Using Features Of Models To Improve State Space Exploration

Author
A.R. Heijblom
a.r.heijblom@alumnus.utwente.nl

Supervisors
Prof. Dr. J.C. van de Pol
Dr. Ir. M. van Keulen
J.J.G. Meijer MSc

Master Thesis
Submitted December 14, 2016

Faculty of Electrical Engineering,
Mathematics and Computer Science

University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

State space methods are a popular approach to perform formal verification. However, these methods suffer from the state space explosion problem. In the past decades many methods did arise to cope with the state spaces of larger models. As result, a user has many different strategies in which state space methods can be applied on new models.

Due to the wide variety of strategies and models is may be hard for a user to select an appropriate strategy. If a bad strategy is selected, the given model can be unsolvable or the process may waste resources like time and memory. Moreover, the intervention of the user makes state space methods less automated. Therefore, it would be convenient if model checking tools itself determine the strategy for a given model. In this way, model checking tools can determine the most suitable strategy for a given model such that the available resources are optimally utilized.

This process requires model checking tools to predict a strategy based on the information presented in a given model. Our research investigates to what extent characteristics of a model can be used to predict an appropriate strategy. The performance of 784 different PNML and DVE models was determined using LTSmin for 60 selected strategies. This information was used to create several classifiers using machine learning techniques. The classifiers should predict an appropriate strategy given eleven selected features of a model.

The performance data of the models show that each strategy has some set of models it is not appropriate for. None of the strategies was able to solve all models. Hence, for a given set of models a dynamic selection of the strategy is recommended. Unfortunately, the classifiers did not outperform all of the strategies. But each of the examined features did contribute in providing useful information for predicting an appropriate strategy.

Preface

The past year I've worked on a graduation project of the Master Computer Science at the University of Twente. The results of this project are discussed in this thesis. During the project, I learned a lot about the techniques I used, which were mostly new to me. I discovered how simple tools like BASH could make tasks much easier than performing it manually. Due to the massive amount of data, programming and automating tasks were necessary and I definitely enjoyed writing the many programs and scripts needed to collect and analyze the data.

I would like to thank Jeroen for guiding me during the project. During the first weeks, he taught me many things about LTSMIN and provided the necessary tools to start my project. He was always there for me in providing information I needed to advance my project.

I would also like to thank my supervisors for the guidance during my project. The meetings were valuable to me to shape my project and their feedback was very useful to improve and advance my project.

Last but not least, I would like to thank my family and friends. In the past months, they were always there for me. They helped to keep me motivated and supported me during the project.

December 2016, Enschede

Richard Heijblom

Contents

1 Introduction

This chapter gives an introduction to the research described in this thesis. Section 1.1 sketches the context in which the research is performed and provides motivation for this research. Section 1.2 formulated the problems this research aims to solve and a possible solution. Section 1.2 further describes how the solution is accomplished by listing the goals of this research. Section 1.3 describes how the research is performed in order to achieve the stated goals. This chapter concludes with section 1.4 which gives an overview of the structure of this report.

1.1 Background

During creation of programs and software is almost impossible to create a fault free product. As result, the product may behave worse than intended due to the presence of programming errors. An obvious way to increase the quality of a software product is to discover and fix programming errors. Testing is a popular method to discover errors. However, testing can be time consuming and is limited to indicating errors and cannot guarantee the absence of errors. More advanced techniques are needed when one wants to guarantee the presence or absence of certain properties. Formal verification offers the methods and tools to guarantee properties of programs.

Formal verification is the field where one proves or disproves whether a program satisfies some specified formal behavior. This is a powerful method to indicate that a program satisfies certain desirable properties and does not have certain undesirable properties. When a property cannot be proved, this may indicate that the program is lacking some desirable behavior. That information can be used to improve the program, leading to higher quality software.

Formal verification is not limited to evaluation of software. It is also a powerful tool to verify hardware, specifications of protocols or designs. The program, hardware, protocol or design may be too complex to verify directly. It is common that a specification is made, capturing the most important behavior of the object to be verified. The specification of the object where formal verification is applied to is called the model.

1.1.1 Verification Methods

In general, there are two major methods to perform formal verification: theorem proving and state space methods [41]. In theorem proving, one formulates a mathematical theorem about the specification of the object under verification and attempts to prove that theorem. Theorem proving can be done either manually or with the help of a theorem prover tool. Although theorem proving is generic, there are two major drawbacks. Firstly, it is common that a lot of human intervention is needed. Theorem proving can only be done by specifically trained personnel and the human intervention makes theorem proving time consuming. Secondly, theorem proving is not suitable for analyzing the behavior of a model. If a certain theorem cannot be established, then the cause may be unclear. This makes theorem proving inappropriate for identifying the nature

and location of errors and fixing incorrect models.

The second formal verification method offers a solution for the two problems with theorem proving. State space methods analyze models by constructing the state space of the behavior of a model. The state space basically informs which states are possible in a given model. The questions for the object under verification are answered using the state space. Globally, the state space methods can be grouped in two categories: explicit methods and symbolic methods. The methods differ in how the states are stored during the exploration of a model. Explicit methods allocate a fixed amount of memory per state. Symbolic methods use binary decision diagrams (BDD's) or a variant of BDD's to store the states.

The construction and analysis of a state space can mostly be done automatically. Hence, state space methods can be applied by less trained personnel and is less time consuming than theorem proving. State space tools are better in providing the location of errors, because they are based on the behavior. There is, however, a major drawback of the state space methods: the state space explosion problem [41]. Informally, this means that when the size of a model grows linearly, the size of its state space grows exponentially. As result, the state space of most models are too huge to analyze.

1.1.2 State Space Methods In Practice

Despite the state space explosion problem, state space methods are still useful in practice. Many measures have been developed in order to cope with large state spaces, including partial order reduction, abstraction and limiting to specific verification questions [8, 41]. On the other hand, much research is performed on techniques to traverse the state space efficiently. A wide range of tools exist which implement one or multiple techniques to perform state space analysis. As result, a user has many options to apply state space methods on his or her models. In this thesis we will call such a option a strategy. In the most abstract form, a strategy describes how a state space method is applied on a model. Practically, a strategy could be a setting of tool in combination with the algorithms selected of that tool.

This wide variety of strategies raises two problems. An advantage of state space methods is that the tools are mostly automated, which can be applied by less trained personnel. Because of all the options, users has to learn more about the methods and tools, before they can apply them to their models. Moreover, the variety of models is huge. This makes it almost impossible to learn good strategies beforehand. By experience, or by trial and error, one can learn good strategies. This requires more specifically trained personnel for formal verification.

The second problem is closely correlated to the first problem. Due to the state explosion problem, one want to explore the state space as efficient as possible with the available resources, like time and memory. This allows larger models to be verified. When a user should select a strategy, it may pick a bad strategy. This may either lead to a waste of resources or that the model in question cannot

be verified. This is not a desirable property.

Ideally, the user should not be bothered with the details of the strategy when a state space method is applied. In the best case, the user only has to select the model and the verification question to be solved. Any tool implementing state space methods should determine how to solve the verification question efficiently by itself, without user intervention. As result, the available resources can be utilized optimally to verify models, without almost any user intervention.

1.2 Goals

As result of the state space explosion problem, state space methods lacks a form of simplicity due to the many options to tackle large state spaces. This makes state space methods less automated and requires more specifically trained personnel for formal verification. When a bad set of options is selected, state space methods can waste resources or may be unable to solve certain models, which could be verified with another strategy.

Instead of providing the user a large scale of options, it would be convenient to let model checking tools decide which strategy should be applied on the models to be verified. The tools themself could determine a suitable strategy in order to optimally verify a model using the available resources. As result, more and larger models can be verified using less trained personnel. This research investigates whether is it attainable for a model checking tool to determine a suitable strategy based on the properties of a model.

It is mainly unknown which strategies works well in practice. Therefore, we want to investigate how the state space exploration is affected by different strategies. This research investigates whether a fixed strategy should be enforced by a model checking tool or whether a model checking tool should dynamically predict a suitable strategy based on the properties of the given model.

Assuming that there exists no fixed strategy which optimally solves the wide variety of models, it needs to be established whether the model itself provides sufficient information to predict a suitable strategy. This research investigates whether a specified set of properties of a model can predict an appropriate strategy. Furthermore, it is investigated whether each property provides any useful information to predict a suitable strategy.

1.3 Approach

To gain insight in the influence of different strategies on the state space exploration, a large number of tests were executed to measure the performance of different strategies on a large set of models. The LTSmin toolset [16] was used to collect data. LTSmin offers multiple state space exploration tools, multiple options to guide state space exploration and is applicable to multiple different types of models. This data provides insight how well each strategy performs. Hence, using this data it can be established whether a fixed strategy or a more dynamic approach for strategy selection is preferred.

The test data was also used to extract a list with the best strategy per model. This list was used to investigate whether properties of a model can predict an appropriate strategy. If such a prediction is possible, this can be implemented in existing tools to improve state space exploration.

Since the relation between the properties of a model and the best strategy was expected to be nontrivial, machine learning techniques were used to capture the relation between a model and its best strategy. The test data was used to train multiple classifiers. Any of the created classifiers can be used as prediction module within an existing tool.

The classifiers were created using all selected properties of a model. In order to investigate whether each property provides any information, variations of the classifiers were made with different subset of features. These variations provide insight in which properties are relevant to consider and which properties can be ignored for predicting an appropriate strategy.

1.4 Structure Of Thesis

In this chapter, the context of our research is given. Chapter 2 discusses the model checking tool LTSmin used during our research. Chapter 3 gives an introduction to machine learning. In chapter 3, the creation and evaluation of classifiers in general is discussed. Chapter 4 examines existing work related to our research.

Chapter 5 and 6 discuss the research method. The questions on which this research is based are discussed in chapter 5. The method itself is covered in chapter 6 .

Chapters 7, 8 and 9 discuss the results obtained during our research. Chapter 7 discusses the relevant observations obtained during the evaluation of the selected strategies. Chapter 8 evaluates the created classifiers and discusses the performance of the classifiers with respect to metrics and to the performance of the selected strategies. Variations of the classifiers were created by using subsets of features. The results show the relevance of each feature. These results are discussed in chapter 9.

The thesis is concluded in chapter 10. Chapter 10 discusses the most important results obtained during our research and lists possibilities for further research.

2 Preliminaries - LTSmin

This chapter gives an overview of the LTSmin toolset which is used during this research. Section 2.1 gives an overview of the tool. Sections 2.2, 2.3 and 2.4 describe the three layers which can be distinguished in the architecture of LTSmin. Section 2.5 provides a summary of LTSmin.

2.1 Tool Overview

The LTSmin toolset is a high performance model checker [16]. Its modular nature allows to analyze models specified in various languages by various analysis algorithms. This is possible because of the presence of a common interface called PINS. The architecture of LTSmin consists of three layers, where each layer is connected via the PINS interface, see Figure 1. The PINS interface is an implicit state space definition of a model and is used to exchange information between the different modules in LTSmin. It should at least provide the initial state, the partitioned transition function and a labeling function of a model, which together describe a transition system, hence describing a state space. On top of this basic information different extensions are possible [16, 21]. These extensions can be utilized to exchange information about dependencies in the model, which can be exploited to improve model checking. Within our research the dependency matrix of a model, provided by the PINS interface, is used. The dependency matrix defines which transition groups affect which variables in a model.

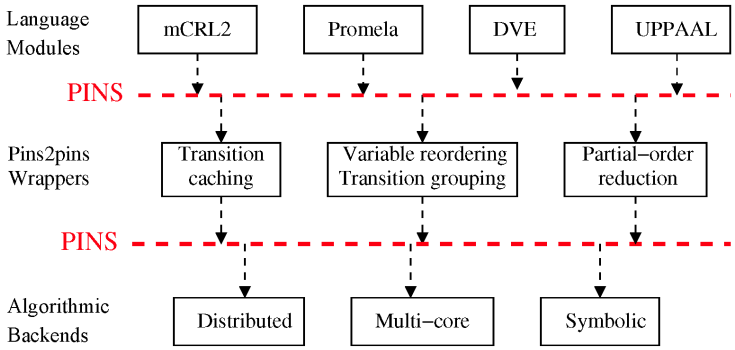

Figure 1: Schematic overview of the architecture of LTSmin.

2.2 Front-end Modules

The front-end modules specify how various languages should be mapped to the PINS interface. This allows users to use the various analysis algorithms of LTSmin not supported in their native tools, without changing the specification language [3]. Currently, LTSmin supports the languages DVE, Promela, mCRL2, ETF, Pbes, Uppaal and Mapa [3, 16, 42]. Recently, a link between ProB and LTSmin was created, allowing the languages B-Method, Event-B, TLA+ and Z notation to be analyzed by LTSmin [2]. LTSmin also supports PNML models, allowing analysis of Petri Nets [22]. Furthermore, one can verify

their own custom model specifications in C by implementing the PINS interface [16].

2.3 Wrapper Modules

The intermediate layer offers various tools to optimize the performance, reduce the state space or verify certain properties. Because they only rely on the model definition by the PINS interface, they can be applied on any model. Currently, it is possible to verify properties specified in LTL and μ-calculus [16]. LTSmin offers partial order reduction [41] for the explicit back-ends and variable reordering for the symbolic back-end. These modules can be enabled to improve state space exploration.

2.4 Back-end Modules

Model checking can be done by storing the states either explicit or symbolically. LTSmin supports both options. Furthermore, the toolset supports verification using multiple cores or distributed systems [16, 18]. Each back-end has its options to specify which algorithm or package it should use and with which configuration. These options allow the user to select specific algorithms or to specify how many resources are used by LTSmin.

2.5 Summary

The LTSmin toolset is a high performance model checker which offer various verification methods for multiple different specification languages. LTSmin has a high modular nature. The modules are connected to each other via a common interface called PINS. Three layers can be distinguished in LTSmin. The front-end layer consists of the language modules which specify how various specification languages are translated to the PINS interface. The intermediate layer consists of wrapper modules which offer various tools to optimize the performance. The back-end layer consists of the various analysis algorithms. These algorithms allow the model to be solved either explicitly or symbolically. Furthermore, it is possible to verify models using multiple cores or distributed systems.

3 Preliminaries - Machine Learning

This chapter gives an overview of machine learning and introduces the metrics used for evaluating the created classifiers in our research. Section 3.1 gives a overview of machine learning and identifies the different subfields within machine learning.

In our research, machine learning techniques are used to predict a strategy given a set of features of a model. More specifically, the goal is to predict which strategy within a finite set of strategies is the most appropriate for a given set of features. The prediction of a strategy is a classification problem. Section 3.2 focuses on the general approach to tackle a classification problem by creating a classifier. Section 3.3 demonstrates three techniques which can be used to train a classifier. Section 3.4, 3.5 and 3.6 discuss metrics to evaluate classifiers.

Section 3.4 defines the basic metrics for classification problem consisting of two different classes. These basic metrics are used in section 3.5 to define metrics for classification problems consisting of three or more classes. Section 3.6 discusses how different types of misclassification can be taken into account by specifying cost or reward per classification of an instance.

3.1 Overview

Machine learning is a subfield in computer science which is concerned with giving a computer the ability to learn. Computers are utilized by giving them a set of instructions to execute in order to perform a task. The set of instructions is usually given by a script or a program, which explicitly describes the steps the computer has to execute. Machine learning deals with giving the computer the ability to learn the steps from data instead of giving the steps explicitly to them. This approach is often utilized for more complex programming tasks where is it infeasible to describe and cover the problem by giving the instructions directly, such as handwriting recognition or image processing. Instead the computer is given an algorithm to learn the relation between the input data and the tasks it has to perform. In the case of handwriting recognition, the input data may be a written text and the task may be to recognize the individual characters of the text.

Within the field of machine learning three subfields can be globally distinguished: supervised, unsupervised and reinforcement learning [11, 15, 23]. These subfields are briefly addressed in sections 3.1.1, 3.1.2 and 3.1.3 respectively.

3.1.1 Supervised Learning

In supervised learning the computer is given a dataset of input and expected output couples. Based on this dataset the computer has to predict output when given a new, unseen input. Within supervised learning two types of problems can be distinguished based on the type of output. If the output is on a continuous domain it is a regression problem. Otherwise it is a classification problem.

Within a classification problem a finite number of classes are defined. The computer is given a dataset of input and class couples. The goal of the computer is to determine the class of new, unseen input. Any program that is able to predict a class based on the input is called a classifier. Actually, any program with the purpose to assign a class to an instance of a classification problem is considered a classifier. When a classifier has to chose between two classes the classifier is a binary classifier. Otherwise, the classifier is called a multiclass classifier.

3.1.2 Unsupervised Learning

Unsupervised learning differs from supervised learning with respect to the provided data to the computer. Instead of providing both input and expected output data, in unsupervised learning only the input data is provided. The tasks of the computer is to discover patterns in the input data [11]. These types of techniques are useful in data mining, where the focus is on discovering relations, but the prediction capabilities of the model are less important.

3.1.3 Reinforcement Learning

The third subfield in machine learning is reinforcement learning. In reinforcement learning the computer has to perform its task in a dynamic environment [15]. The computer has to do actions based on the current state it perceives from the environment. Each action is rewarded with a reinforcement value. The reinforcement value is used by the computer to decide whether its action was good or bad. The goal of the computer is to maximize the reinforcement value over a long period of time. Via trial and error, the computer learns to operate in its environment.

3.2 Classifier Creation

There exists a wide variety of classification problems. In general, one wants to determine the class of a large number of instances. The idea is to create a classifier which is able to determine the class of these instances. As mentioned in section 3.1.1, any program which determines a class for an instance is considered a classifier. Two examples of classifiers are given in section 3.4.3 and 3.5.4. Within machine learning the classifier is created by offering training data, instead of explicitly describing when a given instance belongs to a certain class. After the classifier is trained it is able to predict the classes for new instances.

In general, a classification problem is solved using the following steps [23]:

1. Defining the problem. Firstly, it needs to be established which problem the classifier to be created has to solve. The main goal is to specify the instances of the problem and the corresponding classes.

2. Collecting data. In order to supply training data for the classifier, one has to consider which properties of an instance may be relevant for the classifier. These properties are called features. Furthermore, it has to be decided how these features are represented. In this phase the values of the features of a set of instances are collected together with the class of each instance.

3. Training a classifier. After collecting the data, a training method is selected how the classifier should learn. This method is called the classification algorithm. The classifier is trained with the collected data. After training, the model is ready to determine classes for new, unseen instances.

4. Evaluating the classifier. The classifier may be evaluated in order to determine its quality. The evaluation can be performed on both seen and unseen data. In the last case, a fraction of the data collected during the first step should not be used for training the classifier. During this phase, one can evaluate whether the amount training data is sufficient for the classifier to determine classes for other instances. If the performance of a classifier is poor, a new classifier can be created using other or more features such that the classifier can operate more accurately.

5. Utilizing the classifier. When a classifier with a desirable quality is created, the classifier can be used to solve the classification problem it was designed for.

For a given problem it is common that multiple classifiers are created and evaluated. A common option is that 70% of the data is used for training, while the remaining 30% is used for evaluation. This allows the user to select the most appropriate classification algorithm or to investigate the selection of features. It is possible that this approach reveals which technique is the most useful for the problem at hand. Based on the results a new classifier may be created which uses the entire collected dataset. That classifier will be used to solve the classification problem.

3.3 Classification Algorithms

Different algorithms exists to train a classifier. This section briefly covers three training methods. These methods do not completely cover the available classification algorithms, but give an idea how certain methods train classifiers.

3.3.1 Decision Trees

Decision trees are based on the tree data structure. Each node in a decision tree is a question which can be either true or false for an instance. Each leaf node contains one of the possible classes. The tree is built using the training data. For new instances the tree is traversed to determine the class. Starting with the root node, the questions of the nodes are answered for the given instance and the corresponding branch is taken. The questions are answered until a leaf node is reached. The class of the encountered leaf node is the predicted class for the given instance.

3.3.2 K-Nearest Neighbors

The k-nearest neighbors method treats each instance as a point in a space. The space depends on the format of the features. The dimension of the space equals the number of features selected as input data. A classifier is trained by defining the class of the points extracted from the training data. When offered a new

instance, the k-nearest neighbors method finds the k nearest points with respect to the location of the instance in the defined space. The classes of these nearest points are used to determined the class of the new instance by majority vote.

3.3.3 Support Vector Machines

Support vector machines like k-nearest neighbor also treat the instances as points in a space. Support vector machines try to define hyperplanes which separates the points into their corresponding classes based on the training data. When given a new point, support vector machines check between which hyperplanes the point is located. That outcome decides which class is predicted for the given instance.

3.4 Binary Classifier Evaluation

In machine learning, classifiers are evaluated by examining multiple inputs and comparing the predicted class with the expected class. In general, one can say that the quality of the created classifier depends how closely the predicted classes matches the expected classes. Within machine learning different metrics did arise to formally capture this notion in order to provide information of the quality of classifiers. This section covers the evaluation of binary classifiers.

3.4.1 2×2 Confusion Matrix

The performance of a binary classifier can be recorded using a confusion matrix [24, 39]. A confusion matrix is a special type of contingency table where the rows matches the columns. For a binary classifier the confusion matrix is a 2×2 matrix as depicted in Table 1.

		Predicted class	
		Positive	*Negative*
Actual class	*Positive*	*True Positives*	*False Negatives*
	Negative	*False Positives*	*True Negatives*

Table 1: 2×2 Confusion matrix for a binary classification problem.

The possible classes of any binary classification problem are usually *Positive* and *Negative*. Any definition of two classes can be rewritten into these classes. The confusion matrix summarizes which classes for the test instances are predicted. The rows indicate the classes of the instances in the validation data. The columns indicate the predicted class by the binary classifier. The entries in the confusion matrix count how many instances from the class specified by the row are predicted as from the class specified by the column.

3.4.2 Metrics

Within the literature some standard metrics are derived using the values from a confusion matrix [36]. The following values are defined on a 2×2 confusion matrix:

TP = *True Positives*; the number of instances correctly classified as *Positive*.

FN = *False Negatives*; the of number of instances erroneously classified as *Negative*, but are *Positive*.

FP = *False Positives*; the number of instances erroneously classified as *Positive*, but are *Negative*.

TN = *True Negatives*; the number of instances correctly classified as *Negative*.

The metrics for binary classifiers are constructed using these four values. The following metrics exist:

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

$$F\text{-}Measure = \frac{(\beta^2 + 1) \cdot TP}{(\beta^2 + 1) \cdot TP + \beta^2 \cdot FN + FP}$$

The minimum value for any of these metrics is 0, indicating that all instances are misclassified. The maximum value is 1, which corresponds to that all instances are correctly classified. Any classifier that randomly classifies instances will get 0.5 for each metric[1].

Recall indicates which fraction of the *Positive* instances is correctly classified. This can be interpreted how well the classifier recognizes the *Positive* instances. When a classifier classifies all instances to *Positive*, *Recall* is maximum. In order to distinguish this behavior *Precision* can be used to verify how likely a classifier will classify an instance to *Positive*. *Specificity* resembles the *Recall* metric for *Negative* instances.

Accuracy defines the fraction of instances that is correctly classified. This is useful for obtaining a general idea of the performance, but it does not state how well each class is recognized by the classifier.

F-Measure combines *Precision* and *Recall* to evaluate the performance of a binary classifier. Both metrics are weighted allowing the user to prioritize either metric. The weight for *Precision* and *Recall* is 1 and β^2 respectively. These weights are derived from Van Rijsenberg's effectiveness measure [43].

Besides the five binary metrics discussed above, other metrics exist for binary

[1] Assuming that the number of *Positive* and *Negative* instances in the validation data are equal.

classifiers. Examples include *AUC* [13], *Cohen's Kappa* [9], *Matthews Correlation Coefficient* [20] and *Youden's J Statistic* [45]. These metrics capture the performance of classifying instances of both classes, while the metrics *Recall*, *Precision* and *F-Measure* focus on the *Positive* instances and do not take the *True Negatives* into account. This is useful when both classes are of equal importance. However a drawback of *AUC*, *Cohen's Kappa*, *Matthews Correlation Coefficient* and *Youden's J Statistic* is that these metrics are solely defined for binary classifiers [19].

3.4.3 Example - Spam Filter

In order to demonstrate the metrics given in the previous section, an example is discussed. The example describes the evaluation of a spam filter. A spam filter should mark messages as spam or not-spam. The creation of a spam filter is based on the ability to classify whether a message is spam or not. Whether a message is spam is a binary classification problem, because there are two classes: spam messages and non-spam messages.

Suppose we collected a dataset of 300 messages. A spam filter is created by training a classifier using 200 instances of our data. The remaining 100 instances are used to evaluate the spam filter. 40 of the 100 messages are assumed to be spam, while the other 60 messages are considered non-spam. The predicted classes for these instances by the created classifier are listed in the confusion matrix given in Table 2. It is assumed that *Positive* is the class of spam messages and *Negative* is the class of non-spam messages.

		Predicted class	
		Positive	*Negative*
Actual class	*Positive*	36	4
	Negative	11	49

Table 2: A 2×2 confusion matrix for a spam filter. *Positive* is interpretted as being a spam message, while *Negative* is considered as being a non-spam message.

Using the confusion matrix the following values are defined:

$TP = 36$

$FN = 4$

$FP = 11$

$TN = 49$

Then the following values are assigned to the metrics of this classifier:

$Recall = 0.900$

$Precision = 0.766$

$Specificity = 0.817$

$$Accuracy = 0.850$$

$$F\text{-}Measure = 0.828 \text{ (for } \beta = 1)$$

Based on these values, the following conclusions can be derived. Considering *Recall*, the classifier is able to recognize 90% of the spam messages. However, of all messages that were classified as spam, only 77% was actually a spam message based on the value for *Precision*. *Specificity* indicates that 82% of the non-spam messages are correctly recognized, while the other 18% is erroneously classified as spam. Lastly, the value for *Accuracy* shows that 85% of all messages was correctly classified.

3.5 Multiclass Classifier Evaluation

The evaluation of multiclass classifiers does not differ much from the evaluation of binary classifiers. However, the evaluation metrics are solely defined using the metrics for binary classifiers [37]. Therefore, the evaluation of multiclass classifiers is discussed separately.

3.5.1 $n \times n$ Confusion Matrix

The performance of a multiclass classifier can be recorded using a confusion matrix [24]. Instead of using a 2×2 confusion matrix, a $n \times n$ confusion matrix is used, where n equals the number of classes of the corresponding classification problem. The rows list the different classes and the columns list the predicted classes. Each value in the confusion matrix indicates how many instances of class x are recognized as class y. These values can be used to check how many instances are correctly classified and how many instances are confused by the classifier for being in a different class. The general form of a confusion matrix is given in Table 3.

		Predicted class			
		Class 1	*Class 2*	...	*Class n*
Actual class	*Class 1*	Correctly classified 1's	1's confused for 2's	...	1's confused for n's
	Class 2	2's confused for 1's	Correctly classified 2's	...	2's confused for n's
	...	...	...		...
	Class n	n's confused for 1's	n's confused for 2's	...	Correctly classified n's

Table 3: $n \times n$ Confusion matrix for a classification problem consisting of n classes.

3.5.2 Transformation To 2×2 Confusion Matrix

Any $n \times n$ confusion matrix can be transformed to another confusion matrix of p rows and p columns when the n classes are grouped into p new classes. This concept is used in the definition of metrics for multiclass classification problems [24, 37].

Specifically, one transformation is used, which separates the classes in a one versus rest manner. This transformation transforms a $n \times n$ confusion matrix to a 2×2 confusion matrix. The *Positive* class consists of one specific class, while the *Negative* class consists of the remaining $n-1$ classes. The values in the resulting 2×2 confusion matrix are a defined by combining the individual entries of the given $n \times n$ matrix based on the new classes. This transformation is called $Flatten_i$ in this thesis, where i defines the class which becomes the *Positive* class.

More formally, $Flatten_i$ is a function on a $n \times n$ matrix A, which returns a 2×2 confusion matrix B. The entries of B are defined as follows:

$$
\begin{aligned}
TP &= A_{ii} \\
FN &= \sum_{k=1}^{i-1} A_{ik} + \sum_{k=i+1}^{n} A_{ik} \\
FP &= \sum_{k=1}^{i-1} A_{ki} + \sum_{k=i+1}^{n} A_{ki} \\
TN &= \sum_{k=1}^{i-1} \left(\sum_{m=1}^{i-1} A_{km} + \sum_{m=i+1}^{n} A_{km} \right) + \sum_{k=i+1}^{n} \left(\sum_{m=1}^{i-1} A_{km} + \sum_{m=i+1}^{n} A_{km} \right)
\end{aligned}
$$

An example of this transformation is given in section 3.5.4.

3.5.3 Metrics

The metrics for multiclass classifiers are based on the metrics for binary classifiers [37]. The metrics are defined using the entries of a $n \times n$ confusion matrix. It is assumed that there are n classes named 1, 2, ..., n. The performance of a multiclass classifier is given by a $n \times n$ confusion matrix A. Firstly, for each class i the confusion matrix B_i is determined using the $Flatten_i$ transformation as defined in the previous section. The following definitions are used, assuming $B_i = Flatten_i\,(A)$:

TP_i = *True Positives* of B_i

FN_i = *False Negatives* of B_i

FP_i = *False Positives* of B_i

TN_i = *True Negatives* of B_i

Each metric defined for binary classifiers can be applied on the confusion matrices B_i in order to determine the performance of the multiclass classifier per class. The overall performance of the multiclass classifier can be determined in two ways: macro-averaging or micro-averaging the values for a metric for the confusion matrices B_i [33, 40, 44]. With macro-averaging each *class* equally influences the metrics, while with micro-averaging each *instance* in the validaton data equally influences the metrics. So the bigger classes are favored with micro-averaging.

The metrics using macro-averaging [37] are defined as:

$$Recall_M = \frac{1}{n} \cdot \sum_i \frac{TP_i}{TP_i + FN_i}$$

$$Precision_M = \frac{1}{n} \cdot \sum_i \frac{TP_i}{TP_i + FP_i}$$

$$Accuracy_M = \frac{1}{n} \cdot \sum_i \frac{TP_i + TN_i}{TP_i + FN_i + FP_i + TN_i}$$

$$F\text{-}Measure_M = \frac{(\beta^2 + 1) \cdot Precision_M \cdot Recall_M}{\beta^2 \cdot Precision_M + Recall_M}$$

The metrics using micro-averaging [37] are defined as:

$$Recall_\mu = \frac{\sum_i TP_i}{\sum_i (TP_i + FN_i)}$$

$$Precision_\mu = \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)}$$

$$F\text{-}Measure_\mu = \frac{(\beta^2 + 1) \cdot Precision_\mu \cdot Recall_\mu}{\beta^2 \cdot Precision_\mu + Recall_\mu}$$

Interestingly, $Recall_\mu$, $Precision_\mu$ and $F\text{-}Measure_\mu$ will give the same value for any confusion matrix (see Appendix A). This metric resembles the formula for $Accuracy$ and can be considered as $Accuracy_\mu$. $Accuracy_\mu$ indicates like $Accuracy$ which fraction of the models is correctly classified. Using A_{ij} to denote the value in the ith row and jth column in A, $Accuracy_\mu$ is defined as:

$$Accuracy_\mu = \frac{\sum_i A_{ii}}{\sum_i \sum_j A_{ij}}$$

Like the binary variants, the minimum value for all metrics listed above is 0. A value of 0 indicates that all instances are misclassified. When all instances are correctly classified, each metric attains the maximum value of 1.

3.5.4 Example - Simple Handwriting Recognition

This section demonstrates how the metrics are determined for a multiclass classifier. The evaluation of a simple handwriting recognition system is discussed. The purpose of such a system is to determine which word or character is written given by an image. It is assumed that the simple handwriting recognition system examines individual characters. The goal is to determine whether the character is a letter, a digit or a punctuation mark. The evaluation of the simple handwriting recognition system is given using a 3×3 confusion matrix as given in Table 4.

For each class, a 2×2 confusion matrix is determined using the transformation $Flatten_i$ as defined in section 3.5.2. The values of these confusion matrices are

		Predicted class		
		Letter	*Digit*	*Punctuation*
Actual class	*Letter*	37	13	2
	Digit	21	56	8
	Punctuation	5	0	42

Table 4: A 3×3 confusion matrix for a simple handwriting recognition system.

		Confusion matrix		
		$Flatten_{Letter}$	$Flatten_{Digit}$	$Flatten_{Punctuation}$
Value	TP_i	37	56	42
	FN_i	15	29	5
	FP_i	26	13	10
	TN_i	106	86	127

Table 5: 2×2 confusion matrices derived from Table 4.

listed in Table 5.

The values in the 2×2 confusion matrices give the following values for the metrics for the simple handwriting recognition system:

$Recall_M = 0.755$

$Precision_M = 0.736$

$Accuracy_M = 0.822$

$F\text{-}Measure_M = 0.745$ (for $\beta = 1$)

$Recall_\mu = 0.734$

$Precision_\mu = 0.734$

$F\text{-}Measure_\mu = 0.734$ (for any real constant β)

$Accuracy_\mu = 0.734$

3.6 Metrics For Cost-sensitive Classifiers

For some classification problems the classification of each instance is linked to a reward or cost [10, 12]. When an instance is correctly classified there is a reward (or negative cost) and when an instance is incorrectly classified there is a cost (or negative reward). This cost may depend on which predicted class is chosen for a certain class. For this type of classifiers the total expected reward or cost is more important than the number of instances correctly classified.

3.6.1 Metrics

The concept of classifying instances is paired with cost or reward, can be formally described. Given a confusion matrix A each entry has a weight W_{ij} which specifies the cost of classifying an instance of class i as class j. The

metric *Cost* can be defined as sum of the individual classification costs. Likewise, *Reward* can be defined as the inverse of *Cost*. A cost-sensitive classifier performs better when the cost is lower or the reward higher. These metrics defined as:

$$Cost = \sum_i \sum_j W_{ij} \cdot A_{ij} \begin{cases} W_{ij} \leq 0 & \text{if } i = j \\ W_{ij} \geq 0 & \text{if } i \neq j \end{cases}$$

$$Reward = \sum_i \sum_j W_{ij} \cdot A_{ij} \begin{cases} W_{ij} \geq 0 & \text{if } i = j \\ W_{ij} \leq 0 & \text{if } i \neq j \end{cases}$$

Unlike the metrics examined so far, the metrics *Cost* and *Reward* do not have a minimum or maximum value. Both the minimum and maximum value depend on the number of instances per class in the validation data and the weights W_{ij}. Nevertheless, these metrics can be used to compare the classifier with a predefined threshold or with another classifier evaluated with the same validation data.

3.6.2 Example - Comparing Spam Filters

Spam filters can be used to automatically clean inboxes by removing the messages which are marked as spam. This can be convenient for the user because one does not have to deal with the messages marked as spam. However, is unfortunate when non-spam messages are deleted because the spam filter marked these messages as spam. The user may miss fundamental information contained in these messages.

Suppose there are two spam filters available, whose performance is given by confusion matrices depicted in Table 6 and 7 respectively. Further, it is assumed that misclassifying a non-spam message is five times worse than misclassifying a spam message. Then the weights[2] W are expressed by the matrix $\begin{bmatrix} 0 & 1 \\ 5 & 0 \end{bmatrix}$.

		Predicted class	
		Spam	*Non-spam*
Actual class	*Spam*	48	2
	Non-spam	8	42

Table 6: A 2×2 confusion matrix for spam filter option A.

		Predicted class	
		Spam	*Non-spam*
Actual class	*Spam*	36	14
	Non-spam	1	49

Table 7: A 2×2 confusion matrix for spam filter option B.

Although classifier option A is more accurate than classifier option B, the cost of using classifier option A is 42 and the cost of using classifier option B is only 19.

[2]It is assumed that determining the correct class for a message induces no cost. Therefore, $W_{0,0} = 0$ and $W_{1,1} = 0$.

This suggest classifier option B is better, which corresponds to the reality since it is less likely that a non-spam message is erroneously is classified as spam.

4 Related Work

This chapter discusses the work related to our research. Section 4.1 shows that the use of symbolic state space methods is necessary but has limitations too. Section 4.2 discusses different proposed methods to improve symbolic state space methods. To our best knowledge, using the features of a model to predict a strategy is a fairly new concept to improve state space exploration. Section 4.3 discusses work related to strategy prediction.

4.1 Symbolic State Space Methods

State space methods rely on examining all reachable states for a given model. The process of discovering the reachable states is called the state space exploration. During state space exploration the discovered states can be either stored explicitly or symbolically.

When the states are stored explicitly, each discovered state is stored in memory, occupying a specific amount of memory. This approach is not suitable for large state spaces. Lets assume that a given model has over 10^{20} reachable states and each state can be stored in 128 bits. Without overhead, this approach will take over 1.6 zettabyte[3] of memory. This amount of memory is simply too large for any computer and therefore the use of explicit state space methods is limited to smaller state spaces.

Symbolic state space methods are able to deal with large state spaces [6]. Symbolic state space methods store the discovered states in a Binary Decision Diagram (BDD) or a variant of BDDs. Unfortunately, symbolic state space methods are limited by time and memory too [8]. On top of that, there is no clear relation between the number of states stored and the size of the BDD [32]. During state space exploration the size of the BDD may be much larger than the final BDD describing the entire state space. The largest size encountered during exploration is called the peak size of de BDD. The peak size is often reached midway during the state space exploration. The peak size may be hundreds or thousands of times larger than final size of the BDD, placing a high demand on resources [8]. Therefore, it is important that during the exploration of the state space, the size of the BDD is kept as small as possible in order to utilize the available resources the most efficient.

4.2 Improvements Of State Space Methods

Due to the state space explosion problem [41] it may be unfeasible to directly apply either explicit or symbolic methods on given models. Since the introduction of state space methods, much research is performed to enhance state space methods, allowing to investigate more models and larger state spaces. This section gives an overview of improvements which are related to our research, which aims to improve state space methods.

[3] 1.6 zettabyte equals $1.6 \cdot 10^{12}$ GB

4.2.1 BDD Construction

Symbolic state space methods rely on BDDs. The way the BDD is constructed affects the performance of a state space method. The peak size of the BDD determines the amount of resources needed [8]. Since the peak size may be significantly larger than the final size of the BDD, it is important that the size of the BDD is kept as small as possible. Maintaining a small BDD allows to explore larger state spaces.

The size of the BDD depends on how the variables in the BDD are ordered. The variables can be reordered in order to reduce the size of the BDD, but determining this order is a NP-complete problem [4]. Therefore it is not feasible to perform variable reordering according to the best variable order during state space exploration.

Since it is not feasible to maintain the best variable ordering during exploration, other methods were proposed to reduce the peak size of a BDD. Some methods aim to find a good variable ordering beforehand [14, 22, 35] or try to maintain a good ordering during exploration [31].

4.2.2 Partitioning Of Transitions

Another improvement is to divide the transitions of the model into groups [5]. This method is able to verify models with 10^{50} and 10^{120} states and is required to investigate models with large state spaces. This idea was applied in the PINS interface of LTSmin [3]. The partition was improved by distinguishing read and write-dependencies and it was shown that a significant improvement can be achieved [21].

4.2.3 State Space Traversal Techniques

The construction of BDDs also depends on order of the states discovered during exploration, so the way the state space is traversed influences the size of the BDD. As result, different traversal algorithms were proposed.

The classical breadth-first-search (BFS) is a suitable traversal algorithm for state space exploration. An alternative traversal algorithm called chaining was proposed by [30]. This traversal technique was applied on Petri Nets and was found two orders of magnitude faster than regular BFS. However, they admitted that this statement is not verified on larger models.

Ciardo et al [8] gave a variation of both BFS and chaining where all states were used instead of only the previous discovered states. Their results showed that these variations were overall slightly better, but for some models they worsen the amount of resources needed for the state space exploration. They showed that chaining is marginally better than BFS. However they only presented these results for five selected models.

Sóle et al [38] introduced four additional traversal algorithms to reduce the peak size of the BDD. Their algorithms aim to improve the state space exploration

of concurrent systems and showed that in most cases an improvement can be achieved. Still, chaining was found to be a good alternative.

4.2.4 Saturation

Saturation is another method to greatly reduce the peak size of a BDD [7, 8]. The general idea is that only transitions at a certain level in the BDD are fired. When all transitions are applied, that level is saturated and a higher level is examined. By saturating levels in the BDD, the size of the BDD is kept small. It was reported that this method may achieve several orders of magnitude better performance [7].

4.3 Strategy Prediction

Closely related to our research is the work of Pelánek. Firstly, he defined properties of state spaces and specifies groups of state spaces [26, 29]. These characteristics were used to find an appropriate strategy for a model beforehand [27]. This ultimately lead to the development of EMMA which implements the prediction of a strategy for a model [28]. However, his work was limited to explicit state space methods. Furthermore, only the models from the BEEM database [25] were used. Our research includes a wider array of models and focuses on symbolic state space methods.

5 Research Questions

This chapter discusses the research questions of this project. The research questions specify which problems the research aims to solve in order to reach the goal specified in chapter 1, namely letting the model checking tools itself decide which strategy is suitable for a given model. The research is formulated using one main question described in section 5.1. The main question is subdivided in three research questions which are described in section 5.2, 5.3 and 5.4. Section 5.5 summarizes the research questions.

5.1 Main Question

As mentioned in chapter 1, state space methods may consume many resources due to the state space explosion problem [41]. Currently many different solutions are proposed to tackle the state space explosion problem. The user is provided multiple strategies to solve its models. Because of the wide variety of models and options, it may be hard for the user to select an appropriate strategy. Selecting an inappropriate strategy may lead to a waste of resources or that the model checking tool is unable to answer the given verification questions.

Therefore, the model checking tools itself, instead of the users, should decide which strategy should be applied on a given model. This research investigates whether it is possible for a model checking tool to select an appropriate strategy using only the information presented in the given model. The pieces of information embedded in a model are called the features of a model in this report. Since most verification questions are based on the state space of a model [41], the problem of verifying a model is refined to the problem of determining the state space of a model. This leads to the following main question:

To what extent can the features of a model be used to predict an appropriate strategy in order to improve state space exploration?

It is not necessary to come up with the absolute best strategy currently available. It is sufficient to find a strategy which performs almost as good as the best strategy given a specific model.

This main question is answered using three research questions specified in the remaining of this chapter.

5.2 Research Question 1

Because of the wide varieties in strategies and models, it is unknown how well the strategies perform on different models. This leads to the first research question:

How does the strategy influence the state space exploration of a model?

The answer on this question determines whether it is relevant at all to even examine the features of the model. We investigated how many resources it takes to explore the state space of model given a strategy. This investigation reveals the

differences between multiple strategies for a model. If the differences are negligible, any strategy can be selected and the features of the model does not matter.

On the other hand, we examined whether there exists a superior strategy. A superior strategy is a strategy which performs better than any other strategy for almost all models. If any superior strategy exists, that superior strategy should be selected without even considering the features of the models.

5.3 Research Question 2

Assuming the selection of the strategy significantly influences the state space exploration, a best strategy can be selected per model. We investigated how the features of the model relates to its best strategy. This investigation is captured by the second research question:

Which features are relevant to consider when predicting a strategy for a model?

The answer on this question reveals which features are relevant to consider and which features does not provide any helpful information for selecting an appropriate strategy. This information should be exploited by the model checking tools such that during the strategy selection the relevant features are considered and the redundant features are ignored.

5.4 Research Question 3

Lastly, we investigated whether the features provide sufficient information to determine an appropriate strategy. The last research question states whether it is possible to predict an appropiate strategy using the features of a model:

To what extent can we make a prediction of an appropriate strategy given the features of a model?

We checked whether it is possible to determine an appropriate strategy given the features of a model. Since the predicted strategy may depend on multiple features on different domains, it is expected that there does not exists a trivial relation between the features and the best strategy of a model. Therefore, supervised machine learning techniques are used to predict an appropriate strategy given the features of a model.

5.5 Summary

We want to improve state space exploration of models by predicting an appropriate strategy based on the features of models. This improvement is realized by investigating whether strategies significantly perform differently for a fixed model and whether there is no superior strategy. This information reveals whether examining features is relevant at all or that a specific fixed strategy should be selected. It is investigated whether features provides *any* information for an appropriate strategy and whether features provides *sufficient* information for predicting an appropriate strategy.

6 Methods

This chapters discusses the methods used in our research. Section 6.1 defines the scope of our research by defining which set of strategies and features are considered. Section 6.2 defines the techniques and materials used in our research. Section 6.3 discusses how the research was performed.

6.1 Scope

The main goal of this research is to investigate to what extent features of a model can be used to predict a strategy. A strategy is defined as any way in which a state space method is applied to solve verification questions. In chapter 5 it was mentioned that the verification questions are confined to the state space exploration. A feature is defined as any property of a model. The terms strategy and feature are too broad to cover in a single research. This sections explains which strategies and features were selected for our research.

6.1.1 Selected Strategies

The number of strategies had to be limited in order to make testing feasible. Considering related work, it can be observed that the traversal method and saturation may significantly influence the state space exploration of a model. Therefore, these two aspects were set variable in a strategy and other aspects of a strategy were fixed. Some of the models which were used for testing, have huge state spaces. The explicit backend of LTSmin was not suitable for these models. Therefore, the symbolic backend of LTSmin was selected.

Four different traversal algorithms were incorporated in our research: `bfs`, `bfs-prev`, `chain` and `chain-prev`. These algorithms are based on the ones found in [8]. The pseudo code of the algorithms as implemented in LTSmin is given in [34]. The traversal algorithms introduced by [38] are not available in LTSmin and were not considered.

LTSmin offers the ability to do saturation, but it differs from the original method [8, 34]. LTSmin offers multiple options to perform saturation and forces the user to divide the dependency matrix into saturation levels. These levels are defined by a parameter called saturation granularity. The saturation granularity indicates the column width of each level[4]. The user has to provide a positive integer for this parameter or otherwise a default value of 10 is used.

Multiple different saturation granularities were considered including the extreme values. The minimum value for saturation granularity is 1. The maximum value equals the width of the dependency matrix and any higher value results in the same performance. Since the width of the dependency matrix depends on the model, the maximum value cannot be fixed precisely. Instead the maximum signed integer value (2147483647) was used in order to capture all models.

[4] In the case the saturation granularity does not divide the width of the dependency matrix, the last level consists of the remaining columns. For example, if the width of the dependency matrix is 53 and the saturation granularity is 10, there are 5 levels consisting of 10 columns and one level consisting of 3 columns.

The values 1, 5, 10, 20, 40, 80 and 2147483647 were selected for saturation granularity.

Two saturation methods called `sat-like` and `sat-loop` were considered. Combining the saturation methods with the selected values for saturation granularity, 14 different saturation strategies were defined. It was also tested how well the models were explored without using saturation (saturation method = `none`). The saturation granularity does not have any meaning when no satuation is used for exploring the model. Incorporating `none` as saturation strategy gave an additional strategy. So in total, 15 different saturation strategies were considered. The variables and selected values for strategies are listed in Table 8.

Variable	**Selected values**
Traversal strategy	`bfs`, `bfs-prev`, `chain`, `chain-prev`
Saturation method	`sat-like`, `sat-loop`, `none`
Saturation granularity	1, 5, 10, 20, 40, 80, 2147483647

Table 8: Selected variables and values for the strategies. Note: saturation granularity was only used when `sat-like` or `sat-loop` was selected as saturation method.

The traversal strategy and saturation strategy can be chosen individually for any model. The saturation method defines how the saturation levels are visited, while the traversal strategy defines the method to visit states within a saturation level. So any of the 4 traversal strategies can be combined with any of the 15 saturation strategies, defining 60 different strategies. These 60 strategies are considered during our research.

Besides the variable aspects of the strategies, some aspects were fixed for all strategies. These fixed aspects are listed in Table 9. The reordering strategy was selected from [22] because that reordering strategy had the best overall performance. `save-sat-levels` is time optimization flag for both `sat-like` and `sat-loop`. This flag has no effect when `none` is selected as saturation method. `vset` specifies the BDD package used. The other aspects enforces that a fixed amount of resources can be utilized for solving a model.

Aspect	**Selected value**
Reorder strategy	`tg,bs,hf`
`save-sat-levels`	`true`
`vset`	`lddmc`
`lace-workers`	1
`ldd-cachesize`	26
`ldd-tablesize`	26
`ldd-maxtablesize`	26

Table 9: Selected values of fixed aspects for all strategies.

All other parameters and flags of LTSmin that are not listed in either Table 8 or 9 attain their default values as defined for LTSmin version 2.1 for all strategies.

6.1.2 Selected Features

Everything based on the information of the model can be considered a feature. The set of features considered were also limited. Since LTSmin is able to handle different types of models, model specific features were not considered. Model specific features are based on the specification language used to describe the model. The maximum number of arcs leaving any place in a Petri Net or the number of lines in a DVE model are examples of model specific features.

All features considered in this research were based on features that can be derived for all models LTSmin is able to analyze. This design choice allows that the results produced by this research can be applied to a large variety of specification languages (see section 2.2). Hence, only features which can be derived from the information available in the PINS interface were considered.

Feature	**Definition**
State Vector Length [16]	Width of the dependency matrix.
Groups [16]	The number of transition groups. This corresponds to the number of rows in the dependency matrix.
Bandwidth [22]	Maximum bandwidth of all rows in the dependency matrix after variable reordering.
Profile [22]	Sum of the bandwidth of all rows in the dependency matrix after variable reordering.
Span [22]	Sum of the span of all rows in the dependency matrix after variable reordering.
Average Wavefront	Sum of the wavefront of all rows in the dependency matrix after variable reordering divided by the number of rows.
RMS Wavefront	Root mean squared over the wavefront of all rows in the dependency matrix after variable reordering.
Event Span (*ES*) [22]	Sum of the span of all rows in the dependency matrix before variable reordering.
Normalized Event Span (*NES*) [22]	Normalized version of *Event Span* which allows to compare dependency matrices of different sizes.
Weighted Event Span (*WES*) [22]	Weighted variant of *Event Span*.
Normalized Weighted Event Span (*NWES*) [22]	Normalized version of *Weighted Event Span* which allows to compare dependency matrices of different sizes.

Table 10: Selected features of the model based on metrics on the dependency matrix.

The PINS interface provides a dependency matrix of a model. There are 11 metrics defined on the dependency matrix. Each metric somehow reflects a part of the nature of the dependency matrix and can be used as feature. These 11 metrics were selected as feature and are listed in Table 10. 9 of these metrics

are based on the following metrics per row [22]:

Bandwidth of a row = Maximum distance of any nonzero entry in a row to the diagonal of the dependency matrix.

Span of a row = Distance between the leftmost and rightmost nonzero entry in a row.

Wavefront of a row = The number of adjacent vertices of all vertices smaller or equal to the vertex corresponding to the row.

6.2 Techniques

This section gives an overview of the models and techniques used during our research. Firstly, the set of models which was used during our research is discussed. Secondly, the tools and programs which were used are listed. The last section gives an overview of the used machines for determining the performance of the selected strategies.

6.2.1 Model Collection

The selected strategies were tested on a large number of models in order to acquire sufficient information. The PNML models of the Model Checking Contest [17] were used. Since LTSmin currently cannot handle colored Petri Nets, only the unfolded Petri Nets were considered. This collection consists of 491 different models. These models are instances of 66 different parameterized Petri Nets[5]. Furthermore, 293 DVE models of the BEEM database[6] were used [25]. These models are instances of 56 parameterized DVE models[7]. So in total 784 different models were used.

6.2.2 Tools And Programs

This section lists the tools and programs used during our research. See section 6.3 for the role of each program.

Overview of the tools and programs:

- awk version 4.0.1
- bash version 4.3.11(1)
- DiVinE [1] version 2.4
- grep version 2.16
- LTSmin[8] A development version was used because there was no stable release which could handle PNML models correctly. LTSmin was build from commit `fbc4d4999d06134984f076eed2f0b8523cfb5704`.

[5] This is the collection of the ≪Surprise≫ and ≪Known≫ models for 2016. The model DotAndBoxes does not offer an unfolded Petri Net variant and is not included in the test cases.

[6] Available on `http://paradise.fi.muni.cz/beem/`, accessed July 2016.

[7] All seven instances of the model train-gate could not be compiled using DiVinE. These models are left out of the test cases.

[8] Available on `https://github.com/Meijuh/ltsmin/tree/next`, accessed May 2016.

- MEMTIME[9]
- PYTHON version 2.7.6
- R[10] version 3.3.1
  - GGPLOT2 package version 2.0.0
  - PLYR package version 1.8.3
  - RESHAPE2 package version 1.4.1
- SCIKIT-LEARN[11] version 0.17.1

6.2.3 Machines

The computer cluster of the University of Twente was used to measure the performance of the selected strategies. Specifically, 44 machines were used. All machines operate under UBUNTU 14.04 LTS and each machine has 64 GB RAM. The CPU of 32 machines is 2× AMD OPTERON 4386, while the CPU of the other 12 machines is 1× AMD OPTERON 4386. The AMD OPTERON 4386 has 8 cores.

6.3 Method

This section explains which actions are performed during our research in order to answer the research questions stated in chapter 5. Firstly, the performance of the selected strategies was determined for the selected models. The resulting data was refined to eliminate errors and remove redundant data. The performance data was investigated in order to establish whether there exists a superior strategy and to what extent the differences between multiple strategies matter. The performance data was used to select the best strategy with respect to time and peak size per model. This data was used to create multiple classifiers. The performance of the classifiers was investigated in order to establish to what extent the features of a model can predict an appropriate strategy.

The technical details of the created scripts and programs are omitted during the discussion of the research method. Appendix D gives insight in the scripts and discusses how the scripts can be utilized to reproduce the results given in this thesis.

6.3.1 Running State Space Exploration Tests

The purpose of our research is to determine to what extent features of a model can predict an appropriate strategy. In order to measure whether a certain predicted strategy is appropriate, the performance of multiple strategies need to be known for a given model. In section 6.1 the strategies and features were selected. The first step consisted of collecting the performance data of the 60 selected strategies for all 784 selected models.

[9] Available on `http://fmt.cs.utwente.nl/tools/scm/memtime.git/`, accessed October 2016.
[10] Available on `https://cran.r-project.org/mirrors.html`, accessed October 2016.
[11] Available on `http://scikit-learn.org/stable/`, accessed July 2016.

The PNML models[12] were extracted from the website of the MCC. The unfolded Petri Nets were separated from the colored Petri Nets using GREP. LTSMIN cannot handle DVE models directly. DIVINE [1] was used to compile the DVE models to DVE2C format.

In order to determine the performance of a strategy on a model accurately, multiple performance runs were performed. The performance of LTSMIN is slightly variable due to the load of the system where the tool is running on. Therefore, ten performance runs per model and strategy pair were performed. Besides the performance runs per model and strategy pair, a single statistic run was performed. The purpose of this run is to measure the peak size. Enabling the collection of statistics in LTSmin negatively influence the performance. So the statistics were collected in a separate run. Since LTSmin is deterministic, one statistics run per model and strategy pair suffices. In total, eleven runs were performed per model and strategy pair.

For each model, the values for the features *ES*, *NES*, *WES* and *NWES* were determined in a separate run. The values for the other seven features could already be extracted from the other runs. The features of a model are in independent of the selected strategy. The collection of the values of these four features required an additional 784 runs.

This approach required 518224 runs[13]. The computer cluster of the University of Twente was used to process this large amount of test cases. An existing BASH script was available which was able to generate test cases and automatically schedule the generated test cases on the cluster. This script was modified in order to run with the collected models and the configurations of LTSMIN. The BASH script generated a small script per run. Each small generated script performed one run with the LTSMIN tool. The bash script also generated a scheduler script. The generated scheduler script schedules the small generated scripts on the cluster. This method allowed that batches of test cases could be scheduled and executed on the cluster.

It is known that some PNML models are too complex to be analyzed [17]. Because of the large number of test cases, it was preferred that many test cases could run in parallel. Therefore, each test case acquired one CPU core. Each test case acquired 8 GB of RAM. This amount is sufficient to store the BDD and the model itself. If the run exceeded the amount of memory or took too long to solve, the test case was aborted. The time limited was set to 30 minutes. These constraints were enforced by using MEMTIME. Furthermore, MEMTIME recorded the elapsed time and memory footprint.

6.3.2 Processing Data

The output of LTSMIN is plain-text. After running all test cases on the cluster, the output of each run was stored in a txt-file. Due to the large amount of files, it is impossible to analyze those files by hand. 30 files were randomly selected

[12] Available at `http://mcc.lip6.fr/models.php`, accessed October 2016.
[13] The number of runs is equal to $(11 \cdot 60 \cdot 784) + 784$

to identify the possible formats LTSmin could produce. This information was used to create a parser in BASH. The programs GREP and AWK are natural to use to extract information of text and were used extensively by the parser. The parser notified the user if it did not recognize the format of the provided text-file. In this way, new formats were discovered and the parser was updated for those formats. As result, all txt-files were successfully parsed. The results were stored in a csv-file.

The programming language R was used for the analysis of the csv-file. R offers various tools to filter and aggregate data, and it is natural to use for data mining. The packages PLYR and RESHAPE2 were used for data compression. The data produced by the parser was not suitable to analyze directly. The data had the following issues:

- Some models are unsolvable for all 60 selected strategies. These models cannot be used, since it is impossible to measure whether the predicted strategy is appropriate for these models. Furthermore, because some models are too large, even the values of some features could not be determined.
- Some models were not solved correctly. For most models the actual or a lower bound of the size of the state space is known. There are 59 models where all selected strategies could not solve the model or produced wrong results. Since these models were solved incorrectly, the performance of the strategies may not reflect the actual effort needed to solve the models correctly. These models were removed from the data.
- The data contains contradicting entries. This was revealed by the size of the calculated state space for a model. The state space is always fixed for a given model, so LTSmin produced faulty results for some models. For some models in combination with a selected saturation granularity of 1, the state space was reported extremely small[14], while it is known that the state space is significantly larger.
- The saturation granularity was selected before solving a model. The meaning of saturation granularity depends on the width of the dependency matrix of the model. If the saturation granularity is larger than the state vector length, the strategy does not differ from the equivalent strategy with a saturation granularity of 2147483647. Hence for smaller models, multiple different strategies may collapse in a single strategy, although they are marked as different strategies in the data.
- Due to the multiple runs per model and strategy pair, the information of the performance of a model with respect to a strategy is scattered over the entire csv-file. The performance of a strategy on a model is defined by at least eleven lines.

Before the data was analyzed, the issues mentioned above were addressed. Firstly, all runs of the models which could not be solved correctly by any strategy, were removed. These 59 models were identified by comparing the

[14] Less than eight states.

produced results with the actual or lower bound of the state space known for the model. From the remaining entries all entries were removed where the state space was smaller than eight. This limit was chosen, because the smallest state space of all models consists of eight states. The data was further refined by removing the entries where none of the remaining strategies could solve the model.

The conflicting entries were removed using voting. The entries per model were grouped in non contradicting sets. The largest set was assumed to be true for each model. All entries contradicting the selected true entries were removed. The saturation granularity was fixed by comparing it with the discovered *State Vector Length*. The remaining entries contain the *State Vector Length* of any model that could be solved. If the listed saturation granularity is larger than the *State Vector Length*, the saturation granularity was altered to 2147483647. The remaining entries were summarized such that only one entry in the resulting csv-file describes the performance of a model and strategy pair.

6.3.3 Analyzing Performance Data

After the processing phase, the data in the csv-file is consistently describing the performance of all models that could be solved by any of the 60 selected strategies. This performance data was analyzed in order to answer the first research question (see section 5.2). The results of this analysis are described in chapter 7.

The performance data describes the best strategy per model. A strategy is better than other, when it uses less resources than the other strategy. In our research, we considered two aspects: elapsed time and peak size. Elapsed time specifies the amount of time needed to solve a model. Obviously, a strategy with a low solving time is preferred over any strategy that is slower to solve a model. Peak size correlates to the minimum amount of memory needed during the exploration of a model. A lower peak size is preferred when the amount of available memory is limited.

Whether a strategy with a low elapsed time and a high peak size is preferred over a strategy with a high elapsed time and a low peak size, depends on which resource is more valuable for the user. Therefore, both aspects were investigated separately. For each aspect and model the best strategy was determined.

6.3.4 Creating And Evaluating Classifiers

For each aspect five classifiers were created. The performance data collected was used to train and verify the classifiers. One classifier was trained with the PNML models and verified with the DVE models. Another classifier was trained with the DVE models and verified with the PNML models. The other three classifiers were trained with a random selection of 70% of both PNML and DVE models and verified with the remaining 30%. We chose to train three classifiers with a random set of models in order to prevent the results were based on a single split. The selection of models is based on the parameterized models. This prevented that the classifiers are evaluated with a model it is

trained with. Details of the division of data into training and validation data is given in section 8.1.

The tool SCIKIT-LEARN was used to train the classifiers. SCIKIT-LEARN is an open source library for PYTHON and provides multiple libraries offering machine learning techniques. The goal of the classifiers is to predict strategies for new unseen models.

Prediction of the traversal algorithm and the saturation method are classification problems, since the predicted values should be in a finite set. The prediction of the saturation granularity resembles a regression problem since it is defined as value on a continuous infinite domain. However, the prediction of the saturation granularity is treated as a classification problem because of two reasons. Firstly, saturation granularity = 2147483647 differs from the other saturation granularities since it specifies the entire width of the dependency matrix. The selected values are not on a continuous domain. This makes the problem hard for the computer, since it has to learn the gap between the low saturation granularities 1, 5, 10, 20, 40, 80 and the high value 2147483647. Secondly, there is no data available when the classifier predicts a saturation granularity other than the selected values. Hence, predicting the saturation granularity is treated as a classification problem. The prediction of a strategy can be described as classification problem, since the classifiers had to select the best strategy from a finite set of 60 classifiers.

SCIKIT-LEARN offers multiple classification algorithms. In order to determine the most suitable algorithm, all available algorithms were evaluated using the three training datasets consisting of a random selection of 70% of the models. The algorithm with the highest performance was selected. The selection of the classification algorithm is discussed in section 8.3.

After determining the classification algorithm, for each aspect the five classifiers as defined above were created. Each classifier was evaluated in two ways. Firstly, a classifier was evaluated using the metrics as introduced in section 3.5.3. However, the metrics enforce that only the predictions which select the best strategy are rewarded. If a classifier predicts the strategy partly correct, it is marked as misclassification. Hence, the classifiers were also evaluated using the performance data. For each prediction the difference between the predicted strategy and the best strategy was determined in order to evaluate whether the predicted strategy is appropriate. The results of the trained classifiers are discussed in chapter 8. The results provide an answer to the third research question (see section 5.4).

6.3.5 Feature Relevance Analysis

The classifiers were trained using all 11 selected features. In order to determine whether the value of a feature is relevant for predicting an appropriate strategy, more classifiers were trained using different subsets of the 11 selected features.

For each aspect and subset of features, three classifiers were created. These classifiers were trained using the same random selections of 70% of the models.

The metrics of these classifiers were used to evaluate the performance of the classifiers. The differences in metrics indicate the effect of missing certain features. That information was used to investigate the relevance of each feature. The results of this investigation is discussed in chapter 9. The results provide an answer to the second research question (see section 5.3).

7 Strategy Evaluation

This chapter evaluates the performance of the selected strategies based on the experiments performed in this research. The results are limited to the models where at least one strategy is able to solve the model. In section 7.1 it is described how the strategies are named. Section 7.2 explains the refinement of the collected data and shows which models are considered. Section 7.3 discusses which models the strategies are able to solve. Section 7.4 and 7.5 discusses the performance of the strategies with respect to time and memory usage respectively. Section 7.6 gives a conclusion based on the information presented in this chapter. This conclusion provides an answer to the first research question as stated in section 5.2.

7.1 Naming Convention

In order to discuss the performance of the strategies, each strategy was given a unique name. The name consists of the saturation strategy followed by the traversal strategy. The saturation granularity is denoted between brackets. The traversal strategy is abbreviated a follows:

`b` = `bfs`

`bp` = `bfs-prev`

`c` = `chain`

`cp` = `chain-prev`

For example, `sat-like(20)-c` indicates the strategy using the `sat-like` method with a saturation granularity of 20, using `chain` as traversal strategy. Likewise, `none-bp` is the strategy that uses no saturation and uses `bfs-prev` as traversal strategy.

7.2 Data Refinement Results

As discussed in section 6.3.2, the data collected from the test runs had some imperfections. Before the data was analyzed, the data was refined. This section shortly discusses the results observed during the refinement process.

The performance and statistics runs resulted in a total of 517440 entries in the data. Firstly, the 59 models which could not be solved correctly were removed. Since there are 660 entries per model, 478500 entries were remaining after deleting all results of these 59 models.

Secondly, the entries describing that the size of the discovered state space is smaller than 8 were removed. 2614 entries fell in that category. The remaining 475886 entries still consisted of 725 different models.

The first two refinement steps removed a large number of the errors produced by LTSmin. The models that could not be solved by the remaining strategies in the data were removed. These models provide insufficient information to evaluate a predicted strategy. This step reduced the entries to 282643. The

data consisted of 432 different models.

The data provides sufficient information to evaluate the predictions made for 432 different models. There were still conflicting entries in the data due to wrongly calculated state spaces. These entries were removed using voting. The contradicting outcomes provided by the smallest number of strategies were removed. This reduced the entries by 4138. This refinement step did not alter the total number of models covered by the data. The remaining 278505 entries still consisted of 432 different models.

The remaining entries were summarized. Firstly, the saturation granularity was adjusted depending on the discovered *State Vector Length*. Since there exists at least one strategy which can solve the model, the data contained the *State Vector Length* for all remaining 432 models. After this adjustment, the data was summarized in one entry per model and strategy pair. The time performance per pair equals the mean of the elapsed time of the performance runs. The elapsed time for each run that resulted in a timeout was set to 1800 seconds. This corresponds to the maximum time given for each in experiment. In this way, the total number of timeouts were incorporated in the time performance. The data was summarized in 18962 entries, consisting of 221 different PNML models and 211 different DVE models.

7.3 Strategy Capability

This section discusses the capability of the 60 selected strategies. For each strategy the total number of different models that can be solved was determined. Since there are 432 different models in the data, the best possible value for a strategy is 432. The results are depicted in Figure 2 ordered by the capability. The corresponding values of eight strategies are listed in Table 24. Appendix B contains a complete overview of the values in Figure 2.

Strategy	**Number of models**	**Coverage**
`sat-like(5)-cp`	402	93.06%
`sat-like(5)-bp`	397	91.90%
`sat-like(5)-c`	393	90.97%
`sat-like(2147483647)-b`	356	82.41%
`sat-like(1)-cp`	333	77.08%
`sat-like(20)-cp`	276	63.89%
`sat-like(40)-cp`	138	31.94%
`sat-like-(80)-c`	96	22.22%

Table 11: Capability of some strategies in Figure 2. The capability is measured by the number of different models the strategy is able to solve. The coverage shows which fraction that number is of the total set of 432 models.

Using these results, it can be observed that the most capable strategy is `sat-like(5)-cp`. This strategy is able to solve 402 different models. However, there exist 29 models that cannot be solved using this strategy. The other strategies perform worse in the sense that they can solve less than 400 out of the 432 different models.

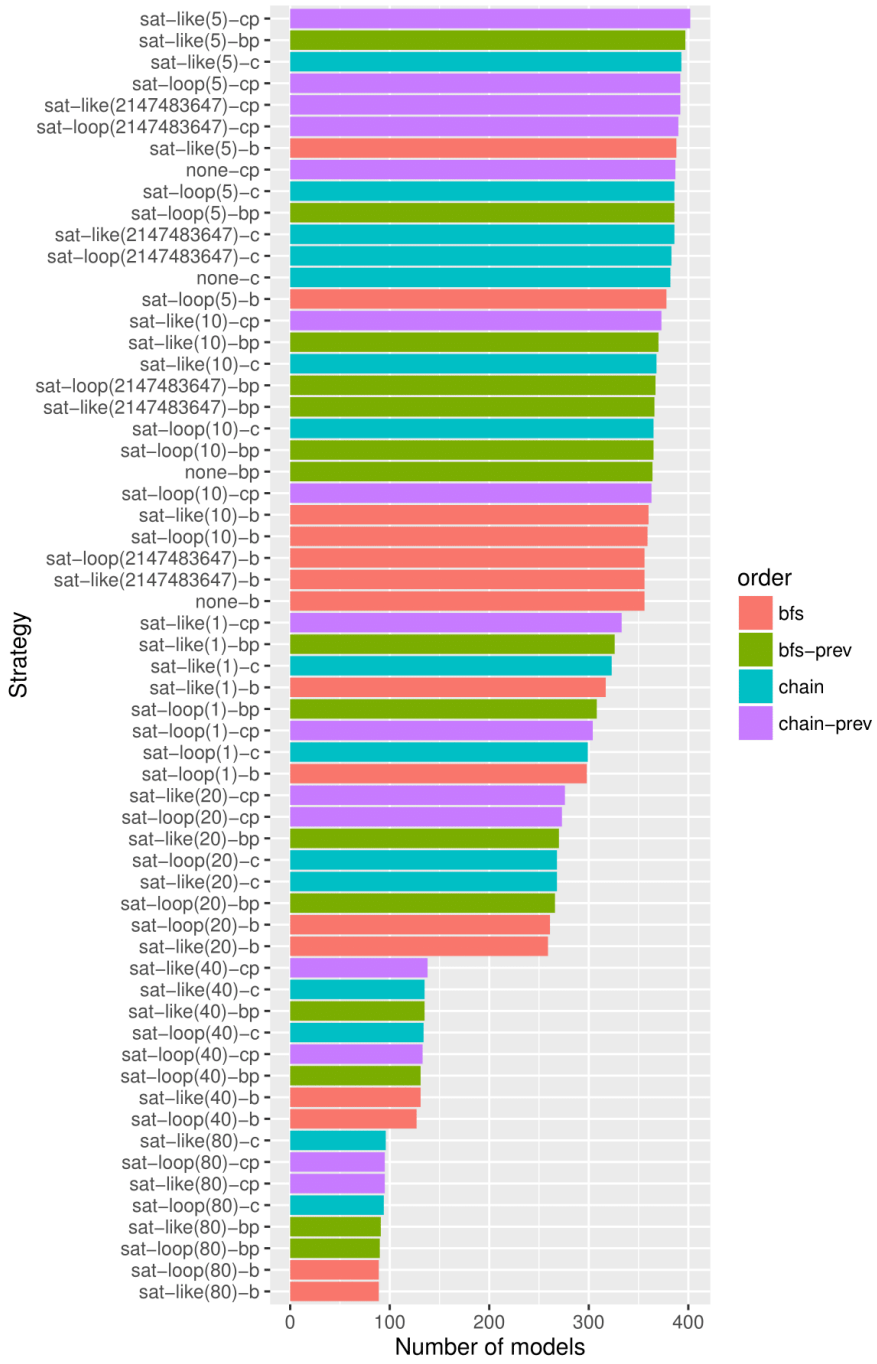

Figure 2: Capability of the 60 strategies. The strategies are listed on the vertical axis and the horizontal axis expresses the capability. The capability is measured by the total number of different models the strategy is able to solve.

It can be observed that there are four drops in the graph:

1. Strategies with a saturation granularity of 80 perform worse than any other strategy.
2. Strategies with a saturation granularity of 40 are better than any strategy with a saturation granularity of 80 but perform worse than any other strategy.
3. Strategies with a saturation granularity of 20 are better than any strategy with a saturation granularity of 40 or 80 but perform worse than any other strategy.
4. Strategies with a saturation granularity of 1 are better than any strategy with a saturation granularity of 20, 40 or 80 but perform worse than any other strategy.

The first three drops can be clarified by the state vector length. If the state vector length is smaller than either 20, 40 or 80, the strategy for a model was redefined as saturation granularity of 2147483647. The total number of occurrences of these strategies are decreased, since these strategies are not relevant to consider for some models.

The last drop can be clarified by the number of errors produced by LTSmin. If the saturation granularity is set to 1, the size of the state space calculated by LTSmin may be extremely small. These errors were removed from the data. Therefore the total number of models solved by any strategy with a saturation granularity of 1 is lower than strategies with a saturation granularity of 5, 10 or 2147483647.

So, the data shows that there is no superior strategy, since there is no strategy capable of solving all models. Moreover, for each strategy x there exist another strategy which is able to solve models which could not be solved by x. This observation suggest that a dynamic selection of a strategy is preferred over a fixed strategy for a set of unknown models.

7.4 Strategy Performance - Time

The results discussed in the previous section do not reveal how well each strategy is able to solve each model. It only shows whether the model can be solved. This section discussed how fast each strategy solved the models it is able solve.

The performance of a strategy is measured by a normalized version of the time needed to solve a model. For each model there exist a strategy which is able to solve the model as fastest. The performance of the other strategies is compared to the fastest solving time. This reveals how much worse the strategy is compared to best strategy for a model. The performance is captured by the metric *Mark*:

$$Mark = \frac{Needed\ resources}{\min\limits_{strategies} Needed\ resources}$$

The minimum value for *Mark* is 1, which indicates the performance of a strategy equals the best performance known for the given model. A high *Mark* indicates a bad performance, since *Mark* defines the factor of resources needed more than the best strategy.

The performance of the selected strategies with respect to the solving time is depicted in Figure 3. Per strategy the performance is given using a box plot. The black dots indicate the outliers. Table 12 contains some values extracted from Figure 3 for eight strategies. A complete overview of the results is listed in Appendix B.

Strategy	*Mark*		
	Minimum	**Mean**	**Maximum**
`sat-like(10)-cp`	1.00	1.72	11.62
`sat-like(1)-cp`	1.00	1.75	48.89
`sat-like(5)-cp`	1.00	1.91	123.72
`none-cp`	1.00	5.97	633.04
`sat-like(80)-b`	1.17	6.91	83.66
`none-b`	1.03	15.23	431.43
`sat-loop(2147483647)-b`	1.02	15.52	415.46
`sat-like(2147483647)-b`	1.02	15.79	431.92

Table 12: Performance of some strategies with respect to time as given in Figure 3. For each strategy the performance is measured using all *Mark*s obtained for the models the strategy is able to solve. A *Mark* reveals the performance with respect to the best strategy. The minimum, mean and maximum of all *Mark*s is listed for a strategy.

For our research the discussion is limited to three values: the minimum and maximum *Mark* given for any model per strategy and the average *Mark* given for all solvable models per strategy.

46 strategies has a minimum *Mark* of 1. That means for any of these strategies there exists at least one model which is solved as best by that strategy. The remaining 14 strategies have a low minimum mark. The highest minimum mark is 1.17. This means that any strategy is appropriate for a certain set of models.

Likewise, the maximum *Mark* reveals that all strategies have some models that could better be solved by another strategy. Each strategy had solved a certain model while there exists another strategy which is at least 11 times faster than the chosen strategy.

The mean *Mark* reveals the overall performance of a strategy. If the strategy is fixed the mean *Mark* shows how much more resources is needed than choosing the best strategy per model. The smallest mean *Mark* is 1.72 meaning that fixing a strategy is at least 1.72 times slower than selecting the best strategy depending on the model.

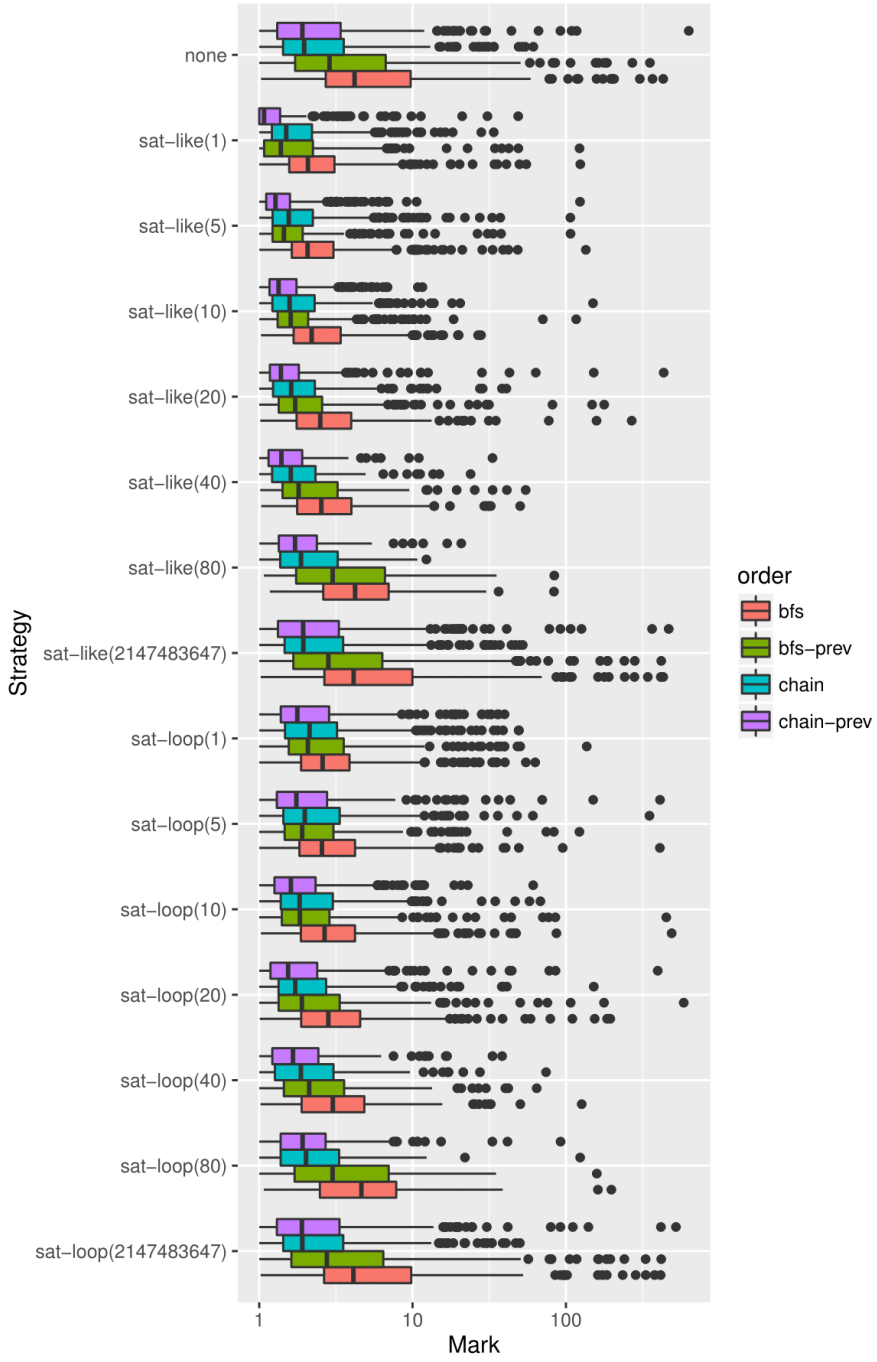

Figure 3: Performance of all 60 selected strategies with respect to time. The vertical axis list the different strategies. The performance is given by a boxplot of all *Mark*s obtained for the models the strategy is able to solve on the horizontal axis. A *Mark* reveals the performance with respect to the best strategy.

The wide range of *Mark*s show that the performance of all selected strategies varies. All strategies has some set of models for which the strategy is effective and another set of models where another strategy is highly recommended. This suggests that a dynamic approach of selecting a strategy is preferred above a fixed strategy.

7.5 Strategy Performance - Peak Size

Like the performance with respect to time, the performance of each strategy with respect to peak size was investigated. The peak size correlates to the amount of memory needed to solve a model. The performance is measured using *Mark*s as introduced in the previous section. The results are depicted in Figure 4. Table 13 contains some values extracted from Figure 4 for eight strategies. A complete overview of the results is listed in Appendix B.

Strategy	*Mark*		
	Minimum	**Mean**	**Maximum**
`sat-like(1)-c`	1.00	1.15	9.82
`sat-like(1)-cp`	1.00	1.15	9.82
`sat-like(1)-bp`	1.00	1.16	9.82
`sat-like(5)-c`	1.00	1.18	6.72
`sat-loop(20)-b`	1.00	2.66	188.21
`sat-loop(2147483647)-b`	1.00	4.09	60.26
`none-b`	1.00	4.21	60.26
`sat-like(2147483647)-b`	1.00	4.22	60.26

Table 13: Performance of some strategies with respect to peak size as given in Figure 4. For each strategy the performance is measured using all *Mark*s obtained for the models the strategy is able to solve. A *Mark* reveals the performance with respect to the best strategy. The minimum, mean and maximum of all *Mark*s is listed for a strategy.

Overall, it can be observed that there exist a few good strategies with respect to peak size. Most notable, any strategy using `sat-like` with a saturation granularity of 1 performs well. For most models the peak size is small. Considering the mean *Mark*, fixing a strategy may still suffer from a higher memory usage, but on average this can be limited to only 15% more memory.

However, there is still room for improvement. All strategies have some outliers. That means that for some models the memory usage is significantly more than needed by some other more efficient strategy. Even for the best strategies, there exists some models that may use more than 9 times as much memory as the most memory efficient strategy.

To summarize, a dynamic approach for the selection of strategy with respect to memory usage is less necessary than with respect to time. Out of the selected strategies some perform overall well. Still, selecting the best strategy per model, can improve the performance on a set of models. Each fixed strategy has a set of models is not effective on. Therefore a dynamic approach can improve the solution of models with respect to memory usage.

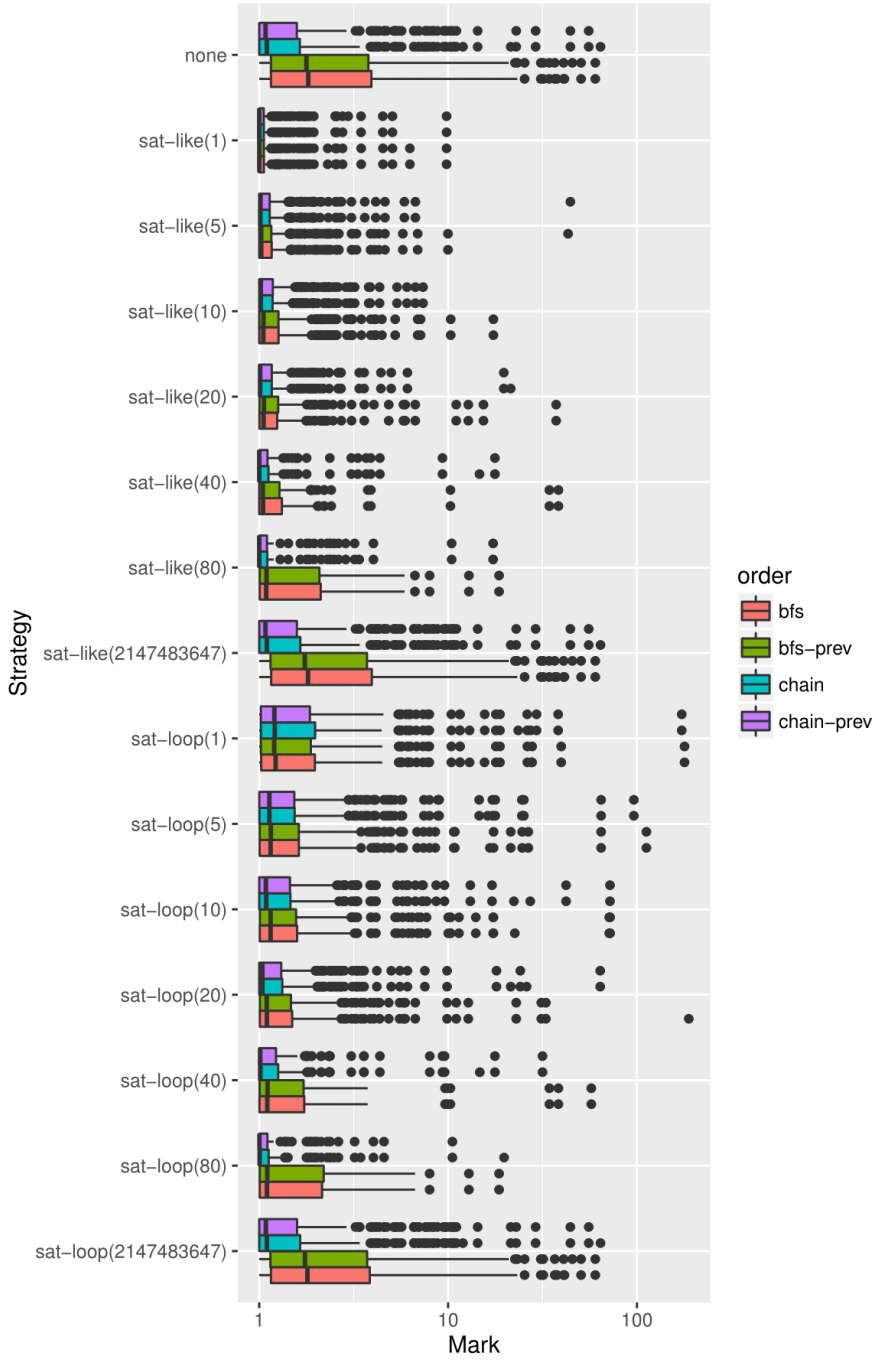

Figure 4: Performance of all 60 selected strategies with respect to peak size. The vertical axis list the different strategies. The performance is given by a boxplot of all *Mark*s obtained for the models the strategy is able to solve on the horizontal axis. A *Mark* reveals the performance with respect to the best strategy.

7.6 Conclusion

In this chapter it was discussed how the 60 selected strategies performed on 432 selected models. The data shows that no strategy is capable of solving all of the 432 selected models. For each strategy there exists a set of models which could not be solved, but are solvable by other strategies.

The performance with respect to time varies for all selected strategies. All strategies are capable of solving a set of models quickly compared to the other strategies. However, for all strategies there is also a set of models which is solved very slowly compared to the other strategies.

The performance with respect to peak size is more stable for the selected strategies. Some strategies perform overall well. Still, for each strategy there exist a set of models where the amount of used memory is significantly larger than necessary. So for each strategy there are some models which can be solved by another strategy which only uses a fraction of the required memory.

Based on the performance with respect to time and peak and the number of models solved, it can be concluded that the selection of the strategy matters. If a bad strategy is selected, the amount of time or memory may significantly larger than necessary. Furthermore, the model may be unsolvable, even though it can be solved by another strategy.

8 Classifier Evaluation

This chapter discusses the creation and evaluation of ten classifiers created to predict an appropriate strategy. The classifiers were created using SCIKIT-LEARN, which is a library for machine learning for PYTHON. Section 8.1 explains which data was used to train and evaluate the classifiers. Section 8.2 specifies which metrics were used to evaluate the results of the classifiers. Section 8.3 discusses the different algorithms available and states which algorithm was the most suitable to use for the ten classifiers. Section 8.4 and 8.5 discusses the evaluation of the classifiers using metrics and the performance data respectively. A conclusion of the results is given in section 8.6.

8.1 Training Data

The performance data collected from running tests on the computer cluster was used to train and validate classifiers. The refined data consisted of 432 different PNML and DVE models (see section 7.2). Six models could not be used as either training or validating data. SCIKIT-LEARN requires for each data point that the values of all features are known. The six models that are not used, missed at least one value for one of the eleven selected features. These models were removed from the data.

The other 426 models were used for training and validating the classifier. As mentioned in section 6.3.4 ten classifiers were trained. The classifiers had to predict the best strategy for a given model with respect to a given aspect. These aspects are elapsed time and peak size. Five classifiers were trained to predict the strategy with the lowest solving time for a given model. The other five classifiers were trained to predict the strategy with the smallest peak size for a given model.

Each classifier was trained with a fixed set of models and evaluated with the remaining models. Per aspect, five different sets were created. The training data of the one set consisted of only the PNML models and were evaluated using the DVE models. Another set was trained the other way around. The other three sets were trained with a random selection of 70% of the models. The remaining 30% of the models were used for evaluation. These three sets are mostly used during our research. The distribution of the PNML and DVE models in each set is given in Tables 14 and 15.

Set	**Training data**		**Validation data**	
	PNML models	DVE models	PNML models	DVE models
1	158	150	57	61
2	151	151	64	60
3	136	158	79	53
4	215	0	0	211
5	0	211	215	0

Table 14: Distribution of the models used for training and evaluating the five classifiers trained to predict a strategy with a low solving time.

Set	Training data		Validation data	
	PNML models	DVE models	PNML models	DVE models
1	171	137	44	74
2	148	145	67	66
3	148	136	67	75
4	215	0	0	211
5	0	211	215	0

Table 15: Distribution of the models used for training and evaluating the five classifiers trained to predict a strategy with a low peak size.

8.2 Metrics Selection

The classifiers created in our research handle multiclass classification problems. Therefore the metrics as defined in section 3.5.3 are applicable for the evaluation of the created classifiers. The metrics $Precision_M$ and $Recall_M$ are used to evaluate the average performance of classifying instances per class. $F\text{-}Measure_M$ is used to combine $Precision_M$ and $Recall_M$ to provide a single verdict. Unless specified otherwise, $F\text{-}Measure_M$ uses 1 as value for β.

$Accuracy_M$ is not used, since its value is usually high for large confusion matrices. Consider a confusion matrix of $n \times n$ with a 1 in each entry. The corresponding $Accuracy_M$ equals $\frac{\left(n^2-2n+2\right)}{n^2}$. For a large n this value is close to 1. That value is assigned to a classifier which randomly assigns classes. This classifier obviously has a poor performance, since only 1 out of n of the classifications is correct. Since 1 is the maximum value for $Accuracy_M$, it is hard to use this metric to distinguish whether the performance of the created classifier is acceptable.

Furthermore, considering the metrics using micro-averaging, only $Accuracy_\mu$ is used to evaluate a classifier. The other three metrics do not provide different values as explained in Appendix A. Therefore it is sufficient to consider only $Accuracy_\mu$.

8.3 Classification Algorithm Selection

There is a wide variety of algorithms available to create the classifiers. SCIKIT-LEARN offers different classification algorithms by providing ten predefined classifiers:

- `BernoulliNB()`
- `DecisionTreeClassifier()`
- `GaussianNB()`
- `KNeighborsClassifier()`
- `LinearSCV()`
- `MultinomialNB()`
- `NuSCV()`

- `RadiusNeighborsClassifier()`
- `SCV()`
- `SGDClassifier()`

The classifiers `BernoulliNB()`, `GaussianNB()`, `LinearSCV()`, `MultinomialNB()`, `NuSCV()` and `SGDClassifier()` could not be used, because they generated errors during training. The performance of the other classifiers is listed in Table 16. The performance equals the average performance on set 1, 2 and 3 listed in Table 14. For some classifiers multiple variants[15] were evaluated to discover the most suitable algorithm.

Classification Algorithm	$Recall_M$	$Precision_M$	$F\text{-}Measure_M$	$Accuracy_\mu$
`DecisionTreeClassifier()`	0.050	0.081	0.062	0.138
`KNeighborsClassifier( weights="uniform")`	0.066	0.113	0.083	0.142
`KNeighborsClassifier( weights="distance")`	0.039	0.065	0.049	0.093
`RadiusNeighborsClassifier( radius=4e7, weights="uniform")`	0.032	0.139	0.052	0.229
`RadiusNeighborsClassifier( radius=4e7, weights="distance")`	0.033	0.114	0.051	0.201
`SVC(kernel="rtg", decision_function_ shape=None)`	0.032	0.229	0.056	0.229
`SVC(kernel="rtg", decision_function_ shape="ovo")`	0.032	0.229	0.056	0.229
`SVC(kernel="rtg", decision_function_ shape="ovr")`	0.032	0.229	0.056	0.229
`SVC(kernel="sigmoid", decision_function_ shape=None)`	0.032	0.229	0.056	0.229
`SVC(kernel="sigmoid", decision_function_ shape="ovo")`	0.032	0.229	0.056	0.229
`SVC(kernel="sigmoid", decision_function_ shape="ovr")`	0.032	0.229	0.056	0.229

Table 16: Average performance of eleven different classification algorithms in SCIKIT-LEARN on set 1, 2 and 3 of Table 14. The performance is measured using the selected metrics for multiclass classifiers.

[15]Details of the values for the parameters of the classifiers can be found on `http://scikit-learn.org/0.17/modules/classes.html`, accessed October 2016. Other kernels than `rtg` and `sigmoid` for `SVC()` produced a time out.

Based on these results it can be observed that all variants of `SVC()` perform equally well. The `KNeighborsClassifier(weights="uniform")` is better than `DecisionTreeClassifier()` and the algorithm called `KNeighborsClassifier(weights="distance")`. `SVC()` performs better than both variants of `RadiusNeighborsClassifier()`.

So `KNeighborsClassifier(weights="uniform")` and `SVC()` are the most appropriate. The $Recall_M$ of `KNeighborsClassifier(weights="uniform")` is twice the $Recall_M$ of `SVC()`, but the $Precision_M$ is less than half of the $Precision_M$ of `SVC()`. The $Accuracy_\mu$ of `SVC()` is significantly better than the $Accuracy_\mu$ of `KNeighborsClassifier(weights="uniform")`. Therefore `SVC()` was selected as most suitable classifier for our research. Since all variants of `SVC()` perform equally well, the default `SVC()` (= `SVC(kernel="rtg",decision_function_shape=None)`) was selected.

8.4 Results - Metrics

The ten classifiers were evaluated using the selected metrics as explained in section 8.2. The classes were defined as the 60 selected strategies. The goal of each classifier is to predict the most suitable strategy with respect to either low solving time or a low peak size given the features of model. A classification is correct if the predicted strategy equals the best strategy for a model. The results of the classifiers created for the time and peak size aspect are listed in Table 17 and 18 respectively. The first row of both tables summarizes the results of set 1, 2 and 3. These values are referred as the average performance of the created classifiers.

Set	$Recall_M$	$Precision_M$	$F\text{-}Measure_M$	$Accuracy_\mu$
Average of 1, 2 and 3	0.032	0.229	0.056	0.229
1	0.032	0.271	0.058	0.271
2	0.036	0.234	0.062	0.234
3	0.027	0.182	0.047	0.182
4	0.032	0.066	0.043	0.066
5	0.024	0.088	0.038	0.088

Table 17: Performance of the classifiers trained and verified with the sets listed in Table 14. The performance is measured using the selected metrics for multiclass classifiers.

In order to put the results in perspective it is reasonable to compare the results with a random classifier. A random classifier randomly assigns a class without considering the available data. The performance of a random classifier for the given multiclass classification problem is a 60×60 confusion matrix where all entries have the value 1. Based on the definition of the metrics as given in section 3.5.3, the random classifier scores 0.017 $\left(= \frac{1}{60}\right)$ for $Recall_M$, $Precision_M$, $F\text{-}Measure_M$ and $Accuracy_\mu$.

Set	$Recall_M$	$Precision_M$	$F\text{-}Measure_M$	$Accuracy_\mu$
Average of 1, 2 and 3	0.032	0.218	0.056	0.218
1	0.030	0.195	0.052	0.195
2	0.038	0.226	0.066	0.226
3	0.028	0.232	0.050	0.232
4	0.030	0.275	0.055	0.275
5	0.024	0.121	0.041	0.121

Table 18: Performance of the classifiers trained and verified with the sets listed in Table 15. The performance is measured using the selected metrics for multiclass classifiers.

Compared to the random classifier, the trained classifiers perform significantly better. The $Precision_M$ and the $Accuracy_\mu$ is on average 13× better than the random classifier. The $Recall_M$ is on average twice as high as the $Recall_M$ of the random classifier.

However, the performance of the classifiers is not good either. An $Accuracy_\mu$ of 0.229 indicates that 77.1% of the models is misclassified. So, over three out of every four models is assigned to the wrong class. Likewise, the low values for $Recall_M$ and $Precision_M$ suggest that the models are poorly recognized by the created classifiers and are mostly assigned to the wrong class.

The classifiers perform worse if they are trained with only one type of model (the classifiers trained with set 4 or 5). The $Recall_M$ is lower and a significant drop in both $Precision_M$ and $Accuracy_\mu$ can be observed. It should be noted that the classifier with respect to peak size trained and evaluated with set 4 is an exception. The output produced by the classifier showed that for each model `sat-like(5)-cp` was predicted. As this strategy is the best strategy for a large number of DVE models, the metrics are better than the other three classifiers trained with either set 4 or 5.

8.5 Results - Appropriateness

The results presented in the previous section have one major limitation. The results are restricted to how well the classifiers are able to predict the best strategy given a model. It is not necessary to come up with the absolute best strategy of model. It is sufficient when the predicted strategy is almost as good as the best strategy for a given model. Therefore the classifiers were also evaluated by comparing the performance of the predicted strategy with the performance of the best strategy for each model.

For each model the predicted strategy was evaluated using the performance data. If the data reveals the predicted strategy is not able to solve the model a failure is recorded. Otherwise, the performance is compared with the best strategy for that model. The difference is expressed using the metric *Mark* as defined in section 7.4. The overall performance of the classifier is defined by the number of models is able to solve and the average of the *Mark*'s for the solvable models.

For each strategy the same evaluation was performed by fixing the predicted strategy. This reveals how a fixed strategy performs on the validation data of the training sets. These results were compared with the evaluation of the classifiers.

The evaluation of the classifiers and fixed strategies are listed in Table 19 with respect to time and Table 20 with respect to peak size. The values are averages of the evaluation using the validation data of set 1, 2 and 3. Both tables only show the performance of the classifiers, the three best fixed strategies and three worst fixed strategies. The complete evaluation is listed in Appendix C. The strategy `dynamic` is used to indicate the three classifiers trained with set 1, 2 and 3.

Rank	**Strategy**	**Solved models**	**Average** *Mark*
1	`sat-like(5)-cp`	97.33%	1.60
2	`sat-like(10)-cp`	96.52%	1.62
3	`sat-like(20)-cp`	96.26%	2.60
53	`dynamic`	82.89%	1.75
59	`sat-loop(1)-cp`	76.74%	3.30
60	`sat-loop(1)-c`	76.47%	3.92
61	`sat-loop(1)-b`	76.47%	4.96

Table 19: The performance of classifiers compared to the performance of a fixed strategy with respect to time. The rank is determined by the number of models in the validation data the strategy is able to solve. The average *Mark* shows how well the strategy is capable of solving the models it is able to solve.

Rank	**Strategy**	**Solved models**	**Average** *Mark*
1	`sat-like(5)-cp`	93.13%	1.20
2	`sat-like(10)-cp`	92.37%	1.30
3	`sat-like(10)-c`	92.11%	1.30
55	`dynamic`	73.28%	1.18
59	`sat-loop(1)-cp`	70.74%	3.36
60	`sat-loop(1)-b`	70.48%	3.48
61	`sat-loop(1)-bp`	69.47%	3.38

Table 20: The performance of classifiers compared to the performance of a fixed strategy with respect to peak size. The rank is determined by the number of models in the validation data the strategy is able to solve. The average *Mark* shows how well the strategy is capable of solving the models it is able to solve.

The results show that the classifiers perform worse than most fixed strategies. With respect to time, 52 different strategies are able to solve more models than the strategies predicted by the classifiers. The number of models solved is 6% higher than the worst fixed strategy, but 14% lower than the best strategy. For the 83% of the models that can be solved, the classifiers predict bad strategies; the predicted strategies need 75% more time to solve than the best strategies for

these models. This is worse than the 60% penalty of choosing a fixed strategy, which is able to solve more models.

With respect to peak size, 54 different strategies are able to solve more models than the strategies predicted by the classifiers. The number of models solved by the classifier is almost equal to the worst fixed strategy. The best fixed strategy can solve 20% more models than the strategies predicted by the classifiers. For the 73% of the models that can be solved, the classifiers predict an appropriate strategy; the predicted strategies only need 18% more memory due to the peak size than the best strategy for these models.

The evaluation of the classifiers trained with set 4 and 5 show similar results. Since these four classifiers are trained and validated with a distinct selection of models, the evaluation of these classifiers are not incorporated in the data presented in Table 19 and 20.

8.6 Conclusion

The data collected from running experiments on the cluster was used to extract data entries for 426 different models. Ten classifiers were trained using a selection of these entries with `SVC()` provided by SCIKIT-LEARN. Each classifier was evaluated using the data entries of the models it was not trained with.

The metrics for multiclass classifiers show that the trained classifiers perform better than a random classifier. However, the results are not good either. The $Accuracy_{\mu}$ of the classifiers is less than 25% indicating that more than three of out of four models is misclassified. The values of the other metrics are below 25% as well, suggesting that the classifiers are not able to predict the best strategy for a model accurately. The classifiers trained with only one type of model perform worse than the classifiers trained with a selection of both PNML and DVE models.

Since the metrics only indicate how well the classifiers are able to predict the best strategy, it was also evaluated how appropriate the predicted strategies were. The predicted strategies of the classifiers were compared with the best strategy for each model. The overall performance of the classifiers was compared to the performance of selecting a fixed strategy. Most fixed strategies perform better, because they can solve significantly more models. The predicted strategies by the classifier trained with respect to a low solving time were not accurate enough to beat a fixed strategy. Some fixed strategies exist which solve models faster, while solving more models. The strategy predicted by the classifier trained with respect to a low peak size were appropriate. The classifiers needed about 18% more memory. However, there exist fixed strategies which the same performance which are able to solve more models.

Overall, the trained classifiers did not outperform the fixed strategies. Currently, there are some strategies which perform overall better than the trained classifiers. Although it was shown that each strategy has some models it is not effective on, on average it is better to fix a strategy for a set of models than using the trained classifiers.

9 Feature Relevance

In this chapter the relevance of the eleven selected features is discussed. The prediction of a strategy by a classifier was based on a total of eleven features as listed in section 6.1.2. It is not known whether each feature contributes to the prediction of a strategy. The goal is to establish which set of features are necessary for the prediction of a strategy and which features are redundant. Training a classifier with fewer features may improve the quality of the classifier, since the redundant features can negatively influence the performance of the classifier.

Section 9.1 explains the approach taken to investigate the relevance of the eleven selected features. Section 9.2 discusses the results for the classifiers created with respect to time. Section 9.3 discusses the results for the classifiers created with respect to peak size. The results are summarized in section 9.4.

9.1 Approach

The ten classifiers created during our research were created using all eleven selected features as specified in section 6.1.2. In order to determine whether a feature is relevant, multiple classifiers were trained using different subsets of the eleven features.

These newly created classifiers were trained using the three sets consisting of a random selection of 70% of the models. The classifiers trained with respect to time were trained using set 1, 2 and 3 as listed in Table 14, while the classifiers trained with respect to peak size were trained using set 1, 2 and 3 as listed in Table 15. In this way, the performance of the created classifiers could be compared with the existing classifiers created using all eleven features.

Due to the large number of subsets[16] to be investigated, the evaluation of the classifiers was limited to the evaluation using multiclass metrics. The metrics $Recall_M$, $Precision_M$ and $Accuracy_\mu$ were used to evaluate the performance of the classifiers, since these metrics were used to evaluate the classifiers created using all eleven features (see section 8.2).

The performance of three classifiers created using a specific subset of features is averaged and compared to the average performance of the three classifiers created using all eleven features. This comparison is categorized using three groups:

Better = The performance of the classifiers using the subset is better than the performance of the classifiers using all features. At least one metric for the classifiers using the subset is higher than the corresponding metric for the classifiers using all features, while the other metrics are equal.

Equal = The performance of the classifiers using the subset is equal to the performance of the classifiers using all features. For each metric, the value

[16] There exist $2^{11} - 2 = 2046$ different proper subsets of any set of eleven elements, excluding the empty set.

for the metric for the classifiers using the subset is equal to the value for the metric for the classifiers using all features.

Worse = The performance of the classifiers using the subset is worse than the performance of the classifiers using all features. At least one metric is worse for the classifiers created using the subset than for the classifiers created using all features. In most cases, all metrics for the classifiers using the subset are less than or equal to the metrics for the classifiers using all features.

In the groups *Better* and *Equal*, each subset is either minimal or non-minimal. A subset x is minimal if there does not exist a subset of x with the same performance as the performance of classifiers created with subset x. A minimal subset contains a combination of features which are all necessary to predict a strategy, while a non-minimal subset contains at least one redundant feature.

9.2 Results - Time

For each subset the performance of the three classifiers was determined. The results are listed in Table 21 grouped by the number of features in each subset.

Number of features	*Better*	*Equal*		*Worse*
		Minimal	Non-minimal	
1	0	0	0	11
2	0	18	0	37
3	0	6	103	56
4	0	0	272	58
5	0	0	419	43
6	0	0	449	13
7	0	0	322	8
8	0	0	164	1
9	0	0	55	0
10	0	0	11	0

Table 21: Performance of the classifiers trained with subsets compared to the performance of the classifiers created using all features with respect to time. The subsets are grouped by the number of features in the set.

The last row in Table 21 indicates that each feature is redundant with respect to the ten other features. Any classifier created using only ten out of the eleven selected features performs as well as a classifier created using all eleven features. Likewise, any classifier created using only nine out of the eleven selected features performs as well as a classifier created using all eleven features.

There exist only one subset with eight features which is not sufficient; any other subset of eight features has a better performance. This subset is {*State Vector Length*, *Groups*, *Bandwidth*, *Average Wavefront*, *RMS Wavefront*, *NES*, *WES*, *NWES*}. This subset is remarkable, since it suggests that any of the features *Profile*, *Span* or *ES* is needed to predict a strategy. Furthermore, all eight subsets of this subset consisting of seven features are in the category *Worse*

too. Thirteen out of twenty-two possible subsets of this subset consisting of six features are insufficient too. The other fifteen subsets are non-minimal subset in the category *Equal*. This data confirms that redundant features may worsen the performance of a classifier.

On the other hand, it can be observed that none of the eleven features by itself is able to predict the strategy as well as multiple features. All subsets consisting of one feature perform worse than a classifier trained with all eleven features.

There exist eighteen subsets consisting of two features which is able to perform as well as the classifier created with all features. These subsets are minimal by definition. These subsets consist of the following combinations of features:

- {*State Vector Length*, *Profile*}
- {*State Vector Length*, *Span*}
- {*State Vector Length*, *ES*}
- {*Groups*, *Profile*}
- {*Groups*, *Span*}
- {*Groups*, *WES*}
- {*Bandwidth*, *Profile*}
- {*Bandwidth*, *Span*}
- {*Bandwidth*, *ES*}
- {*Bandwidth*, *WES*}
- {*Profile*, *Span*}
- {*Profile*, *Average Wavefront*}
- {*Profile*, *RMS Wavefront*}
- {*Profile*, *ES*}
- {*Profile*, *WES*}
- {*Span*, *ES*}
- {*Span*, *WES*}
- {*ES*, *WES*}

Interestingly, all but two subsets contain at least one of the features *Profile*, *Span* or *ES*. But the subsets {*Groups*, *WES*} and {*Bandwidth*, *WES*} are good subsets too, since they achieved the same performance as the classifiers created with all eleven features. The features *NES* and *NWES* do not occur in any minimal subset consisting of two features.

Considering three or more features, only six new minimal subsets did arise. All other subsets which perform as well as the classifiers using all features are non-minimal and contain redundant features. The six minimal subsets with more than two features are:

- {*State Vector Length*, *Average Wavefront*, *WES*}
- {*State Vector Length*, *RMS Wavefront*, *WES*}
- {*Groups*, *Average Wavefront*, *ES*}
- {*Groups*, *RMS Wavefront*, *ES*}
- {*Groups*, *ES*, *NES*}
- {*Groups*, *ES*, *NWES*}

So, in total 1819 different subsets exist which perform as well as the classifiers created using all eleven features. No subset exists which achieved a better performance than the classifiers created using all features. There exist 24 minimal subsets, where all eleven features are present in at least one subset. This suggest that only a few out of the eleven features are necessary to predict a strategy, but that any of the features can be used if the other features are chosen accordingly.

9.3 Results - Peak Size

For each subset the performance of the three classifiers was determined. The results are listed in Table 22 grouped by the number of features in each subset.

Number of features	*Better*		*Equal*		*Worse*
	Minimal	Non-minimal	Minimal	Non-minimal	
1	0	0	1	0	10
2	0	0	19	1	35
3	0	0	5	99	61
4	6	0	0	277	47
5	0	4	0	424	34
6	0	1	0	433	28
7	0	0	0	322	8
8	0	0	0	165	0
9	0	0	0	55	0
10	0	0	0	11	0

Table 22: Performance of the classifiers trained with subsets compared to the performance of the classifiers created using all features with respect to peak size. The subsets are grouped by the number of features in the set.

The last row in Table 22 shows that any ten of the eleven selected features is sufficient to predict a strategy, since the performance is equal to the classifiers created using all eleven features. So any eleventh feature is redundant with respect to the other ten features. Likewise, any subset consisting of eight or nine features is sufficient to train a classifier with a performance equal to the

performance of the classifiers created with all features.

Most classifiers trained with a subset of seven features did perform as well as the classifiers trained with all features. The eight subsets which worsen the performance are subsets of the subset {*State Vector Length*, *Groups*, *Bandwidth*, *Average Wavefront*, *RMS Wavefront*, *NES*, *WES*, *NWES*}. These subsets are the same subsets which worsen the performance of the classifiers trained with respect to time. Furthermore, 27 out of the 28 subsets consisting of six features with a performance worse than the classifiers created using all features are a subset of {*State Vector Length*, *Groups*, *Bandwidth*, *Average Wavefront*, *RMS Wavefront*, *NES*, *WES*, *NWES*}. So, this suggests that the features *Profile*, *Span* and *ES* are relevant for predicting a strategy which has a low peak size during state space exploration.

But the results show that there exists a single feature which is able to predict a strategy as well as a classifier created with all features. This single feature is *NES*. The other classifiers trained with only one of the ten other features perform worse than the classifiers created with all features.

Considering the subsets consisting of two features, twenty different subsets have a performance equal to the classifiers created using all eleven selected strategies. However, only one of these subsets is non-minimal, implying that combining *NES* with only one feature can maintain the performance of a classifier created by only using the feature *NES*. Creating a classifier with the feature *NES* and any one of the other nine features will worsen the performance. The nineteen minimal subsets consisting of two features are the following combinations of features:

- {*State Vector Length*, *Profile*}
- {*State Vector Length*, *Span*}
- {*State Vector Length*, *ES*}
- {*Groups*, *Profile*}
- {*Groups*, *Span*}
- {*Groups*, *WES*}
- {*Bandwidth*, *Profile*}
- {*Bandwidth*, *Span*}
- {*Bandwidth*, *ES*}
- {*Profile*, *Span*}
- {*Profile*, *Average Wavefront*}
- {*Profile*, *RMS Wavefront*}
- {*Profile*, *ES*}
- {*Profile*, *WES*}

- {*Span*, *Average Wavefront*}
- {*Span*, *RMS Wavefront*}
- {*Span*, *ES*}
- {*Span*, *WES*}
- {*ES*, *WES*}

Most of these subsets correspond to the minimal subsets consisting of two features discovered for the aspect time. Only one of these subsets did not contain the features *Profile*, *Span* or *ES*, namely subset {*Groups*, *WES*}. The feature *NWES* does not occur in any of these minimal subsets[17].

The subsets consisting of three features contain 104 different subsets which perform equally well as the classifiers created using all features. Only five of these subsets are minimal. These subsets are the following combinations of features:

- {*Bandwidth*, *Average Wavefront*, *WES*}
- {*Bandwidth*, *RMS Wavefront*, *WES*}
- {*Bandwidth*, *State Vector Length*, *WES*}
- {*Average Wavefront*, *Groups*, *ES*}
- {*RMS Wavefront*, *Groups*, *ES*}

It can be observed that the feature *NWES* is not contained in any of these minimal subsets, while the other ten features do occur in some minimal subset with three or less features.

Considering the subsets consisting of four or more features, eleven subsets create classifiers with a better performance than the classifier created by using all features. Six of these subsets are minimal. These subsets are the following combinations of features:

- {*Groups*, *Bandwidth*, *Average Wavefront*, *RMS Wavefront*}
- {*Groups*, *Bandwidth*, *Average Wavefront*, *NES*}
- {*Groups*, *Bandwidth*, *Average Wavefront*, *NWES*}
- {*Groups*, *Bandwidth*, *RMS Wavefront*, *NES*}
- {*Groups*, *Bandwidth*, *RMS Wavefront*, *NWES*}
- {*Groups*, *Bandwidth*, *NES*, *NWES*}
- {*Groups*, *Bandwidth*, *Average Wavefront*, *RMS Wavefront*, *NES*}
- {*Groups*, *Bandwidth*, *Average Wavefront*, *RMS Wavefront*, *NWES*}

[17]Neither does the feature *NES*, but the subset {*NES*} is already a smaller minimal subset in the category *Equal*.

- {*Groups*, *Bandwidth*, *Average Wavefront*, *NES*, *NWES*}
- {*Groups*, *Bandwidth*, *RMS Wavefront*, *NES*, *NWES*}
- {*Groups*, *Bandwidth*, *Average Wavefront*, *RMS Wavefront*, *NES*, *NWES*}

The performances of the classifiers created by any of these eleven subsets are equal and are listed in Table 23. Each subset contains the features *Groups* and *Bandwidth* and at least two features of the the set {*Average Wavefront*, *RMS Wavefront*, *NES*, *NWES*}. Compared to the classifiers created by all features the $Recall_M$ and $Accuracy_\mu$ are slightly higher (increase of 0.5%), but the $Precision_M$ is notably 0.19 higher. The increase in $Precision_M$ corresponds to an average increase of 19% in *Precision* per class. Unfortunately, all of these subsets perform worse than the classifiers created with all features with respect to time. So, these subsets offer a classifier which is better in predicting strategies with a low peak size, while simultaneously worsen the time needed for the strategies to solve a set of models than the classifiers created using all eleven features.

Feature set	$Recall_M$	$Precision_M$	$Accuracy_\mu$
{*Groups*, *Bandwidth*} and at least two features of set {*Average Wavefront*, *RMS Wavefront*, *NES*, *NWES*}	0.037	0.408	0.223
All eleven selected features	0.032	0.218	0.218

Table 23: Performance of the classifiers created using some subsets compared to the performance of the classifiers created using all features with respect to peak size.

Out of the 2046 different subsets, 1818 different subsets can be used to create a classifier with a performance equal to the performance of the classifiers created by using all eleven features. Only 25 of these subsets are minimal while the other subsets contain at least one redundant feature. These minimal subsets consist of one up to three different features. There are eleven different subsets which improve the performance of the classifier with respect to performance of the classifier created using all features. However, these subsets worsen the performance of the classifier created using all features with respect to time. The results show that only a few out of the eleven selected features need to be selected. Any feature can be used if the other features are selected accordingly.

9.4 Conclusion

For both aspects time and peak size three classifiers were created using a subset of the eleven selected features. All 2046 different subsets per aspect were investigated. Over 85% of the different subsets created, the classifiers created using the subset did not perform differently than the classifiers using all features. For both aspects time and peak size only 24 and 25 different minimal subsets exist, respectively. These minimal subsets consist of one up to three features. The features *Profile*, *Span* and *ES* are the most occurring features, while feature *NWES* is completely absent in these minimal subsets.

The classifiers created using a subset did not improve the classifier with respect to time. Considering peak size, eleven subsets were discovered which could improve the prediction. However these subsets did worsen the time need for the predicted strategies. Each of these subsets contains the features *Groups* and *Bandwidth* and at least two features of the the set $\{$*Average Wavefront*, *RMS Wavefront*, *NES*, *NWES*$\}$. The improvement is most notable reflected by the increase in $Precision_M$, which is almost twice as high.

So, for both aspects only a few features are needed for a classifier to predict a strategy, while adding more features is in most cases redundant or may worsen the performance of the classifier. Any of the eleven selected features can be used if the other features are chosen accordingly. Hence, the subset of the eleven selected features that is sufficient can be adjusted to the features the user is willing to calculate.

10 Conclusion And Future Work

State space methods are a popular method to perform formal verification. Due to the state space explosion problem, many different state space methods did arise to make the methods feasible for larger models. However, it is up to the user to select a strategy, namely the way the state space methods are applied, for a given model under verification. If an inappropriate strategy is chosen, the model may either be unsolvable or lead to a waste of resources. The waste of resources can be noticed by a slow computation time or excessive amount of memory usage compared to another method to solve a model.

Therefore, we want to improve state space methods by moving the responsibility of selecting a strategy from the user to the model checking tools itself. A model checking tool should determine a strategy based on the information embedded in a given model specified by features. Our research aimed to investigate to what extent it is possible to derive a suitable strategy for a given model based on the features of a model.

During the research a large number of runs were performed to collect the performance of 60 selected strategies on 784 different PNML and DVE models using LTSmin. For each model it was examined how fast each strategy was able to solve the model. Furthermore, for each model and strategy the peak size was recorded. The peak size correlates to the minimum amount of memory needed to solve a model.

The data shows that for all solvable models, none of the strategies was capable of solving all models. Furthermore, each strategy was appropriate for some set of models, while it was unsuitable for another set of models. These results suggest that the strategy selection for a model should be dynamic and no fixed strategy suitable for all models can be chosen beforehand. However, since only a selection of strategies could be examined, the results do not rule out another strategy to be superior. It is an open issue whether there exists a strategy which is better than the examined strategies for the selected models.

Based on the collected data, the best strategy per model was determined. Using machine learning techniques, several classifiers were created based on eleven different features of a model. The goal of the classifiers was to predict the best strategy using the features of a given model. The predictions of the classifiers were compared with the collected data for each model. The results show that overall the classifiers were not able to predict appropriate strategies. Selecting a fixed strategy performs overall better in terms of either time or memory needed to solve the models.

The relevance of each feature was established by creating more classifiers using a subset of the eleven selected features. The performance of these classifiers was compared to the classifiers created using all features. Based on the comparisons, only one up to three specified features are necessary for a classifier to predict a strategy. Adding more features is redundant since the added features did not improve the classifier. However none of the features is completely redundant; any of the eleven selected features can be used in combination with other specific

features to create a classifier with the performance equal to the classifier created with all features. No subset was discovered which lead to a classifier with a better performance with respect to both time and peak size.

Although the performance of the created classifiers is poor, the analysis of the relevance of each feature shows that each feature contains some information which could be exploited to derive an appropriate strategy for a model. Our research gained insight in the performance of the strategies and was a first attempt to investigate the relation between the models and their best strategies. During the research many scripts were created which could be reused to further investigate the performance of strategies and the relation between models and strategies.

Future work mainly consists of improving the ability to predict an appropriate strategy using information of the model itself. The classifiers in our research were trained with little data compared to number of classes. 300 instances in the training data is not much data for a classifier to distinguish 60 classes. More models need to be examined, such that more data becomes available for the classifiers to train with.

Another possibility to improve the classifiers is to revisit the classification algorithm used by the classifiers. A more complex method like neural networks can be more suitable for the given classification problem, leading to an improvement of the performance of the classifiers. The classification problem could also be split over multiple smaller classification problems. The main classification problem can be solved in multiple steps by these smaller classification problems. Since the classifiers for the smaller classification problems may achieve better performance, the main classification problem can be easier to solve.

Considering the features, more different features can be examined. Only two or three features were necessary for the classifier, while the remaining eight or nine features are redundant. Other features may give more or different information usable by the classifier. Furthermore, the eleven selected features were limited to be generic, such that the results were applicable for a wide array of models. Model specific features may enhance the performance of the classifiers for a selected model type.

Another possibility for further work is to examine more or other strategies. The strategies were constrained in our research, since it was not attainable to investigate more strategies. Additional strategies can be considered by adding them to the investigation or replacing poor performing strategies with new strategies.

The created classifiers are currently not implemented in a model checking tool. When the performance of a classifier achieves reasonable quality, it still has to be implemented in existing model checking tools. The main challenge is to determine the values of features efficiently. If it is computationally expensive to determine the value for a specific feature, the classifier may actually worsen the state space exploration. There exist a tradeoff between finding a better strategy and accepting a non-optimal strategy for any given model.

References

[1] Barnat, J., Brim, L., Havel, V., Havlíček, J., Kriho, J., Lenčo, M., Ročkai, P., Štill, V., and Weiser, J. DiVinE 3.0 – An Explicit-State Model Checker for Multithreaded C & C++ Programs. In *Computer Aided Verification 2013* (2013), vol. 8044 of *LNCS*, Springer, pp. 863–868.

[2] Bendisposto, J., Körner, P., Leuschel, M., Meijer, J., van de Pol, J., Treharne, H., and Whitefield, J. Symbolic Reachability Analysis of B Through ProB and LTSmin. In *Integrated Formal Methods - 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings* (2016), pp. 275–291.

[3] Blom, S., van de Pol, J., and Weber, M. LTSmin: Distributed and Symbolic Reachability. In *International Conference on Computer Aided Verification* (2010), Springer, pp. 354–359.

[4] Bollig, B., and Wegener, I. Improving the Variable Ordering of OBDDs Is NP-Complete. *IEEE Transactions on Computers 45*, 9 (1996), 993–1002.

[5] Burch, J. R., Clarke, E. M., and Long, D. E. Symbolic Model Checking with Partitioned Transistion Relations. In *VLSI* (1991), pp. 49–58.

[6] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. Symbolic Model Checking: 10^20 States and Beyond. *Information and Computation 98*, 2 (1992), 142–170.

[7] Ciardo, G., Lüttgen, G., and Siminiceanu, R. Saturation: An Efficient Iteration Strategy for Symbolic State-Space Generation. In *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings* (2001), pp. 328–342.

[8] Ciardo, G., Marmorstein, R. M., and Siminiceanu, R. The saturation algorithm for symbolic state-space exploration. *International Journal on Software Tools for Technology Transfer 8*, 1 (2006), 4–25.

[9] Cohen, J. A Coefficient of Agreement for Nominal Scales. *Educational and Psychosocial Measurement*, 20 (1960), 37–46.

[10] Domingos, P. M. MetaCost: A General Method for Making Classifiers Cost-Sensitive. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999* (1999), pp. 155–164.

[11] Duda, R. O., Hart, P. E., and Stork, D. G. Unsupervised Learning and Clustering. *Pattern Classification* (2001), 519–598.

[12] Elkan, C. The Foundations of Cost-Sensitive Learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001* (2001), pp. 973–978.

[13] Fawcett, T. An introduction to ROC analysis. *Pattern Recognition Letters 27*, 8 (2006), 861–874.

[14] Grumberg, O., Livne, S., and Markovitch, S. Learning to Order BDD Variables in Verification. *Journal of Artificial Intelligence Research 18* (2003), 83–116.

[15] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement Learning: A Survey. *CoRR cs.AI/9605103* (1996).

[16] Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., and van Dijk, T. LTSmin: High-Performance Language-Independent Model Checking. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings* (2015), pp. 692–707.

[17] Kordon, F., Garavel, H., Hillah, L. M., Hulin-Hubard, F., Chiardo, G., Hamez, A., Jezequel, L., Miner, A., Meijer, J., Paviot-Adet, E., Racordon, D., Rodriguez, C., Rohr, C., Srba, J., Thierry-Mieg, Y., Trịnh, G., and Wolf, K. Complete Results for the 2016 Edition of the Model Checking Contest. http://mcc.lip6.fr/2016/results.php, June 2016.

[18] Laarman, A., van de Pol, J., and Weber, M. Multi-Core LTSmin: Marrying Modularity and Scalability. In *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings* (2011), pp. 506–511.

[19] Lachiche, N., and Flach, P. A. Improving Accuracy and Cost of Two-class and Multi-class Probabilistic Classifiers Using ROC Curves. In *Machine Learning, Proceedings of the 20th International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA* (2003), pp. 416–423.

[20] Matthews, B. W. Comparison of the Predicted and Observed Secondary Structure of T4 Phage Lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure 405*, 2 (1975), 442–451.

[21] Meijer, J., Kant, G., Blom, S., and van de Pol, J. Read, Write and Copy Dependencies for Symbolic Model Checking. In *Hardware and Software: Verification and Testing - 10th International Haifa Verification Conference, HVC 2014, Haifa, Israel, November 18-20, 2014. Proceedings* (2014), pp. 204–219.

[22] Meijer, J., and van de Pol, J. Bandwidth and Wavefront Reduction for Static Variable Ordering in Symbolic Reachability Analysis. In *NASA Formal Methods - 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings* (2016), pp. 255–271.

[23] Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of Machine Learning*. MIT press, 2012.

[24] Parker, J. R. Rank and response combination from confusion matrix data. *Information Fusion 2*, 2 (2001), 113–120.

[25] Pelánek, R. BEEM: Benchmarks for Explicit Model Checkers. In *Model Checking Software, 14th International SPIN Workshop, Berlin, Germany, July 1-3, 2007, Proceedings* (2007), pp. 263–267.

[26] Pelánek, R. Model Classifications and Automated Verification. In *Formal Methods for Industrial Critical Systems, 12th International Workshop, FMICS 2007, Berlin, Germany, July 1-2, 2007, Revised Selected Papers* (2007), pp. 149–163.

[27] Pelánek, R. Model Classifications and Automated Verification. In *Formal Methods for Industrial Critical Systems, 12th International Workshop, FMICS 2007, Berlin, Germany, July 1-2, 2007, Revised Selected Papers* (2007), pp. 149–163.

[28] Pelánek, R., and Rosecký, V. EMMA: Explicit Model Checking Manager (Tool Presentation). In *Model Checking Software, 16th International SPIN Workshop, Grenoble, France, June 26-28, 2009. Proceedings* (2009), pp. 169–173.

[29] Pelánek, R., and Šimecek, P. Estimating State Space Parameters. In *Proceedings of the 7th International Workshop on Parallel and Distributed Methods in Verification* (2008).

[30] Roig, O., Cortadella, J., and Pastor, E. Verification of Asynchronous Circuits by BDD-based Model Checking of Petri Nets. In *Application and Theory of Petri Nets 1995, 16th International Conference, Turin, Italy, June 26-30, 1995, Proceedings* (1995), pp. 374–391.

[31] Rudell, R. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993, Santa Clara, California, USA, November 7-11, 1993* (1993), pp. 42–47.

[32] Sahoo, D., Jain, J., Iyer, S. K., Dill, D. L., and Emerson, E. A. Predictive Reachability Using a Sample-Based Approach. In *Correct Hardware Design and Verification Methods, 13th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2005, Saarbrücken, Germany, October 3-6, 2005, Proceedings* (2005), pp. 388–392.

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

[34] Siaw, T. L. Saturation for LTSmin. Master's thesis, University of Twente, 2012.

[35] Siminiceanu, R., and Ciardo, G. New Metrics for Static Variable Ordering in Decision Diagrams. In *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings* (2006), pp. 90–104.

[36] Sokolova, M., and Lapalme, G. Performance Measures in Classification of Human Communications. In *Advances in Artificial Intelligence, 20th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2007, Montreal, Canada, May 28-30, 2007, Proceedings* (2007), pp. 159–170.

[37] Sokolova, M., and Lapalme, G. A systematic analysis of performance measures for classification tasks. *Inf. Process. Manage. 45*, 4 (2009), 427–437.

[38] Solé, M., and Pastor, E. Traversal Techniques for Concurrent Systems. In *Formal Methods in Computer-Aided Design, 4th International Conference, FMCAD 2002, Portland, OR, USA, November 6-8, 2002, Proceedings* (2002), pp. 220–237.

[39] Stehman, S. V. Selecting and Interpreting Measures of Thematic Classification Accuracy. *Remote Sensing of Environment 62*, 1 (1997), 77–89.

[40] Tsoumakas, G., Katakis, I., and Vlahavas, I. Mining Multi-label Data. In *Data Mining and Knowledge Discovery Handbook*. Springer, 2009, pp. 667–685.

[41] Valmari, A. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996* (1996), pp. 429–528.

[42] van der Berg, F., and Laarman, A. SpinS: Extending LTSmin with Promela through SpinJa. *Electronic Notes in Theoretical Computer Science 296* (2013), 95–105.

[43] Van Rijsbergen, C. Information Retrieval: Theory and Practice. In *Proceedings of the Joint IBM/University of Newcastle upon Tyne Seminar on Data Base Systems* (1979), pp. 1–14.

[44] Yang, Y. An Evaluation of Statistical Approaches to Text Categorization. *Information retrieval 1*, 1-2 (1999), 69–90.

[45] Youden, W. J. Index for Rating Diagnostic Tests. *Cancer 3*, 1 (1950), 32–35.

List of Figures

List of Tables

A Metrics Using Micro-averaging

This appendix shows that the metrics $Recall_\mu$, $Precision_\mu$, $F\text{-}Measure_\mu$ and $Accuracy_\mu$ define the same metric for any fixed confusion matrix. Section A.1 defines the functions and symbols used in this appendix. Sections A.2 and A.3 show that $Recall_\mu$, $Precision_\mu$ and $F\text{-}Measure_\mu$ define the same metric. Section A.4 shows $Accuracy_\mu$ is a proper replacement of $Recall_\mu$, $Precision_\mu$ or $F\text{-}Measure_\mu$. This appendix concludes with section A.5 which discusses some edge cases about the proofs in sections A.2, A.3 and A.4.

A.1 Definitions

$\mathbb{N}$ = The set of natural numbers including 0.

$\mathbb{R}$ = The set of real numbers. Greek lower-case characters are used to indicate members of this set.

A = A confusion matrix. $A \in \mathbb{N}^{n \times n}$ where $n \in \{s \in \mathbb{N} \mid s \geq 2\}$.

A_{ij} = The value of the entry in row i and column j in A.

The following functions are defined on A:

$$TP_i(A) = A_{ii}$$

$$FP_i(A) = \sum_{k=1}^{i-1} A_{ki} + \sum_{k=i+1}^{n} A_{ki}$$

$$FN_i(A) = \sum_{k=1}^{i-1} A_{ik} + \sum_{k=i+1}^{n} A_{ik}$$

$$Recall_\mu(A) = \frac{\sum_{i=1}^{n} TP_i(A)}{\sum_{i=1}^{n} (TP_i(A) + FN_i(A))}$$

$$Precision_\mu(A) = \frac{\sum_{i=1}^{n} TP_i(A)}{\sum_{i=1}^{n} (TP_i(A) + FP_i(A))}$$

$$F\text{-}Measure_\mu(A, \beta) = \frac{(\beta^2 + 1) \cdot Precision_\mu(A) \cdot Recall_\mu(A)}{\beta^2 \cdot Precision_\mu(A) + Recall_\mu(A)}$$

$$Accuracy_\mu(A) = \frac{\sum_{i=1}^{n} A_{ii}}{\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}}$$

The function parameters are omitted if they are known in the given context.

A.2 $Recall_\mu$ equals $Precision_\mu$

In order to show that $Recall_\mu$ equals $Precision_\mu$ it is sufficient to show that $\sum_{i=1}^{n}(TP_i + FN_i)$ equals $\sum_{i=1}^{n}(TP_i + FP_i)$ for any confusion matrix A.

Proof:

$$\sum_{i=1}^{n}(TP_i + FN_i) = \sum_{i=1}^{n}\left(A_{ii} + \sum_{k=1}^{i-1}A_{ik} + \sum_{k=i+1}^{n}A_{ik}\right) = \sum_{i=1}^{n}\sum_{k=1}^{n}A_{ik} = \sum_{k=1}^{n}\sum_{i=1}^{n}A_{ik} =$$
$$\sum_{k=1}^{n}\left(A_{kk} + \sum_{i=1}^{k-1}A_{ik} + \sum_{i=k+1}^{n}A_{ik}\right) = \sum_{k=1}^{n}(TP_k + FP_k) = \sum_{i=1}^{n}(TP_i + FP_i)$$

Since $\sum_{i=1}^{n}(TP_i + FN_i)$ equals $\sum_{i=1}^{n}(TP_i + FP_i)$, $Recall_\mu$ equals $Precision_\mu$ for any confusion matrix A by definition.

A.3 $F\text{-}Measure_\mu$ equals $Recall_\mu$

Given that $Precision_\mu$ equals $Recall_\mu$ by the proof of section A.2, $F\text{-}Measure_\mu$ equals $Recall_\mu$ for any weight β by definition.

Proof:

$$F\text{-}Measure_\mu = \frac{(\beta^2+1)\cdot Precision_\mu \cdot Recall_\mu}{\beta^2 \cdot Precision_\mu + Recall_\mu} = \frac{(\beta^2+1)\cdot Recall_\mu \cdot Recall_\mu}{\beta^2 \cdot Recall_\mu + Recall_\mu} =$$
$$\frac{(\beta^2+1)\cdot Recall_\mu \cdot Recall_\mu}{(\beta^2+1)\cdot Recall_\mu} = Recall_\mu$$

A.4 $Accuracy_\mu$ equals $Recall_\mu$

The previous two sections show that $Recall_\mu$, $Precision_\mu$ and $F\text{-}Measure_\mu$ define the same metric for any confusion matrix A. This section shows that $Accuracy_\mu$ equals $Recall_\mu$. Therefore, $Accuracy_\mu$ can be used instead of $Recall_\mu$, $Precision_\mu$ or $F\text{-}Measure_\mu$.

Proof:

1) $$\sum_{i=1}^{n}A_{ii} = \sum_{i=1}^{n}TP_i$$

2) $$\sum_{i=1}^{n}\sum_{j=1}^{n}A_{ij} = \sum_{i=1}^{n}\left(A_{ii} + \sum_{j=1}^{i-1}A_{ij} + \sum_{j=i+1}^{n}A_{ij}\right) = \sum_{i=1}^{n}(TP_i + FN_i)$$

Combining 1 and 2 gives:

$$Accuracy_\mu = \frac{\sum_{i=1}^{n}A_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n}A_{ij}} = \frac{\sum_{i=1}^{n}TP_i}{\sum_{i=1}^{n}(TP_i + FN_i)} = Recall_\mu$$

A.5 Remarks

$Recall_\mu$, $Precision_\mu$, $F\text{-}Measure_\mu$ and $Accuracy_\mu$ are only well defined when the denominator is non-zero. Therefore, it is assumed that there exists a row i and a column j in A such that $A_{ij} > 0$. This assumption is no restriction in practice, because the validation of a classifier is performed using a non-empty test set.

$F\text{-}Measure_\mu$ is undefined when $Recall_\mu$ equals 0. However, it is arguable that $F\text{-}Measure_\mu$ equals 0 when $Recall_\mu$ equals 0, according to the following one-sided limit:

$$\lim_{x \to 0^+} \frac{\alpha \cdot x^2}{\alpha \cdot x} = 0 \text{ for any constant } \alpha \geq 1.$$

B Strategy Evaluation - Data

This Appendix contains a complete overview of the evaluation of the 60 selected strategies. The data presented in this Appendix lists the data presented in the Figures of chapter 7. The naming convention of section 7.1 is used to name the strategies.

B.1 Capability

The data presented in Figure 2 is listed in Table 24. The second column shows the capability per strategy by listing the number of models the strategy is able to solve. Since the capability was determined using 432 different models, the capability can be expressed as percentage. This percentage is listed in the last column.

Strategy	Number of models	Coverage
none-b	356	82.41%
none-bp	364	84.26%
none-c	382	88.43%
none-cp	387	89.58%
sat-like(1)-b	317	73.38%
sat-like(1)-bp	326	75.46%
sat-like(1)-c	323	74.77%
sat-like(1)-cp	333	77.08%
sat-like(5)-b	388	89.81%
sat-like(5)-bp	397	91.90%
sat-like(5)-c	393	90.97%
sat-like(5)-cp	402	93.06%
sat-like(10)-b	360	83.33%
sat-like(10)-bp	370	85.65%
sat-like(10)-c	368	85.19%
sat-like(10)-cp	373	86.34%
sat-like(20)-b	259	59.95%
sat-like(20)-bp	270	62.50%
sat-like(20)-c	268	62.04%
sat-like(20)-cp	276	63.89%
sat-like(40)-b	131	30.32%
sat-like(40)-bp	135	31.25%
sat-like(40)-c	135	31.25%
sat-like(40)-cp	138	31.94%
sat-like(80)-b	89	20.60%
sat-like(80)-bp	91	21.06%
sat-like(80)-c	96	22.22%
sat-like(80)-cp	95	21.99%
sat-like(2147483647)-b	356	82.41%
sat-like(2147483647)-bp	366	84.72%
sat-like(2147483647)-c	386	89.35%

continued on next page

continued from previous page

Strategy	Number of models	Coverage
sat-like(2147483647)-cp	392	90.74%
sat-loop(1)-b	298	68.98%
sat-loop(1)-bp	308	71.30%
sat-loop(1)-c	299	69.21%
sat-loop(1)-cp	304	70.37%
sat-loop(5)-b	378	87.50%
sat-loop(5)-bp	386	89.35%
sat-loop(5)-c	386	89.35%
sat-loop(5)-cp	392	90.74%
sat-loop(10)-b	359	83.10%
sat-loop(10)-bp	365	84.49%
sat-loop(10)-c	365	84.49%
sat-loop(10)-cp	363	84.03%
sat-loop(20)-b	261	60.42%
sat-loop(20)-bp	266	61.57%
sat-loop(20)-c	268	62.04%
sat-loop(20)-cp	273	63.19%
sat-loop(40)-b	127	29.40%
sat-loop(40)-bp	131	30.32%
sat-loop(40)-c	134	31.02%
sat-loop(40)-cp	133	30.79%
sat-loop(80)-b	89	20.60%
sat-loop(80)-bp	90	20.83%
sat-loop(80)-c	94	21.76%
sat-loop(80)-cp	95	21.99%
sat-loop(2147483647)-b	356	82.41%
sat-loop(2147483647)-bp	367	84.95%
sat-loop(2147483647)-c	383	88.66%
sat-loop(2147483647)-cp	390	90.28%

Table 24: Capability of all strategies in Figure . The capability is measured by the number of different models the strategy is able to solve. The coverage shows which fraction that number is of the total set of 432 models.

B.2 Performance - Time

The data presented in Figure 3 is listed in Table 25 and 26. The performance of a strategy on a given model was measured using the metric *Mark* as defined in section 7.4. The overall performance of each strategy is captured by a box plot of all *Mark*'s of the models the strategy is able to solve. The minimum and maximum *Mark*'s are listed in Table 25, while the lower quantile, median and upper quantile are listed in Table 26. The average *Mark* per strategy is included in Table 25 too.

Strategy	*Mark*		
	Minimum	**Mean**	**Maximum**
none-b	1.0286	15.2278	431.4320
none-bp	1.0000	10.3107	353.7416
none-c	1.0000	4.0767	61.3941
none-cp	1.0000	5.9689	633.0379
sat-like(1)-b	1.0000	3.9811	124.1853
sat-like(1)-bp	1.0000	2.9263	122.6956
sat-like(1)-c	1.0000	2.4238	33.8622
sat-like(1)-cp	1.0000	1.7523	48.8889
sat-like(5)-b	1.0000	3.6983	135.0180
sat-like(5)-bp	1.0000	2.3929	107.2963
sat-like(5)-c	1.0000	2.7802	107.0290
sat-like(5)-cp	1.0000	1.9113	123.7178
sat-like(10)-b	1.0262	3.2637	27.9904
sat-like(10)-bp	1.0000	2.5878	116.7907
sat-like(10)-c	1.0000	2.6675	150.4832
sat-like(10)-cp	1.0000	1.7226	11.6161
sat-like(20)-b	1.0150	5.6562	268.5196
sat-like(20)-bp	1.0000	4.2845	177.6538
sat-like(20)-c	1.0000	2.6672	40.9179
sat-like(20)-cp	1.0000	4.4045	434.0941
sat-like(40)-b	1.0301	4.8041	50.3332
sat-like(40)-bp	1.0145	3.7359	54.7583
sat-like(40)-c	1.0000	2.4490	23.9408
sat-like(40)-cp	1.0000	2.0892	33.1948
sat-like(80)-b	1.1727	6.9124	83.6568
sat-like(80)-bp	1.0711	5.8600	84.1744
sat-like(80)-c	1.0000	2.7130	12.3181
sat-like(80)-cp	1.0016	2.6949	20.8032
sat-like(2147483647)-b	1.0224	15.7867	431.9200
sat-like(2147483647)-bp	1.0000	10.9053	419.2246
sat-like(2147483647)-c	1.0000	4.1075	52.3609
sat-like(2147483647)-cp	1.0000	6.5406	468.2955
sat-loop(1)-b	1.0000	4.8161	63.0946
sat-loop(1)-bp	1.0000	4.6865	136.7755
sat-loop(1)-c	1.0000	3.9302	49.4756
sat-loop(1)-cp	1.0000	3.6982	39.8693
sat-loop(5)-b	1.0000	5.3439	410.2941
sat-loop(5)-bp	1.0000	3.6643	122.4231
sat-loop(5)-c	1.0000	4.2718	351.3293
sat-loop(5)-cp	1.0000	4.4569	411.0274
sat-loop(10)-b	1.0251	6.1133	490.1125
sat-loop(10)-bp	1.0000	4.7604	451.7672
sat-loop(10)-c	1.0000	3.2983	68.2742
sat-loop(10)-cp	1.0000	2.5276	61.2289
sat-loop(20)-b	1.0080	7.2600	194.9179

continued on next page

continued from previous page

Strategy	*Mark*		
	Minimum	**Mean**	**Maximum**
sat-loop(20)-bp	1.0000	7.1365	587.6108
sat-loop(20)-c	1.0000	3.7107	152.0828
sat-loop(20)-cp	1.0000	4.7316	397.5839
sat-loop(40)-b	1.0193	6.3012	126.7630
sat-loop(40)-bp	1.0000	4.7321	64.6127
sat-loop(40)-c	1.0000	3.6433	74.2890
sat-loop(40)-cp	1.0000	3.0368	38.3892
sat-loop(80)-b	1.0721	10.1815	198.0802
sat-loop(80)-bp	1.0000	7.4435	159.1434
sat-loop(80)-c	1.0000	4.1933	123.8836
sat-loop(80)-cp	1.0000	4.3879	92.3649
sat-loop(2147483647)-b	1.0218	15.5246	415.4632
sat-loop(2147483647)-bp	1.0000	11.4407	418.6881
sat-loop(2147483647)-c	1.0000	3.9582	50.0984
sat-loop(2147483647)-cp	1.0000	6.7536	522.5205

Table 25: Performance of all 60 selected strategies with respect to time as given in Figure 3. For each strategy the performance is measured using all *Mark*s obtained for the models the strategy is able to solve. A *Mark* reveals the performance with respect to the best strategy. The minimum, mean and maximum of all *Mark*s is listed for a strategy.

Strategy	*Mark*		
	Lower quantile	**Median**	**Upper quantile**
none-b	2.7140	4.1836	9.7182
none-bp	1.7141	2.8732	6.6788
none-c	1.4295	1.9630	3.5579
none-cp	1.3104	1.9115	3.3948
sat-like(1)-b	1.5697	2.0776	3.0952
sat-like(1)-bp	1.0766	1.3820	2.2392
sat-like(1)-c	1.2087	1.4997	2.2081
sat-like(1)-cp	1.0000	1.0756	1.3687
sat-like(5)-b	1.6312	2.0681	3.0470
sat-like(5)-bp	1.2216	1.4490	1.9200
sat-like(5)-c	1.2222	1.5593	2.2359
sat-like(5)-cp	1.1104	1.2763	1.5863
sat-like(10)-b	1.6758	2.1970	3.3984
sat-like(10)-bp	1.3178	1.6043	2.0850

continued on next page

continued from previous page

Strategy	*Mark*		
	Lower quantile	**Median**	**Upper quantile**
sat-like(10)-c	1.2191	1.5761	2.2985
sat-like(10)-cp	1.1667	1.3372	1.7500
sat-like(20)-b	1.7528	2.5000	3.9796
sat-like(20)-bp	1.3400	1.7162	2.5699
sat-like(20)-c	1.2305	1.6151	2.3056
sat-like(20)-cp	1.1733	1.3846	1.8125
sat-like(40)-b	1.7677	2.5405	3.9902
sat-like(40)-bp	1.4191	1.8069	3.2438
sat-like(40)-c	1.2128	1.6068	2.3279
sat-like(40)-cp	1.1508	1.3955	1.9070
sat-like(80)-b	2.6143	4.2178	6.9800
sat-like(80)-bp	1.7407	3.0107	6.6022
sat-like(80)-c	1.3705	1.8726	3.2518
sat-like(80)-cp	1.3417	1.7121	2.3758
sat-like(2147483647)-b	2.6602	4.1263	9.9808
sat-like(2147483647)-bp	1.6686	2.8233	6.3492
sat-like(2147483647)-c	1.4633	1.9379	3.5267
sat-like(2147483647)-cp	1.3244	1.9337	3.3057
sat-loop(1)-b	1.8748	2.5896	3.8770
sat-loop(1)-bp	1.5584	2.0753	3.5559
sat-loop(1)-c	1.4726	2.1278	3.2160
sat-loop(1)-cp	1.3823	1.7729	2.8544
sat-loop(5)-b	1.8284	2.5643	4.2145
sat-loop(5)-bp	1.4646	1.9050	3.0472
sat-loop(5)-c	1.4361	1.9806	3.3473
sat-loop(5)-cp	1.3026	1.7466	2.7697
sat-loop(10)-b	1.8687	2.6667	4.2137
sat-loop(10)-bp	1.4032	1.8418	2.8710
sat-loop(10)-c	1.3810	1.8333	3.0194
sat-loop(10)-cp	1.2573	1.6085	2.3303
sat-loop(20)-b	1.8776	2.8228	4.5519
sat-loop(20)-bp	1.3377	1.8982	3.3472
sat-loop(20)-c	1.3369	1.7164	2.7293
sat-loop(20)-cp	1.1837	1.5427	2.3889
sat-loop(40)-b	1.8904	3.0188	4.8448
sat-loop(40)-bp	1.4471	2.1167	3.5776
sat-loop(40)-c	1.2657	1.8696	3.0492
sat-loop(40)-cp	1.2174	1.6596	2.4310
sat-loop(80)-b	2.4859	4.6372	7.8213
sat-loop(80)-bp	1.6998	3.0085	6.9860
sat-loop(80)-c	1.3797	2.0204	3.3196
sat-loop(80)-cp	1.3792	1.9173	2.7093
sat-loop(2147483647)-b	2.6479	4.1122	9.7981
sat-loop(2147483647)-bp	1.6207	2.7616	6.4427

continued on next page

continued from previous page

Strategy	*Mark*		
	Lower quantile	**Median**	**Upper quantile**
`sat-loop(2147483647)-c`	1.4333	1.9093	3.5329
`sat-loop(2147483647)-cp`	1.3027	1.8982	3.3391

Table 26: Performance of all 60 selected strategies with respect to time as given in Figure 3. For each strategy the performance is measured using all *Mark*s obtained for the models the strategy is able to solve. A *Mark* reveals the performance with respect to the best strategy. The distribution of all *Mark*s for a strategy is given by the lower quantile, median and upper quantile.

B.3 Performance - Peak Size

The data presented in Figure 4 is listed in Table 27 and 28. The performance of a strategy on a given model was measured using the metric *Mark* as defined in section 7.4. The overall performance of each strategy is captured by a box plot of all *Mark*'s of the models the strategy is able to solve. The minimum and maximum *Mark*'s are listed in Table 27, while the lower quantile, median and upper quantile are listed in Table 28. The average *Mark* per strategy is included in Table 27 too.

Strategy	*Mark*		
	Minimum	**Mean**	**Maximum**
`none-b`	1.0000	4.2103	60.2591
`none-bp`	1.0000	4.0509	60.2591
`none-c`	1.0000	2.4624	64.1746
`none-cp`	1.0000	2.1378	55.6508
`sat-like(1)-b`	1.0000	1.1690	9.8224
`sat-like(1)-bp`	1.0000	1.1637	9.8224
`sat-like(1)-c`	1.0000	1.1469	9.8224
`sat-like(1)-cp`	1.0000	1.1505	9.8224
`sat-like(5)-b`	1.0000	1.2540	10.0000
`sat-like(5)-bp`	1.0000	1.3568	43.2354
`sat-like(5)-c`	1.0000	1.1826	6.7182
`sat-like(5)-cp`	1.0000	1.2956	44.4608
`sat-like(10)-b`	1.0000	1.3667	17.4000
`sat-like(10)-bp`	1.0000	1.3647	17.4000
`sat-like(10)-c`	1.0000	1.2514	7.3783
`sat-like(10)-cp`	1.0000	1.2522	7.3783
`sat-like(20)-b`	1.0000	1.5684	37.3982
`sat-like(20)-bp`	1.0000	1.5700	37.3982
`sat-like(20)-c`	1.0000	1.3655	21.4705

continued on next page

continued from previous page

Strategy	*Mark*		
	Minimum	**Mean**	**Maximum**
sat-like(20)-cp	1.0000	1.2843	19.7387
sat-like(40)-b	1.0000	1.9021	38.4403
sat-like(40)-bp	1.0000	1.8753	38.4403
sat-like(40)-c	1.0000	1.4898	17.7217
sat-like(40)-cp	1.0000	1.3701	17.7217
sat-like(80)-b	1.0000	2.0543	18.6393
sat-like(80)-bp	1.0000	2.0445	18.6393
sat-like(80)-c	1.0000	1.5332	17.3200
sat-like(80)-cp	1.0000	1.5022	17.3200
sat-like(2147483647)-b	1.0000	4.2183	60.2591
sat-like(2147483647)-bp	1.0000	3.9963	60.2591
sat-like(2147483647)-c	1.0000	2.4474	64.1746
sat-like(2147483647)-cp	1.0000	2.1574	55.6508
sat-loop(1)-b	1.0000	3.0514	178.6154
sat-loop(1)-bp	1.0000	2.8520	178.6154
sat-loop(1)-c	1.0000	3.0957	172.9239
sat-loop(1)-cp	1.0000	2.8532	172.9239
sat-loop(5)-b	1.0000	2.3568	112.3564
sat-loop(5)-bp	1.0000	2.3021	112.3564
sat-loop(5)-c	1.0000	2.2499	96.3754
sat-loop(5)-cp	1.0000	2.2059	96.3754
sat-loop(10)-b	1.0000	2.1906	72.1202
sat-loop(10)-bp	1.0000	2.1086	72.1202
sat-loop(10)-c	1.0000	2.0367	72.0777
sat-loop(10)-cp	1.0000	1.8830	72.0777
sat-loop(20)-b	1.0000	2.6648	188.2146
sat-loop(20)-bp	1.0000	1.8913	33.0019
sat-loop(20)-c	1.0000	1.9642	63.9347
sat-loop(20)-cp	1.0000	1.7809	63.9347
sat-loop(40)-b	1.0000	2.6322	57.4175
sat-loop(40)-bp	1.0000	2.5789	57.4175
sat-loop(40)-c	1.0000	1.9296	31.6617
sat-loop(40)-cp	1.0000	1.8197	31.6617
sat-loop(80)-b	1.0000	2.1488	18.6272
sat-loop(80)-bp	1.0000	2.1467	18.6272
sat-loop(80)-c	1.0000	1.6105	19.8324
sat-loop(80)-cp	1.0000	1.3656	10.5441
sat-loop(2147483647)-b	1.0000	4.0874	60.2591
sat-loop(2147483647)-bp	1.0000	3.9551	60.2591
sat-loop(2147483647)-c	1.0000	2.4502	64.1746
sat-loop(2147483647)-cp	1.0000	2.2033	55.6508

Table 27: Performance of all 60 selected strategies with respect to peak size as given in Figure 4. For each strategy the performance is measured using all *Mark*s obtained for the models the strategy is able to solve. A *Mark* reveals the performance with respect to the best strategy. The minimum, mean and maximum of all *Mark*s is listed for a strategy.

Strategy	*Mark*		
	Lower quantile	**Median**	**Upper quantile**
none-b	1.1531	1.8180	3.9258
none-bp	1.1531	1.7783	3.7831
none-c	1.0000	1.0972	1.6407
none-cp	1.0000	1.0804	1.5816
sat-like(1)-b	1.0000	1.0008	1.0585
sat-like(1)-bp	1.0000	1.0008	1.0578
sat-like(1)-c	1.0000	1.0000	1.0578
sat-like(1)-cp	1.0000	1.0000	1.0566
sat-like(5)-b	1.0000	1.0244	1.1636
sat-like(5)-bp	1.0000	1.0242	1.1591
sat-like(5)-c	1.0000	1.0143	1.1363
sat-like(5)-cp	1.0000	1.0160	1.1360
sat-like(10)-b	1.0000	1.0508	1.2656
sat-like(10)-bp	1.0001	1.0532	1.2639
sat-like(10)-c	1.0000	1.0194	1.1766
sat-like(10)-cp	1.0000	1.0195	1.1802
sat-like(20)-b	1.0022	1.0544	1.2452
sat-like(20)-bp	1.0026	1.0586	1.2603
sat-like(20)-c	1.0000	1.0116	1.1650
sat-like(20)-cp	1.0000	1.0120	1.1650
sat-like(40)-b	1.0000	1.0482	1.3181
sat-like(40)-bp	1.0000	1.0440	1.2797
sat-like(40)-c	1.0000	1.0001	1.1190
sat-like(40)-cp	1.0000	1.0000	1.1060
sat-like(80)-b	1.0053	1.0888	2.1181
sat-like(80)-bp	1.0054	1.0929	2.0845
sat-like(80)-c	1.0000	1.0000	1.1035
sat-like(80)-cp	1.0000	1.0000	1.0994
sat-like(2147483647)-b	1.1564	1.8128	3.9471
sat-like(2147483647)-bp	1.1475	1.7426	3.7201
sat-like(2147483647)-c	1.0000	1.0972	1.6504
sat-like(2147483647)-cp	1.0000	1.0788	1.5826
sat-loop(1)-b	1.0244	1.2212	1.9709
sat-loop(1)-bp	1.0194	1.2002	1.8759
sat-loop(1)-c	1.0195	1.2084	1.9771
sat-loop(1)-cp	1.0190	1.2025	1.8549
sat-loop(5)-b	1.0048	1.1486	1.6215
sat-loop(5)-bp	1.0053	1.1521	1.6215
sat-loop(5)-c	1.0022	1.1324	1.5395
sat-loop(5)-cp	1.0009	1.1317	1.5338
sat-loop(10)-b	1.0068	1.1486	1.5873
sat-loop(10)-bp	1.0081	1.1486	1.5680
sat-loop(10)-c	1.0000	1.0896	1.4633

continued on next page

continued from previous page

Strategy	*Mark*		
	Lower quantile	**Median**	**Upper quantile**
`sat-loop(10)-cp`	1.0000	1.0841	1.4545
`sat-loop(20)-b`	1.0073	1.0978	1.4942
`sat-loop(20)-bp`	1.0083	1.0952	1.4724
`sat-loop(20)-c`	1.0000	1.0388	1.3246
`sat-loop(20)-cp`	1.0000	1.0388	1.3076
`sat-loop(40)-b`	1.0027	1.1030	1.7313
`sat-loop(40)-bp`	1.0043	1.1063	1.7180
`sat-loop(40)-c`	1.0000	1.0143	1.2603
`sat-loop(40)-cp`	1.0000	1.0123	1.2273
`sat-loop(80)-b`	1.0051	1.0984	2.1518
`sat-loop(80)-bp`	1.0054	1.1026	2.1948
`sat-loop(80)-c`	1.0000	1.0050	1.1221
`sat-loop(80)-cp`	1.0000	1.0027	1.1049
`sat-loop(2147483647)-b`	1.1537	1.8001	3.8494
`sat-loop(2147483647)-bp`	1.1456	1.7463	3.7267
`sat-loop(2147483647)-c`	1.0000	1.0979	1.6473
`sat-loop(2147483647)-cp`	1.0000	1.0842	1.5861

Table 28: Performance of all 60 selected strategies with respect to peak size as given in Figure 4. For each strategy the performance is measured using all *Mark*s obtained for the models the strategy is able to solve. A *Mark* reveals the performance with respect to the best strategy. The distribution of all *Mark*s for a strategy is given by the lower quantile, median and upper quantile.

C Classifier Evaluation - Data

In chapter 8 the appropriateness of the ten created classifiers is discussion. The performance of the classifiers is compared to fixed strategies, but the discussion is limited to six fixed strategies per aspect. This appendix contains the performance of the classifiers compared to all 60 fixed strategies. Section C.1 contains the evaluation of the classifiers with respect to time. Section C.2 contains the evaluation of the classifier with respect to peak size. The naming convention of section 7.1 is used to name the strategies and `dynamic` is used to denote the classifiers.

The evaluation of each strategy is based on three data sets used to validate the created classifiers. Each strategy is evaluated by listing how many models the strategy is able to solve in each data set. The average capability is captured using by the percentage of models the strategy is able to solve. For each strategy the average *Mark* per data set is determined too. A *Mark* can only be determined for the models the strategy is able to solve. The average performance is captured by averaging the three average *Mark*'s per set weighted by the size of each data set. The rank is determined by the average capability and performance. The average capability has higher priority since the performance is determined based on the models the strategy is able to solve.

C.1 Appropriateness - Time

The capability and performance of each fixed strategy with respect to the created classifiers is listed in Table 29 and Table 30 respectively. The sets correspond to the validation sets listed in Table 14.

Rank	**Strategy**	**Solved models**			
		Set			**Coverage**
		1	**2**	**3**	
1	`sat-like(5)-cp`	114	121	129	97.33%
2	`sat-like(10)-cp`	114	119	128	96.52%
3	`sat-like(20)-cp`	114	118	128	96.26%
4	`sat-like(5)-bp`	113	118	128	95.99%
5	`sat-like(40)-cp`	113	119	125	95.45%
6	`sat-loop(20)-cp`	114	116	127	95.45%
7	`sat-like(80)-cp`	111	118	126	94.92%
8	`sat-loop(80)-cp`	112	116	125	94.39%
9	`sat-like(5)-c`	111	115	126	94.12%
10	`sat-like(20)-c`	111	115	126	94.12%
11	`sat-like(10)-c`	111	115	125	93.85%
12	`sat-loop(10)-cp`	112	114	125	93.85%
13	`sat-loop(40)-cp`	111	116	124	93.85%
14	`sat-like(20)-bp`	109	115	127	93.85%
15	`sat-like(10)-bp`	109	114	127	93.58%
16	`sat-loop(20)-c`	112	113	125	93.58%
17	`sat-like(40)-c`	110	115	125	93.58%

continued on next page

continued from previous page

Rank	Strategy	Solved models			
		Set			Coverage
		1	2	3	
18	sat-like(5)-b	110	114	125	93.32%
19	sat-loop(10)-c	111	113	125	93.32%
20	sat-like(80)-c	110	115	124	93.32%
21	sat-loop(40)-c	111	114	124	93.32%
22	sat-loop(5)-bp	109	113	125	92.78%
23	sat-loop(5)-cp	111	111	125	92.78%
24	sat-like(40)-bp	108	114	125	92.78%
25	sat-like(2147483647)-cp	109	114	124	92.78%
26	sat-loop(80)-c	110	113	123	92.51%
27	sat-loop(5)-c	110	112	124	92.51%
28	sat-loop(10)-bp	109	111	126	92.51%
29	sat-loop(2147483647)-cp	109	113	123	92.25%
30	none-cp	108	112	124	91.98%
31	sat-loop(20)-bp	109	112	123	91.98%
32	sat-like(10)-b	107	111	124	91.44%
33	sat-like(2147483647)-c	107	112	123	91.44%
34	sat-loop(40)-bp	107	111	123	91.18%
35	sat-like(80)-bp	107	112	122	91.18%
36	sat-loop(10)-b	109	109	123	91.18%
37	sat-loop(5)-b	109	108	122	90.64%
38	sat-loop(2147483647)-c	107	110	121	90.37%
39	sat-loop(20)-b	107	108	123	90.37%
40	none-c	107	110	120	90.11%
41	sat-like(20)-b	105	110	122	90.11%
42	sat-loop(80)-bp	107	109	121	90.11%
43	sat-like(40)-b	106	109	122	90.11%
44	sat-loop(40)-b	106	108	120	89.30%
45	sat-like(80)-b	104	106	119	87.97%
46	sat-loop(80)-b	105	105	118	87.70%
47	none-bp	101	103	115	85.29%
48	sat-loop(2147483647)-bp	101	103	115	85.29%
49	sat-like(2147483647)-bp	100	103	114	84.76%
50	sat-like(2147483647)-b	101	99	112	83.42%
51	sat-loop(2147483647)-b	101	99	112	83.42%
52	none-b	101	99	112	83.42%
53	dynamic	106	101	103	82.89%
54	sat-like(1)-cp	106	101	103	82.89%
55	sat-like(1)-bp	102	99	104	81.55%
56	sat-like(1)-c	103	96	102	80.48%
57	sat-like(1)-b	101	93	100	78.61%
58	sat-loop(1)-bp	99	95	100	78.61%
59	sat-loop(1)-cp	98	92	97	76.74%
60	sat-loop(1)-c	96	91	99	76.47%

continued on next page

continued from previous page

Rank	Strategy	Solved models			
		Set			Coverage
		1	2	3	
61	sat-loop(1)-b	98	90	98	76.47%

Table 29: Capability of the classifiers compared to the fixed strategies with respect to time. The capability was determined using the validation data of set 1, 2, and 3 listed in Table 14.

Rank	Strategy	Average *Mark*			Total
		Set			
		1	2	3	
1	sat-like(5)-cp	1.7061	1.7507	1.3707	1.6025
2	sat-like(10)-cp	1.6829	1.6557	1.5177	1.6156
3	sat-like(20)-cp	2.6786	2.7967	2.3374	2.5973
4	sat-like(5)-bp	2.1201	2.1328	1.8142	2.0163
5	sat-like(40)-cp	2.9659	3.2577	3.4510	3.2339
6	sat-loop(20)-cp	3.0461	4.4235	3.4931	3.6606
7	sat-like(80)-cp	3.2759	3.7380	3.7720	3.6042
8	sat-loop(80)-cp	3.3259	5.0142	4.5548	4.3194
9	sat-like(5)-c	2.5988	3.0577	2.0236	2.5479
10	sat-like(20)-c	2.9090	3.2553	2.5527	2.8981
11	sat-like(10)-c	2.6336	2.7934	1.8838	2.4220
12	sat-loop(10)-cp	1.9012	3.2945	2.3721	2.5293
13	sat-loop(40)-cp	3.0246	4.2331	3.8032	3.7001
14	sat-like(20)-bp	5.9246	4.4145	2.2977	4.1439
15	sat-like(10)-bp	3.2931	2.4631	1.6906	2.4523
16	sat-loop(20)-c	3.0059	3.7357	3.0348	3.2581
17	sat-like(40)-c	3.3480	3.8079	2.9363	3.3552
18	sat-like(5)-b	3.6068	3.6824	2.5639	3.2638
19	sat-loop(10)-c	3.3448	4.4647	2.5210	3.4253
20	sat-like(80)-c	3.6438	4.2063	3.3348	3.7212
21	sat-loop(40)-c	3.2980	4.7150	3.4941	3.8370
22	sat-loop(5)-bp	2.3118	5.8230	3.3985	3.8595
23	sat-loop(5)-cp	5.6969	5.0419	2.5252	4.3603
24	sat-like(40)-bp	8.1422	5.5454	3.0387	5.4800
25	sat-like(2147483647)-cp	3.2599	9.3524	5.1719	5.9547
26	sat-loop(80)-c	3.4879	5.0649	3.3221	3.9522
27	sat-loop(5)-c	5.7938	4.5020	2.5170	4.2090
28	sat-loop(10)-bp	7.6882	4.9371	3.2362	5.2048
29	sat-loop(2147483647)-cp	3.1597	9.9751	4.9406	6.0479
30	none-cp	3.2421	10.5876	5.2238	6.3769
31	sat-loop(20)-bp	6.3110	5.5969	7.2698	6.4127

continued on next page

continued from previous page

Rank	Strategy	Average *Mark*			
		Set			Total
		1	2	3	
32	sat-like(10)-b	4.4135	4.1938	2.6790	3.7285
33	sat-like(2147483647)-c	3.6711	5.0031	3.7881	4.1540
34	sat-loop(40)-bp	7.7000	6.8195	3.5073	5.9283
35	sat-like(80)-bp	8.1915	7.3144	4.3158	6.5328
36	sat-loop(10)-b	9.9505	6.5871	4.2888	6.8371
37	sat-loop(5)-b	7.6545	6.0775	3.5601	5.6866
38	sat-loop(2147483647)-c	3.5523	4.6927	3.6008	3.9475
39	sat-loop(20)-b	8.1597	8.1333	5.6516	7.2657
40	none-c	3.8140	4.5177	3.2360	3.8433
41	sat-like(20)-b	7.3590	6.7885	3.3743	5.7635
42	sat-loop(80)-bp	7.9615	7.9065	3.9952	6.5434
43	sat-like(40)-b	10.9210	7.8377	4.5561	7.6523
44	sat-loop(40)-b	10.6552	9.8291	4.9521	8.3685
45	sat-like(80)-b	11.6279	10.0571	6.1794	9.1841
46	sat-loop(80)-b	11.6305	10.6207	5.4556	9.1163
47	none-bp	10.7744	10.6147	8.4602	9.9047
48	sat-loop(2147483647)-bp	11.0528	10.6593	8.6837	10.0862
49	sat-like(2147483647)-bp	9.4196	10.8946	7.0546	9.0740
50	sat-like(2147483647)-b	13.4371	13.7486	9.1227	12.0176
51	sat-loop(2147483647)-b	13.6037	13.5014	9.2084	12.0185
52	none-b	13.6732	13.5287	9.3160	12.0874
53	dynamic	2.2218	1.6579	1.4166	1.7507
54	sat-like(1)-cp	2.2218	1.6579	1.4166	1.7507
55	sat-like(1)-bp	3.0254	2.3160	2.5876	2.6357
56	sat-like(1)-c	2.4192	2.5615	2.1286	2.3638
57	sat-like(1)-b	3.7785	3.9940	3.3273	3.6907
58	sat-loop(1)-bp	2.8873	5.3683	4.5767	4.3061
59	sat-loop(1)-cp	2.6529	3.7246	3.4765	3.2989
60	sat-loop(1)-c	3.5110	4.4413	3.7916	3.9185
61	sat-loop(1)-b	4.3762	6.0261	4.4858	4.9619

Table 30: Performance of the classifiers compared to the fixed strategies with respect to time. The performance was determined using the validation data of set 1, 2 and 3 listed in Table 14.

C.2 Appropriateness - Peak Size

The capability and performance of each fixed strategy with respect to the created classifiers is listed in Table 31 and Table 32 respectively. The sets correspond to the validation sets listed in Table 15.

Rank	Strategy	Solved models			
		Set			Coverage
		1	2	3	
1	sat-like(5)-cp	110	121	135	93.13%
2	sat-like(10)-cp	111	117	135	92.37%
3	sat-like(10)-c	111	116	135	92.11%
4	sat-like(10)-bp	110	116	135	91.86%
5	sat-loop(20)-cp	111	115	135	91.86%
6	sat-like(20)-cp	111	114	135	91.60%
7	sat-like(5)-c	108	117	133	91.09%
8	sat-like(5)-bp	108	116	134	91.09%
9	sat-like(20)-c	108	115	135	91.09%
10	sat-like(40)-cp	110	114	134	91.09%
11	sat-loop(40)-c	110	114	134	91.09%
12	sat-like(40)-c	109	115	133	90.84%
13	sat-like(80)-cp	109	114	134	90.84%
14	sat-like(80)-c	109	115	133	90.84%
15	sat-loop(10)-cp	109	115	132	90.59%
16	sat-loop(40)-cp	109	113	134	90.59%
17	sat-loop(80)-cp	108	114	134	90.59%
18	sat-loop(20)-c	108	113	134	90.33%
19	sat-loop(80)-c	108	114	133	90.33%
20	sat-loop(5)-c	109	114	132	90.33%
21	sat-loop(10)-bp	110	111	133	90.08%
22	sat-loop(20)-bp	110	109	135	90.08%
23	sat-loop(2147483647)-cp	108	113	133	90.08%
24	sat-like(2147483647)-c	108	114	132	90.08%
25	sat-loop(5)-cp	109	113	132	90.08%
26	sat-like(10)-b	109	112	132	89.82%
27	sat-loop(10)-c	108	112	133	89.82%
28	sat-like(20)-bp	109	111	133	89.82%
29	sat-like(2147483647)-cp	108	112	133	89.82%
30	sat-loop(40)-bp	111	109	133	89.82%
31	sat-loop(5)-bp	109	112	131	89.57%
32	sat-like(40)-bp	110	110	132	89.57%
33	sat-like(5)-b	106	113	132	89.31%
34	sat-loop(2147483647)-c	108	112	131	89.31%
35	none-cp	107	111	131	88.80%
36	none-c	106	110	131	88.30%
37	sat-loop(5)-b	105	110	131	88.04%
38	sat-like(20)-b	106	108	131	87.79%
39	sat-loop(20)-b	109	105	131	87.79%
40	sat-like(80)-bp	106	107	131	87.53%
41	sat-loop(40)-b	108	106	130	87.53%
42	sat-like(40)-b	107	107	129	87.28%
43	sat-loop(10)-b	106	106	130	87.02%
44	sat-loop(80)-bp	104	105	131	86.51%

continued on next page

continued from previous page

Rank	Strategy	Solved models			
		Set			Coverage
		1	2	3	
45	sat-like(80)-b	105	105	129	86.26%
46	sat-loop(80)-b	103	105	128	85.50%
47	sat-like(2147483647)-bp	102	104	126	84.48%
48	none-bp	103	103	126	84.48%
49	sat-loop(2147483647)-bp	102	102	126	83.97%
50	sat-like(2147483647)-b	101	102	125	83.46%
51	sat-loop(2147483647)-b	100	101	124	82.70%
52	none-b	97	98	123	80.92%
53	sat-like(1)-c	85	97	107	73.54%
54	sat-like(1)-bp	84	99	105	73.28%
55	dynamic	84	98	106	73.28%
56	sat-like(1)-cp	84	98	106	73.28%
57	sat-like(1)-b	82	96	103	71.50%
58	sat-loop(1)-c	80	95	104	70.99%
59	sat-loop(1)-cp	82	93	103	70.74%
60	sat-loop(1)-b	81	92	104	70.48%
61	sat-loop(1)-bp	81	90	102	69.47%

Table 31: Capability of the classifiers compared to the fixed strategies with respect to peak size. The capability was determined using the validation data of set 1, 2, and 3 listed in Table 15.

Rank	Strategy	**Average** *Mark*			
		Set			Total
		1	2	3	
1	sat-like(5)-cp	1.1140	1.2176	1.2573	1.2008
2	sat-like(10)-cp	1.2223	1.3097	1.3665	1.3040
3	sat-like(10)-c	1.2223	1.3025	1.3665	1.3016
4	sat-like(10)-bp	1.3757	1.4951	1.5038	1.4624
5	sat-loop(20)-cp	2.0014	2.1656	1.8666	2.0082
6	sat-like(20)-cp	1.4416	1.6843	1.7034	1.6183
7	sat-like(5)-c	1.1150	1.2142	1.2602	1.2011
8	sat-like(5)-bp	1.1339	1.3555	1.3675	1.2933
9	sat-like(20)-c	1.4502	1.8808	1.7034	1.6874
10	sat-like(40)-cp	1.5726	1.7383	2.2901	1.8880
11	sat-loop(40)-c	1.7608	1.8835	2.6078	2.1084
12	sat-like(40)-c	1.7140	1.8655	2.5565	2.0697
13	sat-like(80)-cp	1.8908	1.8516	2.4994	2.0974
14	sat-like(80)-c	1.9219	1.9764	2.6823	2.2151
15	sat-loop(10)-cp	1.8044	1.8322	1.5613	1.7260

continued on next page

continued from previous page

Rank	Strategy	Average *Mark*			
		Set			Total
		1	2	3	
16	sat-loop(40)-cp	1.6118	1.7941	2.5053	1.9963
17	sat-loop(80)-cp	1.8834	1.8489	2.6747	2.1576
18	sat-loop(20)-c	2.0289	2.3655	1.8728	2.0864
19	sat-loop(80)-c	1.9314	1.9572	2.7054	2.2198
20	sat-loop(5)-c	2.3952	2.6079	1.9646	2.3116
21	sat-loop(10)-bp	2.2537	2.3084	1.6979	2.0714
22	sat-loop(20)-bp	1.8380	2.2836	2.3452	2.1721
23	sat-loop(2147483647)-cp	1.9659	1.9530	2.6599	2.2123
24	sat-like(2147483647)-c	1.9957	2.0683	2.6725	2.2648
25	sat-loop(5)-cp	2.3952	2.6127	1.9646	2.3132
26	sat-like(10)-b	1.3757	1.4954	1.5140	1.4662
27	sat-loop(10)-c	1.8116	1.8314	1.7549	1.7978
28	sat-like(20)-bp	1.8689	2.4666	2.1155	2.1603
29	sat-like(2147483647)-cp	1.9852	1.9504	2.5063	2.1617
30	sat-loop(40)-bp	2.5357	3.0498	3.3553	3.0058
31	sat-loop(5)-bp	2.5621	2.7770	1.9556	2.4157
32	sat-like(40)-bp	2.4665	3.2953	3.2714	3.0378
33	sat-like(5)-b	1.1336	1.3746	1.3723	1.3014
34	sat-loop(2147483647)-c	1.9957	2.0585	2.6850	2.2660
35	none-cp	1.9639	1.8715	2.4546	2.1099
36	none-c	1.9914	2.0637	2.6850	2.2665
37	sat-loop(5)-b	2.5814	2.7958	1.9560	2.4280
38	sat-like(20)-b	1.8763	2.4841	2.1307	2.1739
39	sat-loop(20)-b	3.5603	4.0824	2.3585	3.3027
40	sat-like(80)-bp	1.9974	3.1230	3.5282	2.9314
41	sat-loop(40)-b	2.5584	3.1324	3.4064	3.0590
42	sat-like(40)-b	2.5046	3.3368	3.3290	3.0841
43	sat-loop(10)-b	2.2796	2.3671	1.7122	2.1042
44	sat-loop(80)-bp	2.0122	2.8416	3.5621	2.8529
45	sat-like(80)-b	2.0022	3.1395	3.5635	2.9512
46	sat-loop(80)-b	1.9922	2.8704	3.6055	2.8723
47	sat-like(2147483647)-bp	2.6101	4.0459	4.2961	3.7052
48	none-bp	2.6210	4.0560	4.2961	3.7119
49	sat-loop(2147483647)-bp	2.6068	3.7738	4.2935	3.6112
50	sat-like(2147483647)-b	2.6211	4.0810	4.6408	3.8449
51	sat-loop(2147483647)-b	2.6065	3.8287	4.6681	3.7651
52	none-b	2.6268	4.1800	4.6949	3.8997
53	sat-like(1)-c	1.0741	1.2275	1.2314	1.1829
54	sat-like(1)-bp	1.0654	1.2236	1.2328	1.1794
55	dynamic	1.0749	1.2252	1.2323	1.1826
56	sat-like(1)-cp	1.0749	1.2252	1.2323	1.1826
57	sat-like(1)-b	1.0722	1.2302	1.2414	1.1868
58	sat-loop(1)-c	4.0304	3.6734	2.7247	3.4378

continued on next page

continued from previous page

Rank	**Strategy**	**Average** *Mark*			
		Set			**Total**
		1	**2**	**3**	
59	`sat-loop(1)-cp`	4.0006	3.5818	2.6249	3.3618
60	`sat-loop(1)-b`	4.0629	3.8115	2.6942	3.4833
61	`sat-loop(1)-bp`	4.0629	3.7123	2.4981	3.3788

Table 32: Performance of the classifiers compared to the fixed strategies with respect to peak size. The performance was determined using the validation data of set 1, 2 and 3 listed in Table 15.

D Reproducibility

This appendix is an addendum of chapter 6. In this appendix it is explained how the various created scripts can reproduce the results described in this thesis. Hence, this appendix lists the technical details of the method. The steps are given in chronological order. The files and scripts are located in `https://vcs.utwente.nl/diffusion/RHMT/`. The scripts mentioned in this appendix refer to the files in that repository.

D.1 Remarks

All absolute paths and user data in the scripts are removed. Therefore, most scripts will not run without modification of the source code. The definition of the corresponding variables must be set. These variables are defined in the beginning of the scripts. Furthermore, all R scripts require a path to the used libraries.

D.2 Data Collection

D.2.1 Preparing Environment

In order to run a large number of experiments, the individual runs with the LTSmin toolset should be automatized as much as possible. The file `./Data_collecting/gen_experiments.sh` can be used to generate a batch of runs. The runs can be executed on a computer cluster. `gen_experiments.sh` assumes SLURM is used to schedule the individual runs and to maintain the resources of the cluster.

Furthermore, `gen_experiments.sh` requires the following directory structure[18]:

- `./bin/ltsminPerf/`
- `./bin/ltsminStat/`
- `./bin/memtime`
- `./experiments/generated/jobs/`
- `./experiments/generated/shuffle.txt`
- `./experiments/generated/steps/`
- `./experiments/gen_experiments.sh`
- `./experiments/in/`
- `./experiments/out/0/`
- `./experiments/out/1/`

[18]The repository does not contain this directory structure. Furthermore, `shuffle.txt` can be an empty file and is also not included in the repository.

The `./` is an absolute path which needs to be specified in `gen_experiments.sh`.

LTSmin needs to be installed in both `./bin/ltsminPerf/` and `./bin/ltsminStat/`. The LTSmin installation in `./bin/ltsminPerf/` will be used to obtain the performance of a run, while the LTSmin installation in `./bin/ltsminStat/` will be used to obtain statistics of a run. In order to function correctly on a cluster, LTSmin should not bind threads on a specific core. This can be achieved by setting `USE_HWLOC` to 0 in `./sylvan/src/lace.c` before building LTSmin. For the LTSmin installation in `./bin/ltsminStat/` the collecting of statistics should be enabled. The statistics can be enabled by setting `SYLVAN_STATS` to 1 in `./sylvan/src/sylvan_config.h` before building LTSmin. When both LTSmin installations are correctly built, the following binaries should be available for `gen_experiments.sh`:

- `./bin/ltsminPerf/bin/pnml2lts-sym`
- `./bin/ltsminPerf/bin/dve2lts-sym`
- `./bin/ltsminStat/bin/pnml2lts-sym`
- `./bin/ltsminStat/bin/dve2lts-sym`

D.2.2 Defining Experiments

The experiments should be defined in `gen_experiments.sh`. Within this script, six variables define the settings of the experiments. The parameters `$PARAM_SLURM`, `$PARAM_GENERAL`, `$PARAM_BDDPACK` and `$PARAM_TEST` define the settings of all experiments of the batch. The parameter `$PARAM_SLURM` defines the settings for SLURM considering one job. The parameters `$PARAM_GENERAL`, `$PARAM_BDDPACK` and `$PARAM_TEST` define settings for LTSmin by specifying flags and their values. Only `$PARAM_TEST` has to be modified when multiple batches of experiments should be run.

The setting differences in the runs within one batch should be specified in `$POPTS` and `$POPTS_VERBOSE`. The values specified in these two variables are an addition to the settings specified in `$PARAM_TEST`. The `$POPTS` specifies the unique runs which should be performed with the LTSmin installation in `./bin/ltsminPerf/`. Likewise, `$POPTS_VERBOSE` specifies the unique runs which should be performed with the `LTSmin` installation in `./bin/ltsminStat/`. Each unique run should be listed in these parameters and the runs should be separated with a `\n`. In order to collect statistics, each entry in `$POPTS_VERBOSE` should include the flags `--peak-nodes` and `--graph-metrics`.

D.2.3 Running Experiments

After defining the experiments in `gen_experiments.sh`, the script should be run to generate the jobs for the cluster. `gen_experiments.sh` has to be executed with the parameters `<repeat> <input> <partition> <ltsminbinary> 1 --gen-steps`. The four variable parameters are:

`<repeat>` = number of times each run is performed for each entry in `$POPTS`.

`<input>` = directory with the models used for the batch. The models should be in a directory located in `./experiments/in/`.

`<partition>` = partition of SLURM used for the experiments.

`<ltsminbinary>` = name of the LTSMIN binary used for the experiments (e.g. `pnml2lts-sym`).

`gen_experiments.sh` generates a run for each model in the input directory and entry in `$POPTS_VERBOSE`. For each model and entry in `$POPTS`, the number of runs generated for the LTSMIN in ./bin/ltsminPerf/ equals the given value for `<repeat>`. For example, if both `$POPTS` and `$POPTS_VERBOSE` list four unique runs, the input directory contains 100 models and `<repeat>` is set to 10, `gen_experiments.sh` generates 4400 experiments.

BASH scripts generated for the individual experiments are stored in `./experiments/generated/steps/`. The jobs for the cluster are stored in `./experiments/generated/jobs/`. The jobs basically specify the task to execute all experiments in `./experiments/generated/steps/`. Furthermore, the file `./experiments/submit_jobs.sh` is generated. `submit_jobs.sh` should be executed to run the batch of experiments. `submit_jobs.sh` schedules all jobs located in `./experiments/generated/jobs/` on the cluster without overloading the system. In this way all experiments in `./experiments/generated/steps/` will be run.

The run of each experiment is stored in a txt-file in `./experiments/out/0/`. The filename contains the name of the model and the settings of LTSMIN as specified in either `$POPTS` or `$POPTS_VERBOSE` depending on the type of run. Each failed run detected by the cluster is stored in the file `./experiments/out/1/failed.txt`.

In order to collect the values for *ES*, *NES*, *WES* or *NWES*, the option `mm` should be added to the reorder flag. Since the values are calculated for a model, it does not matter whether this experiment is performed using LTSMIN in `./bin/ltsminPerf/` or `./bin/ltsminStat/`. However, the calculation of the value for *ES*, *NES*, *WES* or *NWES* may take some time.

D.3 Data Analysis

D.3.1 Parsing Data

The data obtained from the batches of experiments is stored using a txt-file per experiment. The relevant information can be extracted using the created parser. `./Data_refining/ltsmin2csv.sh` is the script which is able to parse the txt-files generated by LTSMIN. The parser takes a directory of txt-files and lists the extracted values in a csv-file. For each experiment one row will be added to the resulting csv-file.

`ltsmin2csv.sh` requires the parameters `<directory> <ltsminbinary> <output> [<satgran>]`. The interpretation of these parameters is:

`<directory>` = directory containing the LTSMIN output of a batch.

`<ltsminbinary>` = name of the LTSmin binary used for the experiments (e.g. `pnml2lts-sym`). This name is used to verify whether a file is generated by running a experiment.

`<output>` = name of the file where the results are stored. The output is a csv-file; a csv extension is recommended.

`<satgran>` = optional parameter. Must be provided if `--sat-granularity` is not specified in `$POPTS` but in `$PARAM_TEST`. Allow the parser to set the saturation granularity for all experiments to the specified value.

When the experiments are collected in multiple batches, after parsing the txt-files, the data may be scattered over multiple csv-files. The script `./Data_collecting/Runs/merge_csvs.sh` can be used to combine the csv-files into a single csv-file.

`ltsmin2csv.sh` does not extract the values for *ES*, *NES*, *WES* or *NWES*. The values for these metrics have to be extracted using `./Data_refining/ltsmin2eventstats.sh`. This will generate a csv-file with *ES*, *NES*, *WES* and *NWES* per experiment.

D.3.2 Refining Data

The runs can be examined using `./R_source/Data_refining/errorFinder.r`. This script finds conflicting entries in the data in the provided csv-file (e.g. two runs came up with a different size of the state space for the same model). `errorFinder.r` gives a csv-file with these conflicting entries and all entries with a small state space. The resulting csv-file has to be checked manually to discover the errors in the data.

The discovered errors in csv-file generated by `errorFinder.r` can be removed using `./R_source/Data_refining/errorRemover.r`. Currently, the script fixes the errors discovered in `./R_source/Data_refining/errorList1.csv`. `errorRemover.r` takes the csv-file containing all runs and removes corrupt entries and produces the result in a new csv-file.

After removing the corrupt entries, the data needs to be summarized. The script `./R_source/Data_refining/performanceOverview.r` generates the performance overview for the csv-file. The script will fix the saturation granularity for all runs and summarize the multiple runs for each strategy and model pair in a single record. `performanceOverview.r` will also determine the *Mark* for time and peak size.

The csv-file with the values for *ES*, *NES*, *WES* and *NWES* may contain multiple entries per model depending on the definition of the experiment in the batch. The script `./R_source/Data_refining/eventstatsMerger.r` can be used to merge the entries per model. The script will deliver the results in a csv-file with one record per model.

D.3.3 Strategy evaluation

The performance overview generated in the previous step can be evaluated. The directory `./R_source/Strategy_evaluation/` contains multiple scripts to analyze the performance of the strategies for the different models. Each script will create diagrams for the performance overview. The data presented in the diagrams is given on the standard output.

The script `analyzeSolved.r` reveals the number of models each strategy is able to solve. The script `analyzePerformance.r` will generate an overview of the *Mark*s per strategy. These two scripts are sufficient to reproduce the results in chapter 7.

D.4 Classifiers

D.4.1 Training Data Creation

The creation of training data is separated in three steps. Firstly, the best strategy is determined for each model. The best strategy can be either the strategy with the lowest solving time or the strategy with the smallest peak size for a model. Secondly, using the best strategy overview, the features of a model are linked to the best strategy. The second step creates a list with all input and output couples for the classifiers. Lastly, the data with the input and output couples is split into training and validation sets. The training sets can directly be used to train a classifier. The validation set can be used to evaluate the performance of the classifiers created with the corresponding training set.

The scripts `createTrainingTime.r` and `createTrainingPeaksize.r` in `./R_source/Trainingdata_creation/` determine the best strategy per model with respect to time and peak size respectively. The results are stored in a csv-file which contains the best strategy per model.

The script `./R_source/Trainingdata_creation/createTrainingComplete.r` uses the csv-file with the best strategies, the csv-file with the performance overview and the csv-file with the values for *ES*, *NES*, *WES* and *NWES* to create the training data. The training data is stored in a csv-file. Each entry in the training data list the name of the model, the features of the model and the best strategy for that model.

The actual training and validation sets are created using the script `./R_source/Trainingdata_creation/splitModelsComplete.r`. The script does not consider the records where one or more features for a model are unknown. `splitModelsComplete.r` divides the training data five times as listed in section 6.3.4 and section 8.1. Each training set and validation set is stored in a separate csv-file.

D.4.2 Classifier Creation And Evaluation

The classifiers are created and evaluated using the PYTHON script `./Python_source/StrategyML_flatten.py`. Within the file the location of the training and validation data needs to be given. In this script the classifier of

SCIKIT-LEARN is also selected. The file handles the complete pipeline: reads training data, trains classifier with selected classification algorithm, evaluates classifier using the validation data and shows the results on the standard output. The output is given in simple format by four floating point values. The first value represents $Recall_M$, the second value represents $Precision_M$, the third value represents $Accuracy_M$ and the last value represents $Accuracy_\mu$.

The predictions of the classifier are stored in a csv-file. In order to determine the appropriateness the predicted strategies of the created classifier, the script `./R_source/Classifier_evaluation/resultVerifier.r` can be used to compare the predictions of the classifier with the fixed strategies.

D.4.3 Feature Relevance Analysis

In order to determine the influence of each feature, the PYTHON script `./Python_source/StrategyML_missing.py` can be used. The script is a modification of `StrategyML_flatten.py`. In this script it is possible to specify how many features should be removed from the training data. The script will determine all combinations of removing the specified number of features. For each combination a classifier will be created. The creation and evaluation of these classifiers are based on the training and validation sets of the three random distributions of models. The results are put on the standard output per set. For each set the combinations are listed including the four metrics ($Recall_M$, $Precision_M$, $Accuracy_M$ and $Accuracy_\mu$) for that combination.

Currently, there is no script available to compare the records produced by `StrategyML_missing.py`. The results can be put into an EXCEL sheet and manually be sorted to discover relevant differences.