic approaches.

However, there has been little quantitative analysis using different approaches to constructing the optimal decision tree to tackle the ASP. One recent study used Mixed Integer Programming (MIP) to generate optimal decision trees [7]. The main weakness of this approach is due to its scalability of the MIP algorithm. The solution for the ASP is limited to a maximum of around 500 problem instances while having significantly lower performance when the problem reaches over 200 instances. The algorithm falls back on decision tree heuristics when it exceeds the maximum capacity of the solution.

This research sets out to investigate the possibility of generating optimal decision trees on the algorithm selection problem using a dynamic programming approach. Dynamic programming (DP) is proven to be an efficient and fast way to generate decision trees, due to breaking the tree into smaller subtrees and adding caching (memoization) to speed up the process [6]. Furthermore, another objective is to investigate whether DP is more scalable compared to the MIP approach [7]. In this research, the MurTree [6] algorithm is extended to construct optimal decision trees that map the problem instance to their best-performing algorithms. Solving this problem will provide more insight into the relationship between algorithmic problems, their features and their solutions.

The rest of this paper has been divided into five sections. Section 2 explains the notations and the definitions of key aspects of this paper. This is followed by Section 3 giving a brief overview of the related work in this field that has been done in the past. Section 4 explains the techniques that are used for this study. Section 5 analysis the result of the experiments done based on the techniques. The remaining part discusses the conclusion, future work and responsible research of this study.

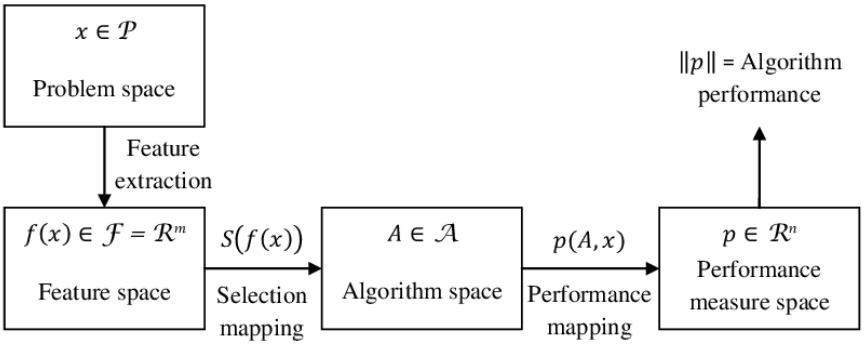


Figure 2: The algorithm selection problem[9]

2 Preliminaries

This section provides some explanation of the methods and literature used to understand the latter part of the paper. The vital part of the algorithm selection problem (ASP) is covered. Furthermore, this research will extend the current MurTree [6] algorithm, it is sufficient to know the high-level implementation of the optimal decision tree algorithm. Some parts of this algorithm are changed to generate the optimal decision tree for the ASP more accurately.

2.1 Algorithm Selection Problem

The ASP is a problem of selecting an effective or optimal algorithm based on performance such as runtime on a given situation, as proposed by Rice [2]. The motivation to solve the ASP using the optimal decision tree model is to provide more insight into and comprehension of the problem space. The feature in the decision node of the tree may provide valuable information about the problem instance. The importance of the feature increases considerably the closer it is to the root node. Furthermore, often a "winner-take-all" is used, where we select the algorithm that has the best performance on average in all cases, but it is essential to notice that there is no dominating algorithm over all different problem instances [1]. Different algorithms perform better in different circumstances. The most important area of the ASP is depicted in Figure 2.

The problem space $\mathcal{P}$, an extensive collection of diverse sets of problem instances. Each of those problems has a certain number of features, such as the problem size or probing features, which is computed by briefly running an existing algorithm on the problem [10].

The feature space $\mathcal{F}$, a set of features extracted from the problems in $\mathcal{P}$

to describe the problem. Ideally, the set of features should be of a significantly lower amount than the problem space $\mathcal{P}$, since this reduces the accuracy of the algorithm selection.

The algorithm space $\mathcal{A}$, a set consisting of algorithms that can solve instances of problem $\mathcal{P}$

The performance space $p(A, x)$, where A is a solution algorithm and x is an instance of the problem in problem space $\mathcal{P}$. This space denotes the performance of the algorithms, this can be speed, the number of possible solutions for a problem instance, etc.

2.2 MurTree Algorithm

The MurTree is a dynamic programming algorithm for computing optimal classification trees. This is done by exhaustive searching through all possible trees. Different techniques are used to speed up this process, such as exploiting the overlap between the trees and avoiding computing suboptimal decision trees. The solution algorithm of constructing an optimal decision tree is based on and extended from the MurTree algorithm [6] for the ASP. This is presented more in-depth in section 4.2.

2.2.1 High-Level Idea

Eq. 1 provides a high-level summary of the MurTree algorithm, including its dynamic programming formulation. The input consists of $\mathcal{D}$, which is the dataset with features $\mathcal{F}$. The upper bound of the depth is defined as d and n is the upper bound of the number of decision nodes. The MurTree algorithm takes binary dataset as input, which means that the dataset can be split into positive and negative instances $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$. Furthermore, $\mathcal{D}(f)$ is a set of instances from $\mathcal{D}$ that contain the feature f. The output of this algorithm is the minimum number of wrongly classified instances of the given dataset and the optimal decision tree consisting of decision and classification nodes.

$$T(\mathcal{D}, d, n) = \begin{cases} T(\mathcal{D}, d, 2^d - 1) & n > 2^d - 1 \\ T(\mathcal{D}, n, n) & d > n \\ \min\{|\mathcal{D}^+|, |\mathcal{D}^-|\} & n = 0 \vee d = 0 \\ \min\{T(\mathcal{D}(\overline{f}), d-1, n-i-1) & n > 0 \wedge d > 0 \\ \quad + T(\mathcal{D}(f), d-1, i) : f \in \mathcal{F}, i \in [0, n-1]\} & \end{cases} \tag{1}$$

The MurTree algorithm is split into four different cases. The first and second cases in this equation limit the maximum number of decision nodes and depth of the tree to avoid redundant calculations, i.e. depth two decision trees cannot have more than three nodes. Case three returns the minimum of positive or negative instances if the depth or number of nodes is zero because this minimizes

the misclassification while having no decision nodes. The fourth case is the core of the MurTree algorithm. This case computes the minimum misclassification of all feature splits and ways to distribute the left and right children of the root node.

3 Related Work

This section gives a literature review of similar works that have been done in the past. First about methods used to tackle the algorithm selection problem (ASP), followed by a more in-depth review of optimal decision tree studies.

3.1 Methods Related to Solving ASP

A considerable amount of literature has been published on the ASP proposed by Rice [2]. Each of those studies tried to tackle this problem with a different approach.

Lagoudakis et al [11] have used ideas from reinforcement learning to solve the ASP. This study reveals that learning algorithms are also possible methods to construct a solution for selecting algorithms. This encouraged future works to think of possible machine-learning solutions.

Malitsky et al [12] used a k-Nearest-Neighbor approach where they improved the performance of building algorithm portfolios by a significant amount, followed by Kadioglu et al [13] who extended the previous study, by improving the technique using *Distance-Based Weighting and neighborhood size clustering* and boosted the performance by another significant amount by comparing to SATzilla and VBS algorithms.

Musliu and Schwengerer [14] tested the performance of different machine learning classification algorithms, such as Bayesian Networks, C4.5 Decision Trees, k-Nearest-Neighbor, etc, to select solution algorithms to tackle the Graph Coloring problem. One weakness of machine learning is that often it only contains one recommended algorithm per instance. This study introduced a new performance measurement, which is the *success rate*. The success rate is the ratio of the number of instances where the solver achieves the best solution and the total number of instances. This approach yields significantly better performance compared to any previously used heuristic. However, this study has not dealt with the importance of the features of the algorithm, which is also a critical aspect of the ASP.

In another recent study by Polyakovskiy et al [15] where they built a substantial benchmark suite to tackle the travelling salesman problem and constructed a decision tree to select the most optimal algorithm. The drawback of this case is that it does not generalize the problem, meaning the performance will not be as high on an unknown set compared to the test set. Furthermore, the decision tree is constructed using heuristics and an optimal decision tree allows for more accurate solutions.

Similarly, Vilas et al [7] constructed optimal decision trees using the MIP approach. This allows for a more accurate selection of algorithms and surprisingly enough the results indicate that the decision tree does not overfit test data. However, this approach leads to sub-optimal scalability of the algorithm. Where the algorithm cannot compute the best upper- and lower bound within a time limit of an hour on a dataset that has over 500 problem instances. This is an important motivation for our research because it shows that the optimal decision tree approach is an efficient and trustworthy approach to tackle the ASP.

3.2 Optimal Decision Trees

The MIP algorithm is one of the many techniques to construct optimal decision trees. Bertsimas and Shioda [16] were one of the first to propose using MIP approach for optimal decision trees. This has been further improved by Bertsimas and Dune [17], where they used a more modern mixed-integer optimization (MIO), followed by a research of Verwer and Zhang [18]. Verwer and Zhang used an efficient encoding of decision tree learning in integer programming (DTIP). This research showed improved performance compared to the existing solutions of greedy heuristics. However, the tests showed that the algorithm is limited to a depth of 5 and a data size of 1000 instances.

In recent years, more works on optimal decision trees were followed, such as using binary linear program formulations [19] by Verwer and Zhang, where they increase the performance by efficiently encoding the problem into binary data.

Hu et al [20] introduced the first practical algorithm for generating optimal decision trees with binary datasets. It shows that this method can further increase the performance and scalability of optimal decision tree algorithms by testing against the previous BINOCT [19] and CART [4] algorithms. However, the experiments were only performed on small datasets, the goal of future works is to extend to much larger datasets.

Aglin et al [21] introduced DL8.5 which increases the performance by several orders of magnitude. The algorithm used the idea of using a cache of itemsets combined with a branch-and-bound search.

This finally leads to the work of Demirović et al [6] where they proposed an optimal decision tree algorithm using a dynamic programming and search approach. This research showed that using this approach leads to a significant increase in speed compared to the current state-of-the-art. In our research, we make use of this algorithm and extend it to select the optimal algorithm based on the features of the problem.

4 Methodology

This section goes into the details of the methods used for our experiment. Firstly we need to obtain the features from our problem for the algorithm selection problem. As those features are key to constructing the optimal decision tree and understanding the problem space. Furthermore, all algorithms should be

able to solve the problem, therefore the misclassification calculations need to be adjusted to the performance of the algorithm given the features of the problem instance. A single classification to a problem instance might result in a lower accuracy as some algorithms can perform very closely to each other while others might have a significantly low performance on a certain set of features.

4.1 Feature Selection

For the algorithm selection problem, it is important to extract the features from the problem space $\mathcal{P}$ first. Since the goal is to select the optimal algorithm based on the features of the problem instance. Instance features are calculated and extracted from two different problems, which are Maximum Satisfiability (MAXSAT) and the Graph Coloring Problem (GCP).

Before we can compute optimal decision trees and execute the experiments, sufficient data and their features are gathered. A diverse set of problem instances are collected from the MaxSAT competition [22] and graph coloring problem [23]. For each problem instance, their features are calculated and extracted.

4.1.1 Features for Maximum Satisfiability (MaxSAT)

Our first problem instance is based on the Maximum Satisfiability problem (MAXSAT). This is an extension of the boolean satisfiability problem (SAT). The MaxSAT is formulated into a set of clauses in CNF. The goal of the solution algorithm is to satisfy as many of those clauses as possible. From this problem, 44 features are extracted and categorized from the SAT problem instances as similarly done in the study of Nudelman et al and Hutter et al [24, 10]. The features are based on the problem size, Variable and Clause graphs statistics, balancing features and proximity to the Horn formula. Some computationally expensive features from those papers are emitted due to time constraints of generating the dataset. Table 1 summarizes the features of the MaxSAT problem for our algorithm and separates the features into six different groups.

- **Problem size features**, this group denotes the size of the problem, measured by the number of variables, clauses and their ratios.
- **Variable-Clause Graph features**, this group generates the features based on a graph representation of the SAT instance. The graph is generated by having a node for each variable and clause. The nodes have an edge between the variable node and the clause node Whenever the variable occurs in that clause.
- **Variable Graph features**, this group generates a variable-based graph representation of the SAT instance. This graph has a node for each variable and an edge between the nodes whenever the variables both occur in a clause.
- **Clause Graph features**, this group generates a clause-based graph representation of the SAT instance. This graph has a node for each clause

Table 1: SAT features

Number	Feature	Description
	Problem size Features	
1	Number of Clauses	c
2	Number of variables	v
3-5	Ratio	c/v, $(c/v)^2, (c/v)^3$
6-8	Ratio Reciprocal	v/c, $(v/c)^2, (v/c)^3$
	Variable-Clause Graph Features	
9-13	Variable nodes degree statistics	mean, variation coefficient, min, max and entropy
14-18	Clause nodes degree statistics	mean, variation coefficient, min, max and entropy
	Variable Graph Features	
19-22	Nodes degree statistics	mean, variation coefficient, min, max
	Clause Graph Features	
23-27	Nodes degree statistics	mean, variation coefficient, min, max and entropy
	Balance Features	
28-31	Ratio literals in each clause	mean, variation coefficient, min, max
32-35	Ratio occurrence variables	mean, variation coefficient, min, max
36-38	Fraction unary, binary, ternary clauses	
	Proximity to Horn Formula	
39	Fraction of Horn Clauses	
40-44	Number of occurrences in a Horn Clause for each variable	mean, variation coefficient, min, max

and an edge between the nodes whenever both clauses have the same negated literal.

- **Balance features**, this group measures the balance of positive and negative occurrences of the SAT instance. The ratios of positive and negative literals within a clause and the ratio of positive and negative occurrences of each variable are calculated.
- **Proximity to Horn Formula**, the final group measures the proximity of the SAT instance to the Horn Formula. A clause is defined as a Horn clause when it has at most one positive literal.

4.1.2 Features for Graph Coloring Problem

The Graph Coloring Problem is a problem where considering a graph, the goal is to minimize the number of different colors to color all nodes in such a way that no nodes sharing an edge have the same color. 48 features for the graph coloring problem are identified based on the study from Musliu and Schewengerer [14]. The features can be categorized into the following five different groups depicted in table 2.

- **Graph size features**, this group denotes the size of the graph, measured by the total number of nodes and edges.
- **Node degree**, this group measures the degree of the nodes. Minimum, maximum, mean, median, quartiles, variation coefficient and entropy are calculated.

Table 2: Graph coloring features

Number	Feature	Description
	Graph size Features	
1	Number of nodes	n
2	Number of edges	m
3-4	Ratio	n/m, m/n
5	Density	(2*m)/(n*(n-1))
	Node degree	
6-13	Node degree statistics	min, max, mean, median, $Q_{0.25}, Q_{0.75}, vc, entropy$
	Maximal clique	
14-20	Maximal clique statistics	min, max, median, $Q_{0.25}, Q_{0.75}, vc, entropy$
21	Computation time	
22	Maximum cardinality	
	Clustering Coefficient	
23-29	Local clustering coefficient	min, max, median, $Q_{0.25}, Q_{0.75}, vc, entropy$
30-36	Weighted local clustering coefficient	min, max, median, $Q_{0.25}, Q_{0.75}, vc, entropy$
37	Computation time	
	DSATUR greedy coloring	
38	Number of colors needed	
39	Computation time	
40-47	DSATUR statistics	min, max, mean, median, $Q_{0.25}, Q_{0.75}, vc, entropy$
	Computation time	
48	Computation time	

- **Maximal clique**, this group used a simple greedy algorithm to calculate the maximal clique of each node.
- **Clustering Coefficient**, this group measures the clustering coefficient. Both their classical definition [25] and their weighted values, where the node coefficient is multiplied by their degree.
- **DSATUR greedy coloring**, this group calculates its statistics by using the DSATUR algorithm [26].
- **Computation time**, the final feature is measured by the average time it takes to generate the above features.

4.1.3 Binarization of the Features

Considering the Murtree algorithm, currently, the method only accepts binary datasets and the features of the problems are in continuous values. We use the sklearn library[2] in Python to binarize our dataset. The library uses a k-means algorithm to generate threshold values for the features and separates the continuous values into two bins (1 or 0) based on the threshold value. This does result in a loss of information from our dataset, which likely results in an optimal decision tree with lower accuracy.

[2]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.KBinsDiscretizer.html

4.2 Cost Function

The current MurTree algorithm calculates the misclassification score based on the leaf misclassification of the subtree. Each problem instance has one label, which is the optimal solution for that problem instance. For our solution algorithm, the misclassification calculation is slightly adjusted to consider all available algorithms on each problem instance. We prioritize the algorithms based on their performance on the current problem instance. In this case, the runtime of the algorithm on that particular problem instance. The reason for this strategy is that some algorithm can perform close to each other, which mean they will be penalized less if we calculate the cost based on the performance. In instances where they are slow or unable to find a solution compared to other algorithms, they will be penalized much more compared to the one instance, one algorithm strategy.

Let $C(\mathcal{D}, a_i)$ denote the total cost of the dataset D and the algorithm a_i. The total cost of algorithm a_i can be calculated with $C(\mathcal{D}, a_i) = \sum_{c \in \mathcal{D}} p(c, a_i)$ where $p(c, a_i)$ is the performance score of the algorithm. Higher cost results in a more sub-optimal algorithm. With this cost function, the goal is to minimize the cost of each classification node.

5 Experimental Setup and Results

The goal of this section is to evaluate the performance of the optimal decision tree construction algorithm and to compare it against the current state-of-the-art. The performance is analysed based on 3 different datasets, each with a different number of problem instances, features and solutions algorithms.

5.1 Dataset and Computational Environment

Datasets are collected from two different studies and one publicly available benchmark dataset. The features of the datasets are calculated and binarized based on the method in Section 4.1.

Graph Coloring dataset is collected from the variable ordering study [23]. 432 graphs are generated from Culberson's random instance graph generator and another set of 137 graphs is collected from Dimacs Challenge [27]. After calculating the features and removing instances where no algorithm can solve, only 105 instances, with 4 solution algorithms remained.

For the MAXSAT dataset, we took the unweighted instances of the 2020 MaxSAT competition[1]. After generating the features, the number of problem instances that are collected is 290 instances with 15 different solution algorithms.

Finally, the last dataset consisting of 1004 problem instances, 37 features and 532 different configurations of solution algorithms is collected from the study of Vilas et al [7].

[1] https://maxsat-evaluations.github.io/2020/benchmarks.html

The experiments were run on a Java implementation of the algorithm and on an AMD Ryzen 7 5800H CPU with 16 GB of RAM on a single processor and a single algorithm at a time. The experiments were also run on a time limit of five minutes. Furthermore, the performance of the algorithm is measured as the runtime in ms.

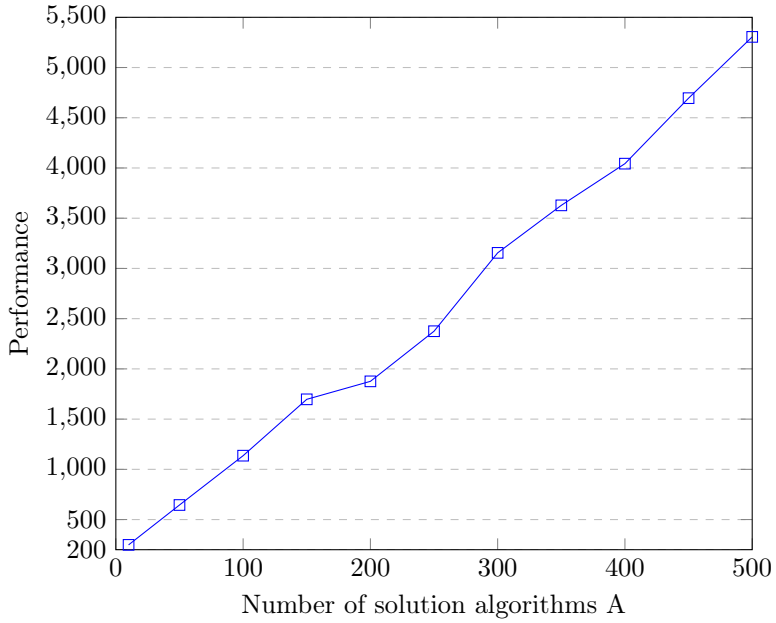
5.2 Experiments to evaluate the Performance and Scalability

In the following texts, some experiments are done on the performance of the number of algorithms and a comparison in performance is made on the MIP algorithm of Vilas et al [7] and this algorithm.

5.2.1 Experiments on the Algorithm Space $\mathcal{A}$

One of the main differences between this algorithm and the MurTree algorithm [6] is that for each instance, we take all available solution algorithms into account whereas the MurTree algorithm only selects the optimal one. For this experiment, we use the dataset of Vilas et al [7]. Where we use 200 problem instances and test on different algorithm sizes with a maximum tree depth of four.

Figure 3: Performance based on number of algorithms



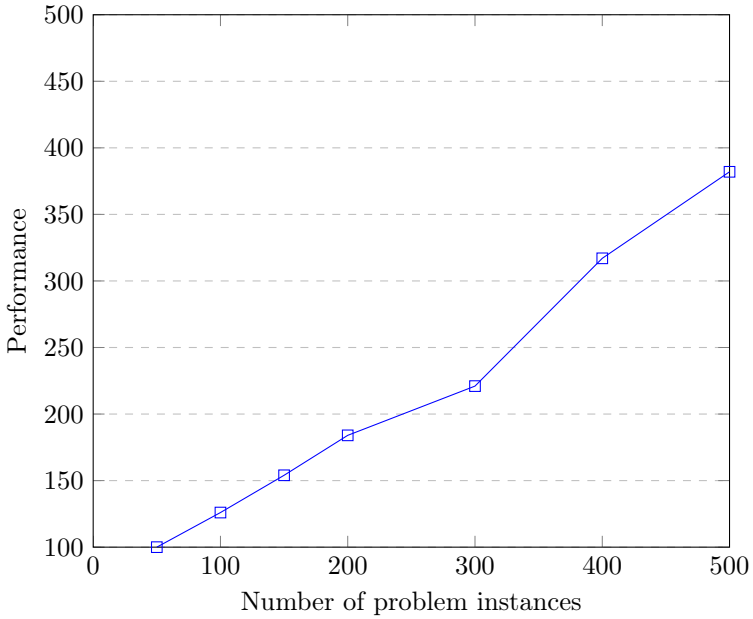
Based on these results, we can conclude that the increase in runtime is linear. The small difference that happens between 250 and 300 is due that the optimal tree construction algorithm found a slightly larger optimal decision tree compared to the results below 250 solution algorithms. But the overall difference is not significant.

5.2.2 Comparison against MIP Implementation [7]

The aim of this research is to evaluate whether the Dynamic Programming approach has better performance and scalability compared to the current MIP implementation. The MIP approaches compute with a maximum tree depth of three and a base of 50 solution algorithms. It is tested on a dataset size ranging from 50 to 500 problem instances.

Unfortunately, there were no data on the computation speed of the MIP approach. However, it is reported to have performance issues where the dataset is over 200 problem instances and no feasible solution is found when the size is over 500 instances within a time limit of one hour.

Figure 4: Performance on MIP dataset



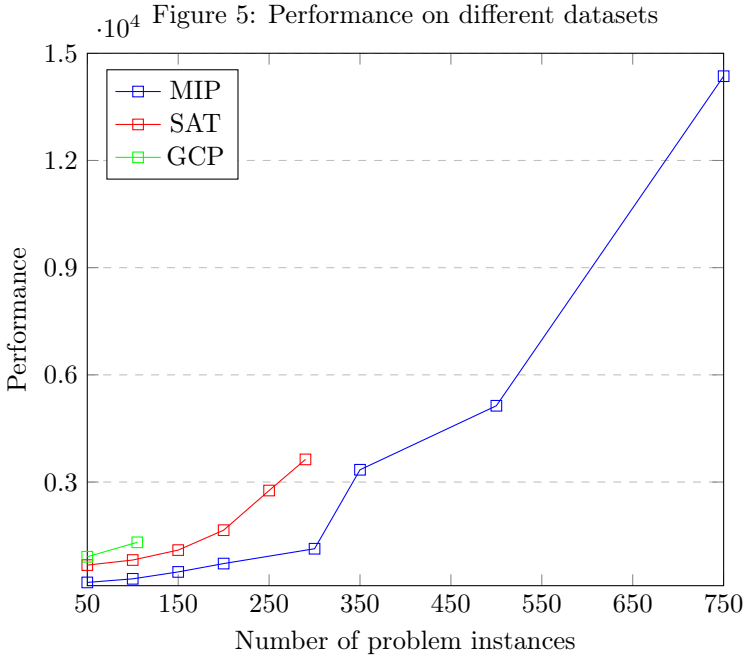
Based on the results, we can indicate that the algorithm is still able to construct a tree within a second compared to the MIP implementation, even in large instances. The results also show that the runtime increases linearly based on the number of instances. As mentioned in the study of Demirović et al [6] the most significant decrease in performance is due to the depth of the tree. The dynamic formulation computes the tree on depth three or lower within seconds but increases exponentially on larger trees.

Furthermore compared to the MIP implementation, the dataset is already preprocessed with binarization. Which removes the calculations of selecting the optimal split value for the feature. This is a trade-off where the algorithm will be sped up by a significant amount but reduces the accuracy of the optimal decision tree.

5.2.3 Performance on Different Datasets

For this experiment, the datasets of all 3 different problems are considered. The performance of the algorithm is calculated based on the size of the datasets and compared to each other. For the MIP dataset, we only take 50 algorithms into account instead of the full 532 solution algorithms size since this impacts the performance dramatically. Furthermore, for this experiment, the maximum tree depth is set to four.

Figure 5: Performance on different datasets



The results indicate that the MIP dataset is still the fastest to compute while having a larger algorithm size $\mathcal{A}$ compared to the other datasets. The other difference between the datasets is the feature size. GCP has a feature size of 48, SAT has 44 and MIP has 37. This indicates that the feature size is more impactful to the performance compared to the algorithm size. This makes sense since for each split, all features are considered and for each of those features, the total cost needs to be computed. However, a limitation of this experiment is that the two datasets are relatively small, especially the GCP dataset which only consists of 105 instances.

6 Conclusion and Future Work

This research showed that a dynamic programming formulation of constructing optimal decision trees for algorithm selection problem is a fast approach compared to the other state-of-the-art algorithm. This approach showed a significant speed-up over different datasets, where, besides the depth of the tree, the number of

features has the biggest impact on the performance. Even on large datasets, the algorithm is able to construct the optimal decision tree within seconds.

However, due to time constraints, possible research in the future is to test the accuracy of the optimal decision tree, compared to the generalisations of the other related works on optimal decision trees. Furthermore, one of those limitations is also due to the small dataset of GCP and SAT problem instances. Another option to be looked into is to consider a dynamic programming algorithm where continuous feature values can be taken into the calculations since the majority of the features are in continuous values.

7 Responsible Research

This research is done according to the FAIR principles [28]. It is vital to make the used dataset and methods publicly available if possible as well to combat the current reproducibility crisis [29]. The Murtree algorithm [6] has an MIT license and allows for redistributing and modifying the code. Furthermore, the source code is openly available from the repository[3].

For the first principle "**Findable**", the datasets that are used are collected from the annual MaxSAT competition [22] and the graph coloring problem [23]. Furthermore, the methods that are used to modify the dataset can be found in the repository and in the Methodology section (4) of this paper.

"Accessible" principle: The data are written in a txt document, no authentication or authorisation is needed to access the data.

"Interoperable" principle: The data and the infrastructure (code) are interoperable. Other datasets can also be used, however, it needs to be converted to the correct notation first. The algorithms used to convert it into the correct notation can also be found in the repository.

"Reusable" principle: The data and the source code can be reused and are provided with descriptions for better understanding and allow for replication or modification.

8 Acknowledgements

We would like to acknowledge and thank our supervisor and responsible professor for their valuable feedback and guiding this project: Koos van der Linden and Emir Demirović. Furthermore, the course and teaching staff for setting up this project and providing us the opportunity to learn important concepts and academic skills. And finally, our group members for their peer feedback and insights on this research topic: Sven Butzelaar and Valentijn Götz.

[3] https://github.com/Henweiz/cse3000

References

[1] David H Wolpert and William G Macready. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[2] John R Rice. "The algorithm selection problem". In: *Advances in computers*. Vol. 15. Elsevier, 1976, pp. 65–118.

[3] Sreerama K Murthy and Steven Salzberg. "Decision Tree Induction: How Effective Is the Greedy Heuristic?" In: *KDD*. 1995, pp. 222–227.

[4] L Breiman et al. "Cart". In: *Classification and Regression Trees* (1984).

[5] Steven L Salzberg. *C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993*. 1994.

[6] Emir Demirović et al. "MurTree: Optimal Decision Trees via Dynamic Programming and Search". In: *Journal of Machine Learning Research* 23.26 (2022), pp. 1–47.

[7] Matheus Guedes Vilas Boas et al. "Optimal decision trees for the algorithm selection problem: integer programming based approaches". In: *International Transactions in Operational Research* 28.5 (2021), pp. 2759–2781.

[8] Hyafil Laurent and Ronald L Rivest. "Constructing optimal binary decision trees is NP-complete". In: *Information processing letters* 5.1 (1976), pp. 15–17.

[9] Litan Ilany and Ya'akov Gal. "Algorithm selection in bilateral negotiation". In: *Autonomous Agents and Multi-Agent Systems* 30 (July 2015). DOI: 10.1007/s10458-015-9302-8.

[10] Frank Hutter et al. "Algorithm runtime prediction: Methods evaluation". In: *Artificial Intelligence* 206 (2014), pp. 79–111. ISSN: 0004-3702.

[11] Michail G Lagoudakis, Michael L Littman, et al. "Algorithm Selection using Reinforcement Learning." In: *ICML*. 2000, pp. 511–518.

[12] Yuri Malitsky et al. "Non-model-based algorithm portfolios for SAT". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2011, pp. 369–370.

[13] Serdar Kadioglu et al. "Algorithm selection and scheduling". In: *International conference on principles and practice of constraint programming*. Springer. 2011, pp. 454–469.

[14] Nysret Musliu and Martin Schwengerer. "Algorithm selection for the graph coloring problem". In: *International conference on learning and intelligent optimization*. Springer. 2013, pp. 389–403.

[15] Sergey Polyakovskiy et al. "A comprehensive benchmark set and heuristics for the traveling thief problem". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 2014, pp. 477–484.

[16] Dimitris Bertsimas and Romy Shioda. "Classification and regression via integer optimization". In: *Operations research* 55.2 (2007), pp. 252–271.

[17] Dimitris Bertsimas and Jack Dunn. "Optimal classification trees". In: *Machine Learning* 106.7 (2017), pp. 1039–1082.

[18] Sicco Verwer and Yingqian Zhang. "Learning decision trees with flexible constraints and objectives using integer optimization". In: *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer. 2017, pp. 94–103.

[19] Sicco Verwer and Yingqian Zhang. "Learning optimal classification trees using a binary linear program formulation". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 1625–1632.

[20] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. "Optimal sparse decision trees". In: *Advances in Neural Information Processing Systems* 32 (2019).

[21] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. "Learning optimal decision trees using caching branch-and-bound search". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3146–3153.

[22] Fahiem Bacchus et al. "MaxSAT Evaluation 2020: Solver and Benchmark Descriptions". In: (2020).

[23] Anthony Karahalios and Willem-Jan van Hoeve. "Variable ordering for decision diagrams: A portfolio approach". In: *Constraints* 27.1 (2022), pp. 116–133.

[24] Eugene Nudelman et al. "Understanding random SAT: Beyond the clauses-to-variables ratio". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 438–452.

[25] Duncan J Watts and Steven H Strogatz. "Collective dynamics of 'small-world'networks". In: *nature* 393.6684 (1998), pp. 440–442.

[26] Daniel Brélaz. "New methods to color the vertices of a graph". In: *Communications of the ACM* 22.4 (1979), pp. 251–256.

[27] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*. Vol. 26. American Mathematical Soc., 1996.

[28] Mark D Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific data* 3.1 (2016), pp. 1–9.

[29] Monya Baker. "Reproducibility crisis". In: *Nature* 533.26 (2016), pp. 353–66.