Universiteit Leiden

Master Computer Science

Per-instance Algorithm Selection for MIP-based Neural Network Verification

Name:	Jasmin Kareem
Student ID:	s3357937
Date:	November 27, 2023
Specialisation:	Data Science
Daily supervisor:	Matthias König
Daily supervisor:	Annelot Bosman
1st supervisor:	Dr. Jan N. van Rijn
2nd supervisor:	Prof.dr. Holger H. Hoos

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Whilst neural networks are an incredibly powerful tool used in many different applications, it has been shown that they are vulnerable to adversarial examples. Adversarial examples are small perturbations to the input, which causes the network to confidently misclassify the label. One way we can address this problem is by formally verifying if a network is robust to a particular perturbed input, known as Neural Network Verification (NNV). Within NNV there is a trade-off between completeness and scalability. Whilst complete verification methods offer guarantees about a network's robustness, they are often not used in practice due to their scalability issues. This means that many verification instances cannot be verified within a reasonable amount of time due to their inherent complexity, especially for large networks and input dimensionality. Some complete verification methods can be considered as solver-based, *i.e.* using either a mixed integer programming (MIP) or satisfiability modulo theories (SMT) solver to verify an instance. In this work, we aim to address the verification scalability problem by looking at ways solver-based complete verification could be made faster. In particular, we are interested in using a MIP-solver-based verifier to create verification instances and then verify them using a MIP-solver. Whilst prior research has shown the potential of running a portfolio of MIP-solver configurations in parallel to achieve speedups, another potentially fruitful approach is to select which algorithm in a portfolio solves each instance the fastest. This is known as per-instance algorithm selection. In this work, we will use MIP instance features to select between different configurations of the MIP solver to achieve further speedups on the parallel portfolio approach. We find that MIP features on their own are not enough to predict the running time of solving an instance and hence this method is not able to select the best configuration. At the same time, we show that MIP instance features are able to accurately predict the satisfiability of an instance with a 98% accuracy and an F1-score of 92% on the default configuration, excluding timeouts. We foresee that MIP features in combination with a simple classifier could help estimate the likelihood of an instance being robust to adversarial attacks, and provide fast estimates of a network's overall robustness.

Acknowledgements

First of all, I would like to thank my supervisors Jan N. van Rijn, Holger Hoos, Matthias König, and Annelot Bosman for all the guidance they gave me during this project. Our meetings every two weeks were instrumental in helping me get a better understanding of this field of research. Each time I was able to meet with Holger I was left with valuable insights that I had not previously considered. Furthermore, I would like to thank fellow master student Hadar Shavit who was a tremendous help, answering many of the questions I had about tools you had used before. I would also like to thank Cesar Gesta, for your support and for helping me with installing SCIP. Lastly, I am grateful for having been a part of the ADA research group. It has been a pleasure to experience office life with you, drinking the mediocre coffee by the machines at the Snellius and talking about our weekends.

Table of Contents

1 Introduction **1**

2 Background **3**
2.1 Mixed Integer Programming . . . 3
2.2 Adversarial Examples . . . 4
2.3 Neural Network Verification . . . 4
2.3.1 Incomplete Methods . . . 5
2.3.2 Complete Methods . . . 5
2.4 Algorithm Selection . . . 7

3 Related Work **8**
3.1 Efficient Neural Network Verification . . . 8
3.2 Predicting Satisfiability . . . 9
3.3 Relevance of this work . . . 9

4 Methods **10**
4.1 Verification method . . . 10
4.2 Feature Extraction . . . 10
4.3 AutoFolio . . . 11
4.4 Satisfiability and Running Time Prediction . . . 12

5 Experiments **13**
5.1 Datasets . . . 13
5.2 Pipeline . . . 14
5.3 Evaluation . . . 16
5.4 Practicalities . . . 17

6 Results **18**
6.1 Finding MIP instance features . . . 18
6.1.1 Running time . . . 18
6.1.2 Satisfiability . . . 20
6.2 Per-instance algorithm selection . . . 20
6.3 Predicting the Satisfiability of an instance . . . 22
6.3.1 Utilising satisfiability prediction for NNV . . . 22

7 Conclusion **25**
7.1 Limitations . . . 26
7.2 Future Research . . . 26

References **27**

A Additional material **31**
A.1 Running time prediction 31
A.2 Principal component analysis 33
A.3 Timeout prediction 33
A.4 AutoFolio results 34
A.5 Satisfiability prediction 36
A.6 Histograms 36
A.7 Top features 37

Chapter 1

Introduction

Deep neural networks (DNN) are widely used to solve many problems in society and have been incredibly successful in doing so in recent years. Whilst achieving near human levels of accuracy on tasks such as image classification, they are known to be vulnerable to slight changes in the values of the input data that lead to a misclassified label with high confidence [1], [2]. This slightly changed, misclassified input is referred to as an *adversarial example*. These examples illustrate how several machine learning models, including neural networks, lack robustness to these small changes or perturbations.

When considering the task of image classification, an adversarial example is characterised as some level of perturbation to the pixels of an input image. This is a problem because neural networks are increasingly adopted in real-world applications such as self-driving vehicles, medical imaging, and facial recognition. As neural networks are more frequently used in these safety-critical applications, the risk that a network is attacked, using methods that add perturbations to an input image as described in [1], is also increased. This can have serious consequences, for instance, a self-driving car could be fooled into classifying a stop sign as a priority road sign or pedestrians could be classified as free space. In the medical imaging domain, noise in an image of an MRI scan could drastically impact the ability of a network to classify a particular illness.

For this reason, it is essential to develop techniques to increase the robustness of a DNN w.r.t. adversarial examples. One solution to the problem is to perform so-called adversarial training. For instance, training a DNN with the fast gradient sign method (FGSM) introduced by the authors of [1] initially showed promising results but was quickly circumvented by more sophisticated attacks [3]. Another approach is to train models, which detect adversarial examples; this includes methods such as defensive distillation [4] and feature squeezing [5]. However, even these approaches were later shown to be ineffective against stronger attacks [6]–[8]. This shows that relying on empirical defence mechanisms is not enough and can easily lead to an endless cat-and-mouse game. Another issue with solely depending on empirically robust neural networks is that we can never be certain that a network is fully robust to an arbitrary input image without engaging in some form of verification.

On the other hand, neural network verification (NNV) methods can provide formal guarantees about the behaviour of a DNN with regard to adversarial input perturbations. More specifically, it is the process of verifying for any possible input instance, whether a DNN output satisfies some defined constraints. One way to perform such verification is by determining the minimal distance to the closest example for each instance [6]. Other metrics for the robustness of a network look at both the adversarial frequency (the number of instances a network fails to be robust) and the adversarial severity, which measures how severe the network's failure to be robust is [9]. In this case, a network is considered robust if a small enough perturbation to the input image doesn't affect the corresponding label.

Early formal verification methods encoded neural networks and the associated verification property into a satisfiability modulo theories (SMT) problem and used an SMT solver to solve them [10]; however, this implementation was inefficient and could only be used to verify small networks. In later work, more sophisticated verifiers were introduced, able to handle larger and more complex

networks [11], utilising the simplex method. Even with recent improvements [12], solving these SMT problems is not trivial and can be quite time-consuming. As complete verification has been proven to be NP-complete [11], applying these algorithms to large datasets becomes much harder.

Alternatively, we can encode a neural network into a mixed-integer programming (MIP) problem and use MIP solvers to find a solution [13]. Previously, this method was shown to improve performance over the state-of-the-art SMT solver Reluplex [11]. However, recent developments in extending the simplex method and branch-and-bound (BaB) approaches have been shown to be more efficient at verifying networks than prior methods [12], [14].

Another aspect of the verification problem is that methods can either be complete or incomplete [15]. Both complete and incomplete methods are sound, that is, the verification algorithm will only state that a property holds if it indeed holds. The main difference between the two is that an incomplete algorithm will not always guarantee a definitive output, whilst complete methods do, given enough computational resources. Therefore, the benefit of a complete algorithm is that given enough compute time the verifier can, in principle, always answer the question of whether a network is provably robust or not. Yet, in practice, these algorithms can be quite inefficient, which limits their usage in real-world applications. For this reason, it is vital that we find methods to improve the efficiency of complete neural network verification algorithms.

In this work, we seek to study MIP-based verification and how features of a MIP instance can be used to make the process of verifying a network more efficient. We intend to use the features of a MIP instance to predict the running time of a solver for each instance [16] and make use of the strengths of automated algorithm selection [17], [18] to select between different configurations of a MIP solver.

Previous work has shown that different configurations of a MIP solver have varying solving times and that using them in parallel can drastically speed up the verification process [19]. The main aim of this work is to build on top of these findings and further improve the efficiency of the proposed approach by leveraging algorithm selection techniques. We show that selecting between the MIP solver configurations using a broad range of MIP instance features does not yield improvements to the efficiency of a MIP-based verifier.

A secondary sub-objective of this thesis is to study other predictive capabilities of MIP features and use them to predict whether an instance of the verification problem is solvable, *i.e.*, predicting satisfiability. We show that MIP features are very useful for predicting the satisfiability of an instance, achieving accuracy and an F1 score of 0.98 and 0.92, respectively, on the default configuration excluding timeouts. Furthermore, we describe how these features can be operationalised.

Moreover, we studied the following research questions and sub-questions:

RQ1: What are relevant features of a MIP instance?

- **RQ1.1:** Which features are useful for running time prediction?
- **RQ1.2:** Which features are useful for predicting satisfiability?

RQ2: Does per-instance algorithm selection using MIP features further reduce the computational cost for MIP-based neural network verification?

RQ3: How well do MIP features predict the satisfiability of a verification instance?

- **RQ3.1:** How can we utilise this to make the verification process more efficient?

In the following section (Chapter 2), a background on the relevant topics is given in order to form a basic understanding of the problem. In Chapter 3, we describe related work on ways to make NNV more efficient, in addition to describing prior relevant work on satisfiability prediction. In Chapter 4, the proposed methods are described in detail, and in Chapter 5 we report the experimental setup of our research, including the datasets and networks used. Finally, we analyse the results and address our research questions in Chapter 6, with concluding remarks and future work outlined in Chapter 7.

Chapter 2

Background

The goal of this chapter is to describe the necessary background needed to understand our research objectives. This includes background on MIP formulations, adversarial examples, and the concept of robustness, as well as an overview of neural network verification and algorithm selection.

2.1 Mixed Integer Programming

A Mixed-Integer Linear Programming problem (MILP or MIP) describes a problem where the objective, bounds, and constraints are all linear. The aim is to minimise or maximise this objective and use algorithms to find the best solution. A simple example of a MIP instance can be found in Equation 2.1 and the corresponding graphical representation in Figure 2.1. As shown in the example, the feasible region of a MIP problem is the set of all possible solutions that satisfy the constraints given.

$$\begin{aligned} \text{minimise} \quad & x \\ \text{subject to} \quad & 2x + y \geq 4 \\ & x \geq 1 \\ & y \geq 0 \end{aligned} \tag{2.1}$$

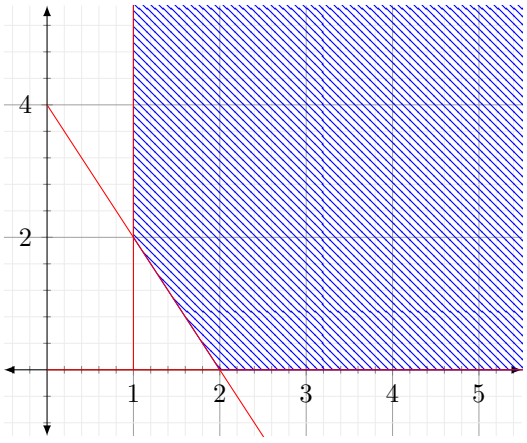

Figure 2.1: A simple example of a MIP instance, showing the constraints in red and the feasible region in a shaded blue.

In general, we can describe the objective of a MIP instance as $c^T x$, where c represents the coefficients for the variable matrix x. Furthermore, linear constraints resemble a set of linear equations $Ax = b$ and $l \leq x \leq u$ is the general formulation commonly used to describe the bound constraints, where l and u represent the lower and upper bounds respectively.

There are several ways we can solve a MIP instance. A common approach is to use the Branch-and-Bound algorithm [20], which partitions the original problem into smaller sub-problems that are easier to solve and then discards the subset of problems that can never reach an optimal solution.

2.2 Adversarial Examples

As described in [1], [2], adversarial examples have showcased a weakness in machine learning models; specifically, they incorrectly classify examples that have been slightly altered from the original input with high confidence. An example of this is demonstrated in Figure 2.2, where noise is added to the pixels of an image of a panda and is thus misclassified as "gibbon" with 99.3% confidence. The strength of the perturbation is controlled with the parameter ϵ where the intention is to keep the perturbations reasonably small. Here θ represents the parameters of the model, x the input image, x' the perturbed input, with y representing the target label of x. More concretely, we consider a classifier f robust if the perturbations are small enough such that the classification of the corresponding label is not affected; $\|x - x'\|_\infty \leq \epsilon$. In this way, we can interpret ϵ as a radius that bounds the perturbations on the input image.

Furthermore, the expression $\text{sign}(\nabla_x J(\theta, x, y))$ symbolizes the loss which is linearised around the models' parameters θ. This is referred to as the adversarial objective of the fast gradient sign method (FGSM) [1]. This method simply works by attacking the gradients of a neural network which are inadvertantly linked to the input image. Adversarial examples thus present the opportunity for machine learning models (including neural networks) to be attacked in a very simple and cheap way. For this reason, we are interested in formally proving the adversarial robustness of a neural network.

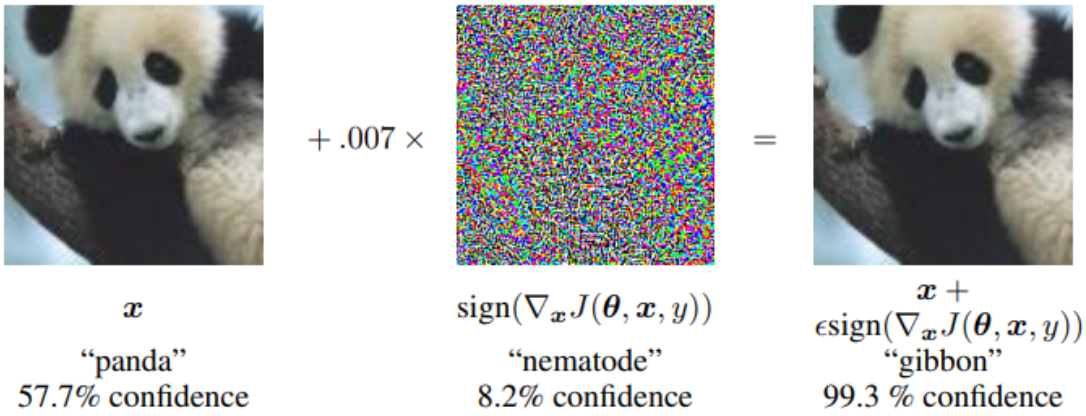

Figure 2.2: An illustration of an adversarial example, originally described in [1]. Adding a slight perturbation in the pixels of an image of a panda has a drastic effect on the neural network's ability to find the correct label. Here the injected perturbation is denoted as $\text{sign}(\nabla_x J(\theta, x, y))$ with an ϵ of 0.007 controlling the strength of the noise. In this example, the classifier predicts "gibbon" with 99.3% confidence.

2.3 Neural Network Verification

Early work on MIP formulations showed that MILP problems can be reformulated into various forms. One such example is that of the "big-M" formulation described in [21]. Moreover, the input images $x \in \mathcal{X}$ and related outputs of a neural network can be formulated as linear constraints. Thus, we can define robustness verification as an optimization problem that can be solved using any MIP solver. Prior work has used the big-M encoding [13], [22], [23] as a basis for verification. Local robustness verification can hence be defined in the following way, as described in [13]. Given

a neural network where $f_i(x)$ represents the i^{th} output of the network with input instance x and corresponding true label $\lambda(x)$, we get the following optimization problem;

$$\begin{aligned} \text{minimise} \quad & d(x', x) \\ \text{subject to} \quad & \text{argmax}_i(f_i(x')) \neq \lambda(x) \\ & x \in (G(x) \cap \mathcal{X}_{\text{valid}}) \end{aligned} \tag{2.2}$$

Here $d(x', x)$ is the distance between the original input x and altered input x'. For instance, this distance metric could represent the l_1, l_2 or the l_∞ norm, where the latter is most commonly considered [15]. By minimizing this distance we ensure that we are either able to find an adversarial example that maximizes the error that the model produces, or it is provably robust within the stated radius ϵ. More concretely, if the solver finds that the problem cannot be solved, *i.e.* all solutions are infeasible, then there exists no perturbation that lies within the constraints stated above. Conversely, if a solution to the MIP problem is found then the network is not robust.

Furthermore, the constraints in the optimization problem denote the following. The first constraint implies that the trained neural network misclassified the instance, where the argmax operator returns a single element, predicting a single class label. If this predicted label is not equal to the true label $\lambda(x)$, then we uphold the constraint. Moreover, the second constraint states that the original input instance x should be a valid input in the domain of $\mathcal{X}_{\text{valid}}$. That is it should be within the pixel value range for a normalized image such that $\mathcal{X}_{\text{valid}} = [0, 1]^m$. In addition to this, x should fall into the region $G(x)$ covering all possible and permitted perturbations. More precisely, we define $G(x)$ as $G(x) = \{x' \mid \forall i : -\varepsilon \leq (x - x')_i \leq \varepsilon\}$, where all perturbed input x' for all i is bounded by ϵ.

There are many more methods that seek to verify neural networks. Verifiers are often segmented into two groups: incomplete and complete methods. The difference and trade-off here is that while complete methods offer a guaranteed verification outcome, they lack scalability which limits their use to smaller neural networks. On the contrary, incomplete methods are relaxations of the optimization problem and thus are computationally much cheaper, yet lacking completeness, that is they do not always guarantee a verification output for all instances. An overview of common verification methods, both complete and incomplete, can be found in Figure 2.3.

2.3.1 Incomplete Methods

Incomplete methods use relaxations and approximations of the original optimization problem to achieve faster verification. Whilst some use randomized smoothing, which 'smoothes' the original classifier and offers fairly tight bounds [24], other approaches focus on linear programming (LP) relaxations [25]–[27] and semidefinite programming (SDP) relaxations [28], [29]. Moreover, methods like [30]–[32] use polyhedra abstractions of the RELU activation function. Again the main drawback of all these approaches is that they are able to prove robustness for some instances, as they are sound but incomplete.

2.3.2 Complete Methods

Within the domain of complete verification methods, two main research paths have emerged. As briefly mentioned in Chapter 1, complete verification can be either SMT and MIP solver-based where existing solvers are used for verification, they can use solvers and make extensions to the internal simplex algorithm used by these solvers such as Marabou [12], or they could use BaB-based methods such as β-CROWN [14]. In particular, BaB-based verifiers have become the current state-of-the-art for verification, with the α, β-CROWN verification framework [14], [32] winning the international verification of neural networks competition *VNN-COMP* in both 2021 and 2022 [33].

Solver-based methods

The initial SMT solver-based method [10] had the major drawback of only being able to handle the verification of small and specific networks. Since then, using an existing MIP solver like Gurobi

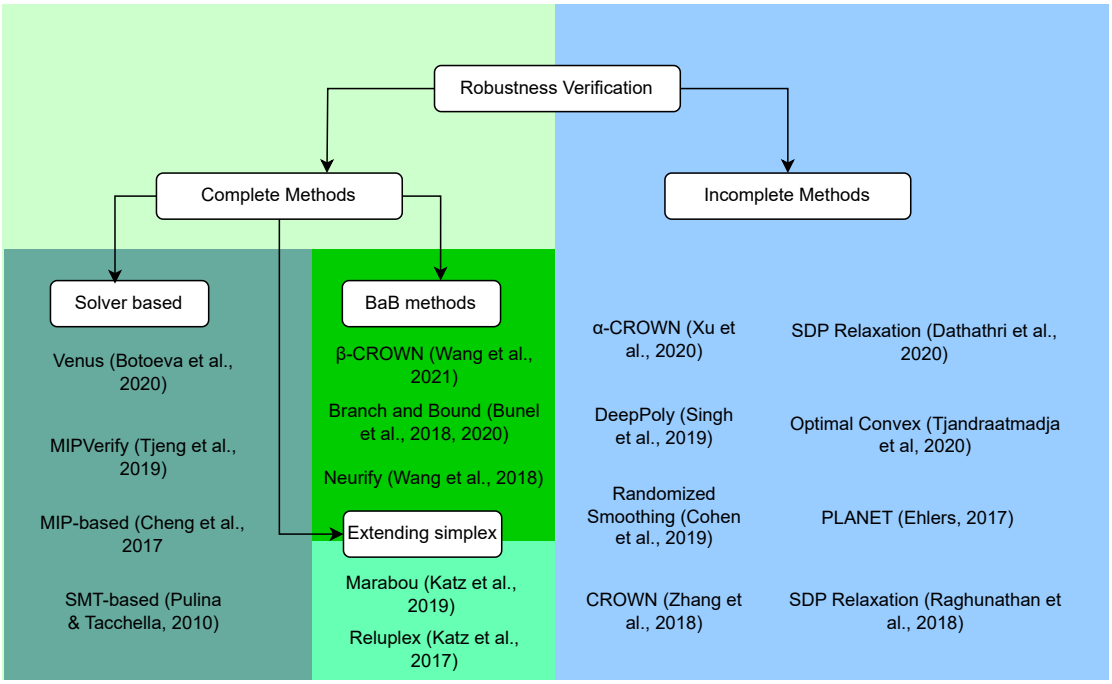

Figure 2.3: A simplified overview of the different verification approaches based on [15]. Here we show the distinction between complete and incomplete methods, as well as whether the verifier uses an existing solver, extends the simplex methods with an existing solver, or applies the branch-and-bound algorithm.

[34] to solve a MIP formulated verification instance has been a straightforward approach. For instance, in [35] the use of smaller big-M encodings to formulate the verification instances enabled the verifier to be more efficient as the size of the instance has a big impact on solving time. Another more recent example of a MIP-based approach is the Venus verifier [22], which also uses the big-M encoding. Similar to BaB, Venus utilizes a divide-and-conquer approach to recursively cut the problem so that it is small enough to be able to solve each of the smaller sub-problems. Lastly, the MIP-based verification tool MIPVerify [13] optimizes its MIP formulations by pre-solving the instance and thus reducing the size of the problem. On top of this, tight formulations for non-linearities of the activation function further reduce the computational cost.

Alternatives to Solver-based methods

Recently many tools used for verification utilize the branch-and-bound (BaB) optimization technique. The strength of this method is that it uses a cheaper incomplete verifier to derive a lower and upper bound of the verification problem. This works by combining the branching with the bounding procedure, where if the bounding stage fails, meaning the bounds found by the verifier did not give a conclusive verification outcome, the branching stage then divides the search space into smaller pieces and tries to verify the properties again until it is verified. This is the case for the fast and incomplete verification method β-CROWN, which combined with the BaB-based algorithm, produces a complete verifier with GPU acceleration [14]. Other BaB-based methods mentioned in Figure 2.3 include the works of [36] or [23].

An alternative to BaB-based methods is to extend the simplex method [11], [12], which is an approach to solving linear programming problems. The purpose of extending the simplex method is so that the respective solvers can handle the non-convex Rectified Linear Unit (ReLU) activation function. This is because non-linear activation functions cannot be encoded into a problem that the original simplex algorithm can solve. Whilst Reluplex does use an SMT solver, in some sense making it solver-based, by extending the original simplex method it is no longer considered solver-based. The successor of this approach, known as Marabou [12], was built on top of this

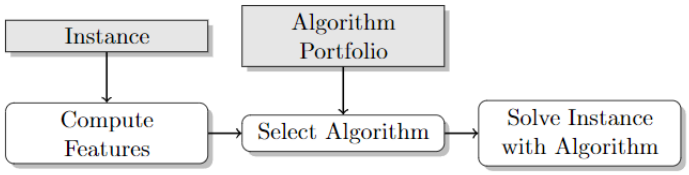

Figure 2.4: A general outline of per-instance algorithm selection. Figure sourced from [40], which is based on [17].

by extending the types of network architectures possible, allowing for arbitrary piecewise-linear activation functions beyond ReLU.

2.4 Algorithm Selection

For many computational problems, there is no single algorithm that works best over all problem instances. Instead, some algorithms perform better on a certain set of instances and perform worse on others. This fact has led to the development of per-instance algorithm selection, which is the process of selecting the best algorithm from a set of algorithms, originally described in [17]. The main objective of per-instance algorithm selection is to minimize the computational cost of running a set of multiple algorithms over many instances by using a predictive model to select an algorithm on a per-instance basis, thus improving upon simply using the same single algorithm each time.

Per-instance algorithm selection is enabled via the computation of features from each problem instance, which are then used to make predictions on which algorithm performs best on a given instance, as outlined in Figure 2.4. These instance features are extracted and dependent on the type of instance, which could be in the form of a MIP, satisfiability (SAT), or travelling salesperson (TSP) problem. Moreover, the cost of extracting these features is incorporated in the performance measure of an algorithm selector. This is because the time it takes to calculate a feature diminishes the objective of taking less time overall.

The evaluation of algorithm selection methods is often done by comparing performance with what could have been achieved without algorithm selection. This is known as the *Single Best Solver* (SBS), which is the best-performing single algorithm over all instances. On the other hand, the perfect selector or *Virtual Best Solver* (VBS) describes what would happen if for each instance we chose the most optimal algorithm. Of course, this is not a realistic scenario but the aim is to get as close as possible to this oracle.

When we run the aforementioned set of algorithms in parallel, we refer to them as a portfolio of algorithms [37]. Parallel portfolios of algorithms are beneficial to solving each instance faster as they can terminate early if one of the algorithms has found a solution to the respective problem. One successful algorithm selection tool that employs a SAT solver with the portfolio framework is *SATZilla* [38]. To obtain the improvements made by SATZilla, machine learning models to approximately predict running time are used, using the features described in [39]. Another unique aspect of SATZilla is its use of pre-solving schedules, *i.e.* running a pre-solving schedule of algorithms for a short amount of time. If one of the algorithms solves the instance, SATZilla ends the process, thus saving time with feature extraction costs.

Conceptually related to automated algorithm selection, algorithm configuration takes an algorithm, instances, and a performance measure as input and configures the algorithm's hyperparameters so that the performance of this algorithm is optimized over all instances. Algorithm configuration can be combined with algorithm selection to further improve performance. This is the case for the algorithm selector AutoFolio [40], which automatically configures multiple algorithm selection approaches, including those used in SATZilla. Moreover, it extends the machine learning strategies and models used, incorporating various regression and pairwise classification models, which leads to improved performance. Algorithm configuration can also be combined with the parallel portfolio approach to create a portfolio of configured algorithms [41], [42].

Chapter 3

Related Work

In this chapter, we will discuss the related approaches attempting to make MIP-based neural network verification more efficient. Furthermore, we will examine relevant work on predicting the satisfiability of instances. This will be followed by a description of the relevance of this work to the aforementioned related research.

3.1 Efficient Neural Network Verification

There are various ways in which one could consider making neural network verification more efficient. As discussed in Chapter 2, there are two main avenues of verification approaches, some with more of a focus on completeness (complete methods) and others on scalability (incomplete methods). These incomplete methods offer speedups to the verification process, yet lack the guarantee of an output. Current state-of-the-art verification tool α, β-CROWN [14], [32] combines the best of both, taking advantage of α-CROWN's GPU hardware accelerated relaxation and combining it with the parallel BaB-based approach of β-CROWN, which ensures efficiency and completeness. This combination was also further analyzed in the VNN competition series [33], where the top two and three teams of the 2021 and 2022 editions, respectively, based their method on the BaB approach. Moreover, the top two teams in 2022 utilized multi-neuron relaxations [43] and solver-generated cutting plane constraints [44]. Overall these findings showed that using GPU hardware accelerated approaches could help with managing tighter MIP encodings and thus providing a more efficient verifier. However, there are still some remaining challenges. BaB-based verifiers do not scale well with networks containing a large number of neurons [33]. This signifies that research into scaling complete methods is still an important endeavour.

Whilst a hardware-accelerated approach is one way to address faster verification, another is to leverage the strengths of running a portfolio of algorithms in parallel. This is precisely what is done in [19], which constructs a parallel portfolio of MIP solver configurations built using the portfolio configuration tool Hydra [41], [42]. In particular, the commercial MIP solver Gurobi [34] was used and configured in different ways to produce the portfolio. Moreover, different MIP solver-based verifiers were used to encode the network and input images into MIP instances. This included verifiers MIPVerify [13] and Venus [22]. The resulting research led to a significantly larger fraction of instances being verified and a reduction in the overall CPU and wallclock time spent on solvable instances. Furthermore, it showed that these solver configurations are complementary to one another, showcasing the strengths of Hydra's parallel portfolio construction, and substantiating its use in this case.

Although the parallel portfolio approach showcased a lowering of the number of timeouts and reduced the overall CPU and wallclock time, running 4 solver configurations in parallel is still quite time-consuming. It is also difficult to make a direct comparison to more recent complete methods such as [12], [14], which may be more usable for most verification use cases.

3.2 Predicting Satisfiability

Predicting the satisfiability of an instance has previously been described in [45]. It showed that using SAT instances, it is possible to predict with a relatively high classification accuracy, the satisfiability of instances at the 3-SAT solubility phase transition. With limited features, accuracies were reported of about 70%. To give some background, random 3-SAT problems are a type of SAT problem that consists of 3 randomly generated Conjunctive Normal Form (CNF) formulae. Moreover, the phase transition is an important phenomenon of SAT instances, revealing an underlying distribution of instances that range in solvability. Some instances are easily identifiable as either sat- or unsatisfiable, in other words, solvable or unsolvable. Other instances that are closer to the boundary between the two classes, *i.e.* closer to the phase transition, are much harder to solve and classify.

Since this work, there has not been a large continuation of this line of research into predicting the satisfiability of other types of instances. It focused solely on predicting the solvability of 3-SAT instances. As we are interested in predicting the solvability of MIP instances, other instance features and methods will need to be considered. Originally, the paper stated that these satisfiability predictions could be used to achieve better running time prediction, as running time and satisfiability are intricately related. However, additional research needs to be done to find out in what other ways these predictions can be used and if they can help us improve our understanding of instances at the phase transition in the verification domain.

3.3 Relevance of this work

This work aims to introduce the benefits of algorithm selection to neural network verification, making the process of running a verification algorithm more efficient. To the best of our knowledge, this is the first work that tries to incorporate algorithm selection in neural network verification. We note that algorithm selection for MIP-based problems is not new, however, their application to the field of neural network verification is. What is presented in this research is heavily based on the work by [19], directly building upon it. We aim to use the parallel portfolio construction described in [19] as a set of algorithms to select from. Because of this dependency, we also use many of the same methods. In the coming chapters, this link with this prior work will be described in more detail. In addition to per-instance algorithm selection, we would like to study if MIP instance features can predict the solvability or satisfiability of a verification instance. We note that there is a large body of work related to the study of SAT problems in general [46]. We aim to use this prior work to the extent of providing a proof of concept for MIP instance solvability prediction.

Chapter 4

Methods

This chapter will cover the methods used to achieve results for per-instance algorithm selection. Henceforth, we refer to Chapter 2.4 to describe the individual methods that make up the algorithm selection pipeline. Firstly, we will cover the verification tool used to create the instances, as shown in the *instances* block of Figure 2.4. These instances are then used to extract MIP features on a per-instance basis. Following this, we describe how we use an algorithm selection tool to select a solver configuration from the portfolio described and constructed in [19]. In addition to the algorithm selection pipeline, we describe the machine learning model used to predict the satisfiability and running time of solving an instance.

4.1 Verification method

A key aspect of this research is the verification method used to create the MIP instances that are to be verified. In terms of the algorithm selection pipeline in Figure 2.4, this is the part where we define the instances. We decided to focus on MIPVerify [13] as a complete MIP-based verifier since we were interested in the problem of scaling up complete verification and leveraging the predictive power of MIP features. MIPVerify creates MIP instances that can be solved by any MIP solver. The encoding of a trained neural network into a verification instance is described in more detail in Chapter 2.3. In our case, we use the commercial solver Gurobi [34] to create and solve instances. The purpose of solving the instances is to generate information on the running time and status of a solver configuration for each instance. Here we follow the work of [19], which configures the Gurobi solver into multiple configurations using the parallel portfolio algorithm configuration tool Hydra [41]. We use the portfolio of solver configurations as the set of algorithms to select from.

Other complete MIP-based verifiers could have been used [22], [35], however, this became out of scope as one verifier on multiple networks was enough to convey initial findings of this method, in addition to the fact that complete verification is time-consuming and thus would need a lot more resources to obtain results within the timeframe of this project. Another reason for sticking with MIPVerify was because this work is based on prior research on complete MIP-based verification [19], which also makes use of MIPVerify, which we intended to expand on using algorithm selection instead of a parallel portfolio.

4.2 Feature Extraction

To be able to predict which solver configuration works best on a particular verification instance, we first need to extract features to base these predictions on. In this research, we decided to focus on the MIP features that can be extracted from an instance. This feature extraction process requires a MIP solver and some defined statistics of interest that the MIP solver can compute. The chosen statistics are an important factor in this equation as different features may have varying predictive power on the running time of an instance. Prior research into algorithm running time prediction [16], which is a proxy for selecting an algorithm from a portfolio, gave some insights and formal-

ized which MIP features could be leveraged for this problem. These features were divided into several groups, each having an associated timing feature that represents the time it takes to compute the corresponding feature group.

For this work, we decided to use two different feature extractors; the MIP features outlined in [16] using the CPLEX solver, and the feature extractor created for the open-source solver SCIP [47]. There are 121 instance features mentioned in [16], including the timing features. These features vary in computational complexity to extract. For instance, whilst problem size features are quite trivial to compute, the LP-based and Probing features require solving the instance until a certain point. This is done by enforcing a node limit on the solving of the BaB tree within the MIP solver. Another time-consuming feature group to extract are the pre-solving features. In this context, pre-solving a MIP instance is to reduce the size of the problem, removing any redundancies, and thus uncovering more information about the instance.

Whilst the features presented in the work mentioned above showed improvements in predicting running time, no research followed it in succeeding at introducing new features for this purpose. For this reason, it was decided to find an alternative to cover more potentially predictive features. SCIP is an open-source solver actively contributed to by the Zuse Institute Berlin (ZIB), thus offering a wide variety of features to extract from a MIP instance. As described in the ZIB report [48], we can define SCIP features into three categories; static features without pre-solving, static features with pre-solving, and dynamic features. Static features are named such because they do not require solving the instance. On the other hand, dynamic features are all based on solving the problem until a defined node limit. In our work, we use a node limit of 1, meaning we only solve until the root node of the BaB tree. In total, there are 146 static features and about 1674 dynamic features. The static features were obtained using the feature extractor described in [49]. On the other hand, to obtain the dynamic features of a MIP instance we needed to extract the statistics generated from solving the instances in SCIP.

4.3 AutoFolio

Once we have obtained features for all instances and defined the portfolio that we would like to select an algorithm from, we can begin with the algorithm selection part of Figure 2.4. There are various algorithm selectors to use for MIP-based per-instance algorithm selection. This includes the previously mentioned SATZilla and AutoFolio, along with another predecessor Claspfolio 2 [50]. A more in-depth background of these methods is described in Chapter 2.4. Being the most recent work, as well as automatically configuring many algorithm selectors and showcasing improved performance in many scenarios [40], we decided to use AutoFolio. Another advantage of AutoFolio is that it combines the strengths of Claspfolio 2 with the sequential model-based algorithm configuration method known as SMAC [51]. SMAC automatically configures algorithms using Bayesian optimization and combines this with a racing mechanism to decide between different configurations. In AutoFolio, SMAC is used to automatically configure each of the machine learning algorithms that AutoFolio uses internally. These models cover pairwise classification and various regression and clustering methods. These are combined to decide on which algorithm to select. As with most algorithm selectors, AutoFolio requires the creation of an ASlib scenario [52], which is simply a way to store the input data for the algorithm selection process. This input data includes the running times of each of the solver configurations in the algorithm portfolio, the features extracted from each MIP instance, and the feature computation costs. Going back to Figure 2.4, once an algorithm in the portfolio has been selected, we run this selected MIP solver configuration on the instance it was selected for. In our case, this has already been done in the constructed portfolio created by [19].

4.4 Satisfiability and Running Time Prediction

Outside of the algorithm selection framework, we are also interested in the satisfiability of an instance. To predict the satisfiability of an instance with the large number of features that we have, we need to employ a classification model. This model should classify between the labels satisfiable, unsatisfiable, and due to the time limit, timeout. We decided to use a random forest classifier to achieve these results. Random forests can either be used for classification or regression purposes. In both cases, they are made up of an ensemble of many decision trees, which use the given features to make splits and decisions to make a prediction. The combined prediction of all decision trees becomes the random forest predictor. Other machine learning models could be considered. However, due to the number of features and the number of instances, more complex models with higher computational costs would defeat the purpose of predicting satisfiability for a verification instance, as we want to make it more efficient. Furthermore, satisfiability prediction is a secondary objective of this research, and thus we are interested in simply showing a proof of concept.

Whilst AutoFolio incorporates running time prediction with random forests as one of the possible machine learning methods it utilises for selection, it does this implicitly. This means that we cannot obtain information on how well our features specifically predict the running time of an instance directly from AutoFolio results. For this reason, we use a random forest regression model to obtain preliminary results on running time prediction and to gain insights into the predictive value of MIP features.

Chapter 5

Experiments

This chapter will describe the dataset and the precise process of how the different aspects of this research come together and how they produce the results we find in the following chapter. Moreover, here we will relate this pipeline to the research questions defined in Chapter 1. We will also explain how we evaluate these results. Lastly, any practicalities regarding the experimental setup of this research are noted.

5.1 Datasets

Creating the verification instances requires several inputs; a set of images, inputs to the verification method, and a trained neural network. Each instance represents an image inputted to that network. We chose to simply use the full MNIST dataset [53] of 10000 images, which is a dataset consisting of images of hand-written numbers from 0 to 9. This dataset is widely used for machine learning classification tasks and is used in several verification methods, including the work of [19]. The trained neural networks we utilize are thus MNIST digit classifiers. The first network we consider is the $\mathrm{SDP_dMLP_A}$ network, which is described in [28] and was specifically designed to be a more robust neural network. Secondly, the classifier $\mathrm{mnistnet}$ described in [22] is also created with adversarial robustness in mind. Both networks use the RELU activation function.

One major difference between the two is network size. Whilst $\mathrm{SDP_dMLP_A}$ only has 2 layers, $\mathrm{mnistnet}$ has 3, with both having an input of 784 features, one for each individual pixel, however, the former having a middle layer of 500 neurons which then link to the 10 output neurons. By contrast, $\mathrm{mnistnet}$ adds an additional layer of 512 neurons to the network. The larger the network the more time it will take to create and solve instances. Lastly, MIPVerify requires a time limit and level of perturbation that needs to be set. For the $\mathrm{SDP_dMLP_A}$ network, again following prior work, we chose a perturbation level or ϵ of 0.1, with a 9600 second time limit. Again to stay in line with prior work, for the $\mathrm{mnistnet}$ network we used $\epsilon = 0.05$, with the same fixed timeout set. Both networks use the l_∞ norm as a distance metric.

Once the verification instances for both networks have been created, they can be solved using the parallel portfolio of Gurobi configurations as described in [19]. The results of running the portfolio configurations contain information on the solving time of each configuration, and their status (*i.e.* if the instance is optimal, infeasible, or if it reached timeout). This in combination with the MIP instances is the dataset that we base our research on. An overview of statistics on the networks in combination with data obtained from using MIPVerify and Hydra can be found in Table 5.1.

Note that in Table 5.1 we are not able to provide the number of instances in the best configuration for $\mathrm{mnistnet}$. This is because the information on running times for each configuration was taken from [19], which is missing some data on running times for each configuration. This makes it impossible to do running time prediction and thus algorithm selection on this dataset without re-running the configurations in [19], which is out of scope for this project. For this reason, we are only able to use $\mathrm{mnistnet}$ for predicting the satisfiability of the solver configurations.

Network	n layers	architecture	n configs	n instances optimal	n instances infeasible	n instances timeout	n instances in best config
$\mathrm{SDP_dMLP_A}$	2	(784, 500) (500, 10)	4	837	6505	2658	3300
mnistnet	3	(784, 512) (512, 512) (512, 10)	2	6859	2984	157	Not applicable

Table 5.1: An overview of the two neural networks used in combination with MIPVerify and the parallel portfolio of Gurobi configurations described in [19]. Information includes network characteristics, as well as statistics of the best configuration. Here we describe the number of instances that are optimal, infeasible, and timeout for the default configuration for both networks. Lastly, we show the size of the majority class of the best configuration over each instance.

5.2 Pipeline

There are many intermediary steps to take to get to the point where answering the research questions of interest is possible. For this reason, it is important to have an overview of how the individual components fit into a pipeline. A diagram of such a pipeline is described in Figure 5.1.

Creation of MIP instances

The MIP instances were created using the Julia programming language, incorporating Gurobi, MIPVerify, the MNIST dataset, and the encoded trained neural network. For both networks, we extracted the 10000 instances, one for each MNIST image. These instances in combination with the solver configurations in [19] produce the solver configuration running time and status information we will use in the following steps.

Extracting MIP instance features

In this step, we start to consider the following research questions:

RQ1: What are relevant features of a MIP instance?

RQ1.1: Which features are useful for running time prediction?

RQ1.2: Which features are useful for predicting satisfiability?

These questions are already important when deciding what features to extract. As mentioned in Chapter 4, we decided to use two different MIP feature extractors. Whilst there is a large overlap between the features considered by the CPLEX-based method and the SCIP features, the latter certainly extracts many more features. This allows us to possibly find new features with predictive power on solver running time. Within the realm of research on estimating how long a MIP instance will run, the MIP gap is often considered one of the most useful measures for the solving time of a BaB algorithm [54], which is what most MIP solvers are based on. This is because the MIP gap measures the gap between the primal and dual bounds, meaning the smaller this gap gets the closer the solver is to finding a solution. We consider this feature in both feature extractors, as it is a dynamic feature for SCIP and a probing feature for the CPLEX-based feature extractor. No matter which features we find to have predictive power, estimating the BaB tree size and thus the remaining time the solver needs to finish is not trivial and is still an active field of research.

Clean extracted features

As mentioned in Chapter 4, there are 146 static SCIP features and approximately 1674 dynamic features. Since we run the static features twice, once without and once with pre-solving, we get a total of 1966 SCIP-based features. Whilst this is quite a large number, they are by no means all equally valuable. Many features have the same values for every instance, which makes them unusable for prediction purposes. In the data-cleaning process, we remove any features that are either constant over all instances or are duplicates of other features. The resulting number of SCIP-based features for the $\mathrm{SDP_dMLP_A}$ network is 363, which is a much more manageable number of features to consider. For the mnistnet network the number of SCIP-based features is 434. We note

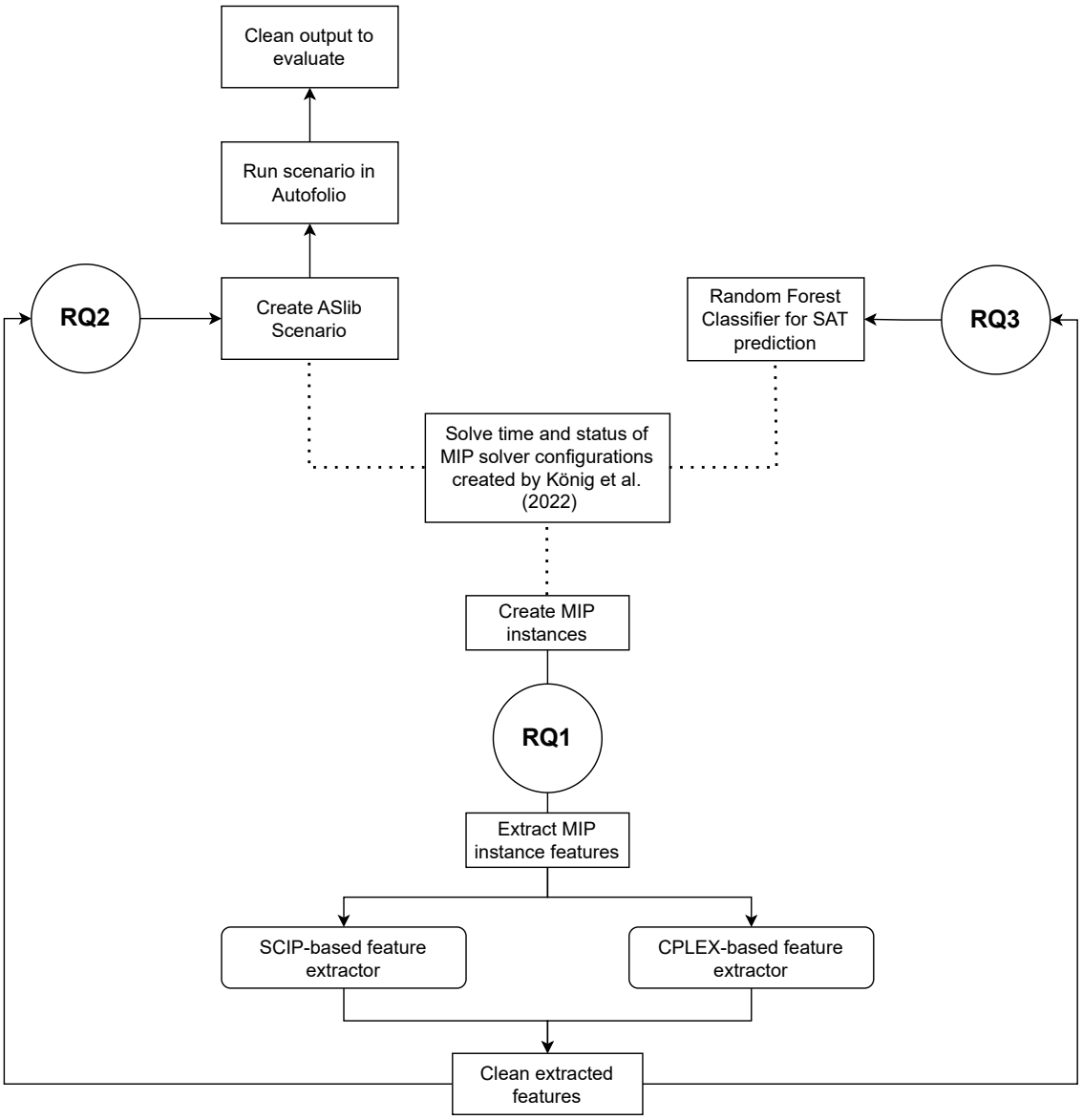

Figure 5.1: An overview of how the different aspects of this work are brought together in a pipeline. Here **RQ1**, **RQ2** and **RQ3** represent the work relating to the corresponding research questions. The arrows represent the order of the process. The dotted lines represent a relation, for example, the same MIP instances we create are used in the work by [19] to produce solver configuration running times. The data from this work is then used in combination with the extracted features to reach our research goals.

that extracting the dynamic features from the SCIP log files causes some features to be removed. This is because the log files are not consistent over all instances and therefore some features were removed because they were only present in a small number of instances. Lastly, we do the same data-cleaning procedure for the CPLEX-based features. The resulting number of features is 81 for $\mathrm{SDP_dMLP_A}$ and 85 for $\mathrm{mnistnet}$.

ASlib and AutoFolio

To answer our second research question, which states:

> **RQ2:** Does per-instance algorithm selection using MIP features further reduce the computational cost for MIP-based neural network verification?

we use the ASlib format to combine the extracted features with the running time data created by [19]. When creating the ASlib scenario there are a few things to consider. The first aspect of creating a scenario is how we should define the feature groups that AutoFolio will use to make predictions and hence selections. The purpose of feature groups is to define the features into groups where each group has an associated cost to compute. In the case of the CPLEX-based features, the feature groups and corresponding feature group costs are already defined [16]. On the other hand, the SCIP-based features have no clear grouping other than the static pre-solve off, on, and dynamic features. Since computing static features is negligible and no concrete data exists on computing time, we set the cost of computing static features to 0. By contrast, pre-solving can be a time-consuming task which we include in the feature cost for both the static pre-solve-on and dynamic feature groups. Lastly, as part of the SCIP log files, we obtain a total solving time feature which we use for the computation of dynamic features.

Another choice to make when creating ASlib scenarios is the number of cross-validations and how to divide the instances. We use 10-fold cross-validation and choose to randomly assign the instances to each fold such that each fold has 1000 instances. We use the Penalized Average Running Time (PAR) measure to compute the running times, as it is used in comparable work [19] and implemented in ASlib [52]. More specifically, this means we multiply the timeouts by a factor k thus penalizing instances that timeout. We choose to use $k = 10$ *i.e.*, PAR10, as was done in [19].

A further choice we make is we run the SMAC algorithm configuration tool for 48 hours, as was recommended in [40]. Once AutoFolio is done, we obtain a folder with many output files over all 10 cross-validations. To use these files for evaluation, we clean them so that we obtain the VBS, SBS, and AutoFolio running time for all 10000 instances. This can then be used to see if AutoFolio does indeed improve on the SBS or gets close to the VBS. Another approach we could compare our results to is the parallel portfolio approach introduced in [19], however, due to some differences in the number of instances used in the evaluation, a one-to-one comparison was not entirely possible within this project. We also obtain the configuration selections AutoFolio makes through the scheduler and use this as another insight into the performance of our method.

Satisfiability prediction

Lastly, to address our secondary objective and the third research question, which states:

RQ3: How well do MIP features predict the satisfiability of a verification instance?

RQ3.1: How can we utilise this to make the verification process more efficient?

We use a simple random forest classifier to make use of the obtained MIP features from both feature extractors to obtain results. To obtain our results, we use a 50/50 train/test split with stratification. As this is a secondary objective of this work, our aim is not to implement a method to incorporate this into making neural network verification more efficient. This is out of scope and will need to be addressed in the future. Therefore, to answer **RQ3.1**, we simply provide a discussion based on the findings of the classifier.

5.3 Evaluation

As we have research questions with different methods, we must also consider different evaluation metrics for these different questions. We describe the evaluation we use to answer each of our research questions.

RQ1

We use Random Forest classifiers and regressors to predict both the satisfiability as well as the running time of a solver configuration of a MIP instance. Using the built-in Random Forest top-n features function within the python package *Scikit-learn* [55], we can see which features the model found were most predictive. We report this for all solver configurations. Additionally, we use

principle component analysis (PCA) to measure how these features predict the outcome with some level of variability. As PCA constructs principle components made up of the eigenvectors and values of the covariance matrix of the features of interest, it is a method to not only reduce the dimensionality of a dataset but also to understand feature importance and correlations between features. It also allows us to visualize the feature space in lower dimensions.

RQ2

We compare our AutoFolio results with the single and virtual best solvers. A common approach to quantify this difference is through the measure known as the closed gap. We define the closed gap as follows:

$$\text{closed gap} = \frac{m_{\text{SBS}} - m_{\text{s}}}{m_{\text{SBS}} - m_{\text{VBS}}}$$

where m_{s} represents the average performance over all instances of the algorithm selection method, with m_{SBS}, m_{VBS} representing the same but for SBS and VBS. A perfect score in this metric is a 1, meaning our method is as good as the Oracle VBS, and 0 corresponds to being on par with the SBS. We also use the average solving time of all 10000 instances of AutoFolio and compare this number with the average solving time of the SBS and VBS.

In addition to this metric, we look at the selection accuracy of AutoFolio *i.e.* how often did AutoFolio indeed select the best configuration. Lastly, we create another two metrics to give insight into the number of instances the method solves before a set time. The first, which we call *Compare Solved*, computes the number of instances AutoFolio solved versus the SBS, the number of instances solved by SBS versus AutoFolio, and the number of instances solved by neither. To clarify, solving, in this case, means solving the instance before reaching the timeout. The other metric we use is called *Compare Times*. Here we compare the running times of the SBS with our results from AutoFolio, where we measure the number of instances AutoFolio was faster, the number of instances the SBS was faster, and the number of instances both were equally fast.

One last way we evaluate our results is to experiment with slightly different settings of AutoFolio to ensure that the algorithm selection process is working. This includes a scenario with undersampling on the best configuration classes, not including feature costs, allowing for feature group selection, and limiting the number of features set in the scenario. Lastly, we conduct a sanity check of AutoFolio by providing it with the running times of all configurations.

RQ3

As we are using a simple Random Forest classifier for satisfiability prediction, we simply use the accuracies and F1 scores to evaluate our results. As Random Forest models are ensembles of decision trees, and we do not configure the model for performance gains, we do not need to repeat the experiments and average out results. Moreover, we show satisfiability prediction results for all configurations over both chosen networks and running time distributions over all instances grouped by the satisfiability of an instance.

5.4 Practicalities

The majority of this work was produced using the GRACE cluster used by the ADA research group[1]. This consists of cluster nodes containing two Intel® Xeon® CPU E5-2683 v4 2.1GHz with 16 cores each and 94 GB of memory. These CPU nodes were used to create the MIP instances, extract features of the CPLEX-based extractor, and run AutoFolio. The only exception was the SCIP-based feature extractor as installation on the cluster was not possible. For this purpose, another computer was used to extract those features. All code used in this work can be downloaded from and is available at:

`https://github.com/jasminkareem/nn-verification_algorithm-selection`

[1] https://ada.liacs.nl/

Chapter 6

Results

In this chapter, we address the research questions previously outlined and show our findings produced by the experiments in the pipeline defined in Chapter 5. In summary, our findings show that per-instance algorithm selection for MIP verification instances does not work with the MIP features found. By conducting this research, we discover the opportunity for using MIP features to predict the satisfiability of a MIP instance.

6.1 Finding MIP instance features

To answer the question of which MIP features are relevant to predicting either the running time or the satisfiability of an instance, we show the features contributing most to these predictions. More details on the meaning of the individual features can be found here for SCIP [1] and the CPLEX-based features [2] which were also described in [16]. We divide this question into the following two sub-sections.

6.1.1 Running time

One of the ways an algorithm selector selects an algorithm to run for a given instance is to predict the running time of each of the algorithms and select the algorithm with the lowest predicted running time. To find relevant features for this prediction we use the feature importance function of a random forest regression model. Table A.2 lists the top 10 features of each of the configurations for both feature extractors for the $\mathrm{SDP_dMLP_A}$ dataset. These rankings were generated using the default hyperparameters of the random forest model in the Python library Scikit-learn. Moreover, Figure 6.1 showcases the predicted versus actual running time of the Random Forest Regression model for both feature extractors. Figure 6.2 shows results obtained from PCA and shows how the explained variability, calculated using the cumulative explained variability ratio, increases as the number of principal components increases. Again, we implement this for both feature extractors.

To answer the question of *which features are useful for running time prediction?*, we begin by analyzing Table A.2. In 5 out of the 8 configurations and feature combinations, as we might expect the mipgap is present in the top 10 predictive features. Moreover, features relating to the bounds of the LP problem seem to be quite frequent as well. But how much do these features explain how running time changes over all instances? We show Random Forest Regression predictions in the top row of Figure 6.1. For the default configuration on the $\mathrm{SDP_dMLP_A}$ dataset for both features, we see that our features don't have a strong connection with running time. We report a root mean squared error (RMSE) of 0.686 with the SCIP features and 0.92 with the CPLEX-based features. Compared to the results in [16] for running time prediction, these are fairly high error values. As initial results, preceding per-instance algorithm selection, this is not promising. On the positive side, we do see clusters of instances that seem to be along the ideal prediction line in both cases of features. Note that this cluster is found by the dynamic features as predictions only using static

[1] https://www.scipopt.org/doc-7.0.0/html/index.php
[2] https://www.ibm.com/docs/en/icos/20.1.0?topic=classes-cplexstats

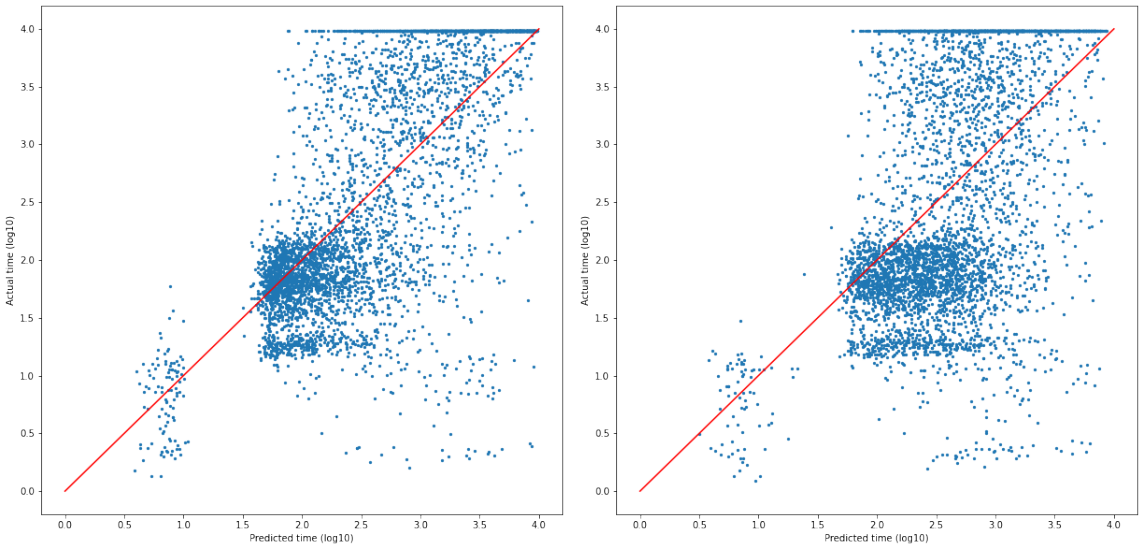

Figure 6.1: Left: Random Forest Regression for default configuration on the $\mathrm{SDP_dMLP_A}$ dataset and using all SCIP features. The predictions obtain an RMSE of 0.686. Right: Random Forest Regression for the default configuration using the CPLEX-based features, obtaining an RMSE of 0.92.

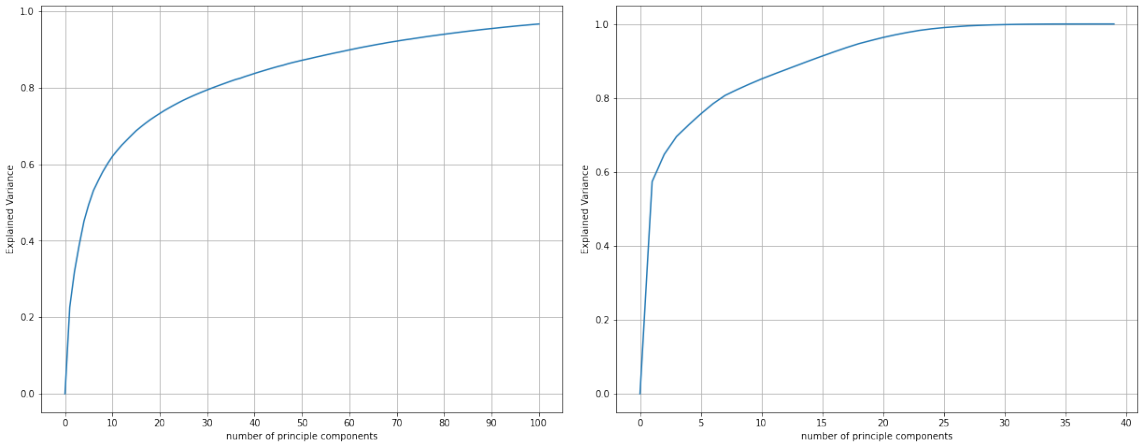

Figure 6.2: Left: The explained variability as the number of principal components increases for the default configuration of the $\mathrm{SDP_dMLP_A}$ dataset with SCIP features. With the number of principal components set to 2, we are merely able to explain about 32% of the variability in the data. Right: The explained variability as the number of principal components increases for the default configuration of the $\mathrm{SDP_dMLP_A}$ dataset with CPLEX-based features. With 2 principal components, we can explain about 64.8% of the variability in the data.

SCIP features on the same data do not report this cluster. Details on this can be seen in Appendix A in Figures A.1 and A.2.

Looking at the results obtained from PCA, we see that the CPLEX-based features explain about 65% of the variability in the data with just 2 principal components. On the other hand, SCIP features only explain about 32%. This increases to 60% with 9 principal components. This could be due to the differences in the number of features, as SCIP has a total of 146 and CPLEX 81 features. These principal components could be used instead of the vast number of features found, however, this comes with the caveat that we may oversimplify the feature space, meaning our predictions may suffer.

6.1.2 Satisfiability

As one of the objectives of this work is to predict the satisfiability of an instance, it is important to investigate the importance of the features of such predictions. Tables A.3 and A.4 show the top 10 features for the default configuration for both feature extractors and both datasets. In addition to this, we show the corresponding accuracies and F1 scores of the classifiers. To showcase the effect of timeouts on predictions, we also report results with timed-out instances removed from the dataset before we run the model. In Figure 6.5, we showcase the default solver configuration status predictions in the form of a confusion matrix. In Figure 6.3, we show how the solver status clusters over 2 principal components using SCIP features for both $\mathrm{mnistnet}$ and $\mathrm{SDP_dMLP_A}$. The same plots for the CPLEX-based features can be found in the Appendix in Figure A.5.

Other than running time, we would like to learn something about how MIP instance features could be used to predict satisfiability, as our research question states (*which features are useful for predicting satisfiability?*). From Tables A.3 and A.4, we see that whilst there is certainly overlap with the types of features reported in the top 10, they vary substantially between $\mathrm{mnistnet}$ and $\mathrm{SDP_dMLP_A}$. We note that for the $\mathrm{mnistnet}$ dataset, some of the most predictive features for satisfiability are the number of variables and constraints after pre-solving. Lastly, Figure 6.3 shows the two found principal components with some defined clusters for both the $\mathrm{SDP_dMLP_A}$ and $\mathrm{mnistnet}$ datasets with SCIP features. These same clusters are not as visible with the CPLEX-based features. Overall, these PCA results suggest that no one feature is all-encompassing in predictive power. Instead, it is likely the combination of many features that help make these predictions.

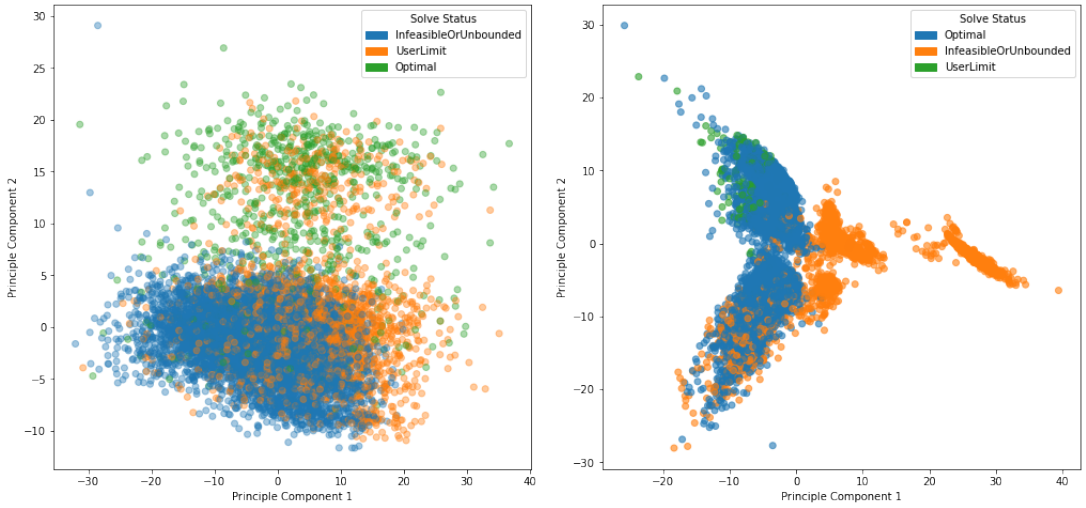

Figure 6.3: Reduced dimensions using PCA with two principle components with the SCIP-based features on both $\mathrm{SDP_dMLP_A}$ (left) and $\mathrm{mnistnet}$ (right) for predicting the status of an instance.

6.2 Per-instance algorithm selection

The primary objective of this work is to answer the question of *Does per-instance algorithm selection using MIP features further reduce the computational cost for neural network verification?*. In Table 6.1, we present the results of different ASlib scenarios with different AutoFolio settings to try to answer this question. We see that no matter how we change the settings and scenario, we are not able to beat the single best solver, with a closed gap of essentially 0. We tried removing feature costs, and feature groups (meaning we place all features in one group), reducing the number of features, and undersampling the dataset by best configuration class to ensure that there was no imbalance in predictions. None of this proved to be fruitful. The SCIP features also showed no improvements in performance. All associated figures with these results can be found in Appendix A, including a confusion matrix of the selected configurations. Our findings imply one of two things, either the

Method	Closed gap	Avg. CPU time	Selection Accuracy	Compare Solved sbs	Compare Times sbs
AF CPLEX	-0.077345	16610.32004874575	0.3092	(8, 23, 1642)	(484, 1253, 8263)
AF CPLEX + no costs	-0.069634	16597.57940072358	0.3064	(8, 22, 1642)	(504, 1999, 7497)
AF CPLEX + no groups	-0.057227	16577.08488054715	0.3145	(7, 17, 1643)	(302, 8055, 1643)
AF clean CPLEX	-0.01225	16502.79504586128	0.3245	(5, 8, 1645)	(324, 7623, 2053)
AF clean CPLEX + undersampling	-0.0687307	21685.67737568222	0.2029	(12, 21, 1173)	(594, 3678, 1173)
AF SCIP	-0.042997	16553.58118749386	0.2928	(12, 22, 1638)	(907, 6767, 2326)
sbs 10k	-	16482.56105497079	-	-	-
vbs 10k	-	14830.81423665438	-	-	-
sbs undersampling	-	21553.55401183142	-	-	-
AF sanity check	0.99387	14840.935594802666	0.8084	(154, 1, 1496)	(5037, 126, 4837)

Table 6.1: Results from various scenarios and settings for the AutoFolio algorithm selector for the $\mathrm{SDP_dMLP_A}$ dataset.

algorithm selector is not functioning well (due to an error or simply poor performance) or the MIP features are not powerful enough to select a solver configuration from this portfolio of algorithms. We assume that AutoFolio is indeed a well-equipped algorithm selection tool. To demonstrate that if we had better features we would also have better predictions, we provide a sanity check, which is a run of AutoFolio where the only features given are the running times of each of the solver configurations, with all else kept equal. We report a closed gap of about 0.99, demonnstrating that the features are the problem in this use case for algorithm selection for neural network verification. We graphically show how AutoFolio with MIP features performs against SBS in Figure 6.4. Here we see that whilst most instances are on par with the single best solver, there are also many instances where the algorithm selector is slower.

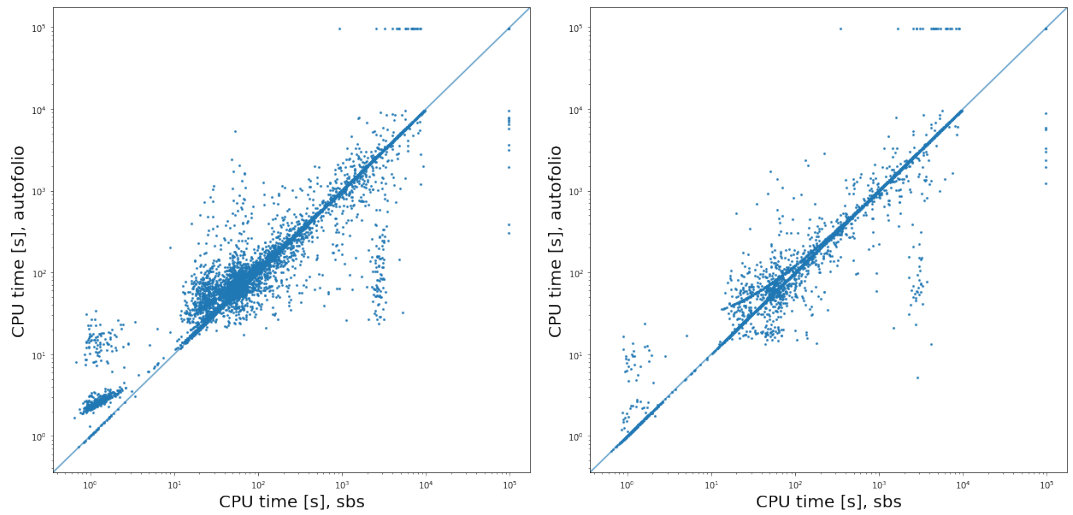

Figure 6.4: AutoFolio Results for the $\mathrm{SDP_dMLP_A}$ dataset using SCIP features - AF SCIP in Table 6.1 (top) and CPLEX-based features (bottom) - AF CPLEX in the same table. Here we show the CPU time it took for the single best solver versus the algorithm selector. If an instance is on the line $y = x$, then both methods are on par.

6.3 Predicting the Satisfiability of an instance

Our last research question(s) relate to the prediction of the satisfiability of an instance and how MIP features could be used to achieve this. Moreover, how might we utilize it in making formal neural network verification more efficient? To begin with, MIP instance features seem to successfully predict the satisfiability of an instance. Without timeout instances, we reach accuracies above 97% and F1 scores of about 90% using the SCIP features over all 4 configurations. This is also visible in the confusion matrices in Figure 6.5, where we see that the model mainly misclassifies instances that are optimal or infeasible, SAT or UNSAT, with timeout instances. Given the high F1 scores, these predictions seem to be fair and are not due to an imbalance in the three classes. We also see this performance in the mnistnet dataset, showing that there is potential to use MIP features for SAT prediction in a wider variety of datasets, results for which can be found in Appendix A. Investigating further into how satisfiable and unsatisfiable instances are distributed over running time. We see in Figure 6.7 for all four configurations of the $\mathrm{SDP_dMLP_A}$ dataset, that satisfiable instances are often concentrated around the lower running times and in the case of configuration 3 there are also instances close to the timeout limit of 9600.

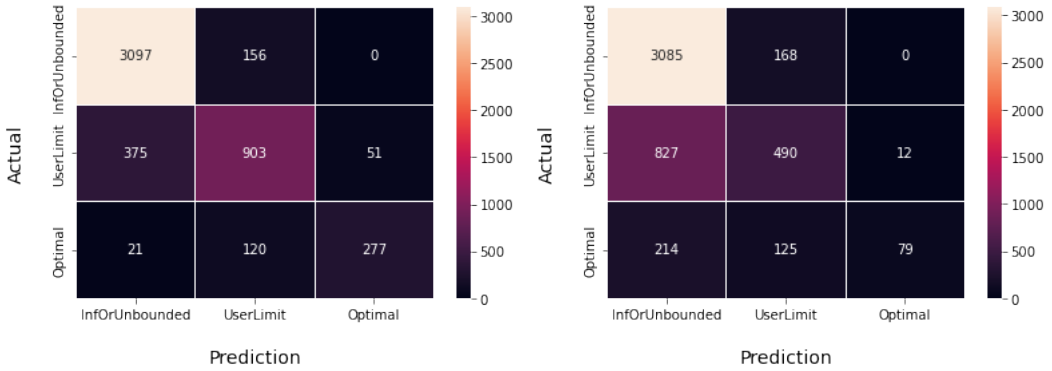

Figure 6.5: Left: Confusion matrix showcasing the predicted versus actual classes for satisfiability prediction using the SCIP features. Right: Predicted versus actual classes using the CPLEX-based features. Both figures show predictions for the default configuration.

One might assume that if predicting the satisfiability of an instance is possible with high accuracy using some set of features, then using these same features we would be able to also successfully predict whether an instance times out or not. However, we find that this is not necessarily the case. Figure 6.6 shows how test accuracies and F1 scores predict whether an instance times out changes as the user set time-out changes. Test accuracies stay over 80% which is fairly high, but lower than the test accuracies obtained using SCIP features for predicting satisfiability. Moreover, the F1 scores for most of the timeout levels are below 70%, which indicates that the imbalance in the data may skew the predictions.

SCIP features	Config	1	2	3	Default
	Accuracy	0.9826	0.9848	0.9794	0.9831
	F1	0.89202	0.8931	0.887139	0.92051
	Accuracy with timeouts	0.8632	0.8792	0.8728	0.854

CPLEX features	Config	1	2	3	Default
	Accuracy	0.9366	0.9438	0.9253	0.915
	F1	0.476	0.4662	0.4388	0.4526
	Accuracy with timeouts	0.7716	0.7948	0.7852	0.7322

Table 6.2: Accuracies and F1 scores with timeouts removed, as well as accuracies including timeouts as a predictive class for all four configurations of the $\mathrm{SDP_dMLP_A}$ dataset. We show results for both feature extractors.

6.3.1 Utilising satisfiability prediction for NNV

The main objective of this research is to make complete neural network verification more efficient. Whilst these findings suggest positive results for a key problem in computer science, to exploit these findings for verification more work needs to be done to give concrete answers. However, we can speculate, based on this work, how this might be achieved. As we used a Random Forest

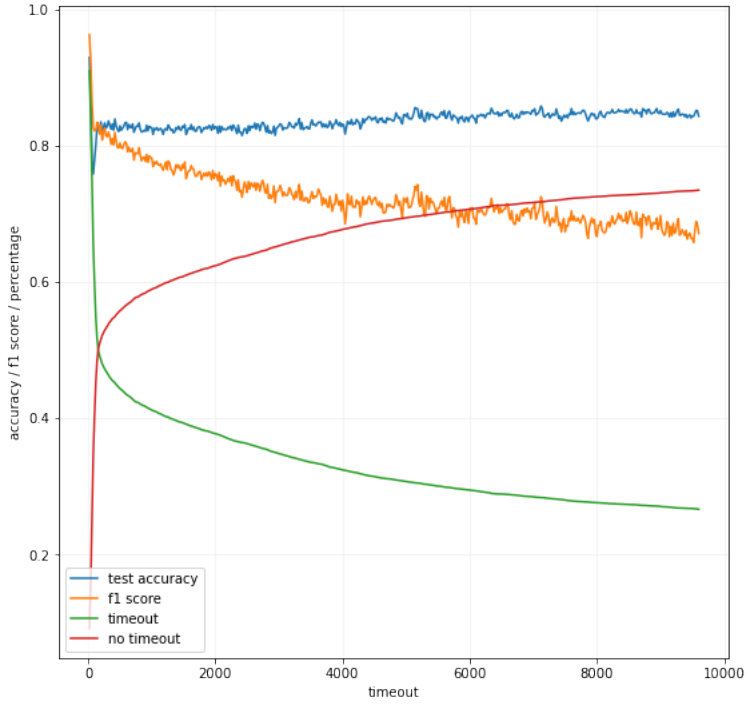

Figure 6.6: As running time changes so do the F1 scores and accuracies of our classifier for the $\mathrm{SDP_dMLP_A}$ dataset with SCIP features on the default configuration, predicting whether or not an instance times out. Here we see the proportion of timed-out versus not timed-out instances as the timeout value changes.

classifier, which is based on an ensemble of decision trees, we can get probabilities associated with the classification of an instance. This is very easy to achieve. With these predictions, we could set a threshold such that if this threshold is reached by an instance, we classify the instance with the corresponding label; either optimal or infeasible. What this means in the context of neural network verification is that we can use these predictions to skip 'easy' instances, *i.e.* instances that pass the threshold. This will hopefully reduce the number of instances to be verified significantly thus reducing overall computation time. The main problem with this approach is that by skipping instances and automatically labelling them optimal or infeasible, we are running the risk that the classifier is wrong and thus, this verification method would be unsound. However, further work would need to go into whether this method would be unsound for any threshold value.

Another possible use for an accurate prediction of the satisfiability of a verification instance is to gain a better understanding of which instances are interesting for a verifier to solve. This information could help future verification tools focus on those instances that are more difficult to solve.

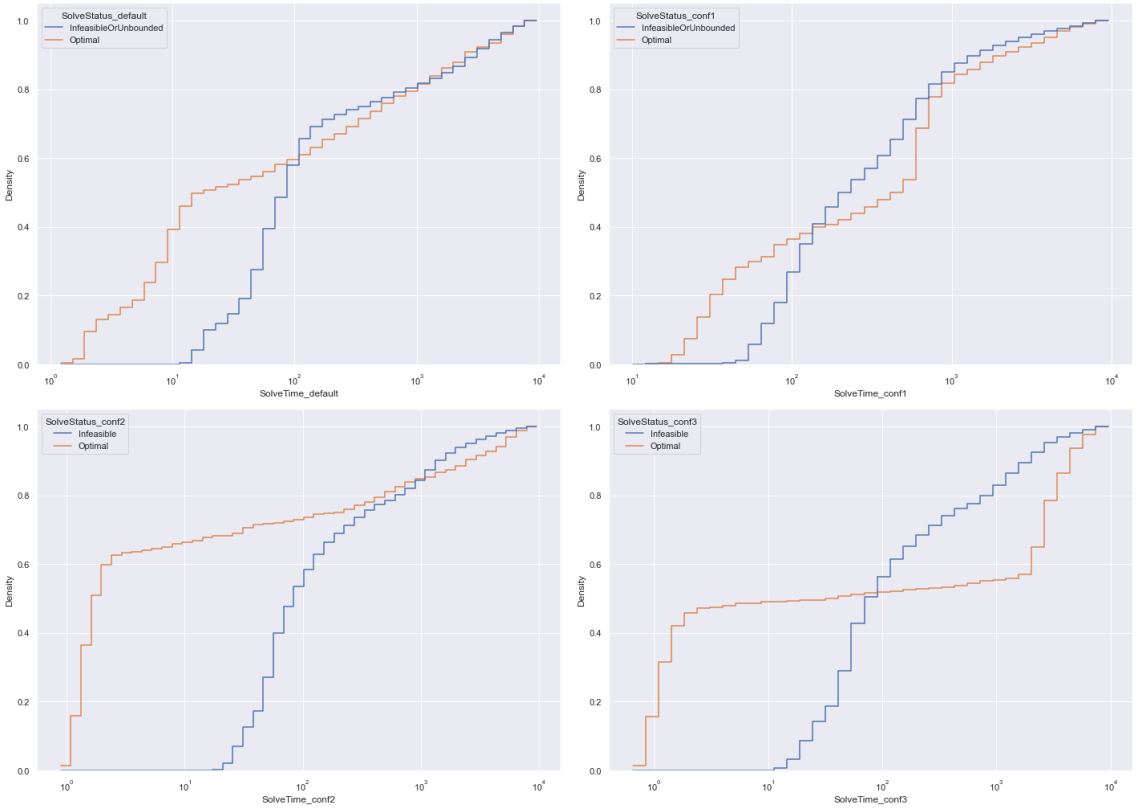

Figure 6.7: Density distributions of running time for both optimal and infeasible instances (*i.e.*, satisfiable and unsatisfiable). These plots were created using the $\mathrm{SDP_d MLP_A}$ data from [19], where the four figures represent the four different configurations.

Chapter 7

Conclusion

We summarize our findings in this chapter and reiterate our research goals. Furthermore, we present the limitations of this work and our ideas on where future research could go.

RQ1: What are relevant features of a MIP instance?
One objective of this research was to answer the question of which MIP features were important for running time and satisfiability prediction. We find that there is no individual feature strong enough to predict either running time or satisfiability, but as a whole the features found do show some predictive value. This is especially true for predicting the satisfiability of an instance with accuracies above 90% for both SCIP and CPLEX features with timeouts removed. In particular, dynamic features played an important role in both running time and satisfiability prediction. In the case of running time prediction, the absence of a cluster of instances predicted close to the actual running times using static features further illustrated the need for dynamic features. Lastly, whilst MIP features work well for SAT prediction, this is not necessarily the case for running time. Compared to previous work on using MIP features for running time prediction [16], showing RMSE scores below 0.5, we obtain an RMSE of 0.686 and 0.92 for the SCIP and CPLEX features, respectively. This demonstrates that MIP features can predict some general aspects of running time but struggle to be precise.

RQ2: Does per-instance algorithm selection using MIP features further reduce the computational cost for MIP-based neural network verification?
In this thesis, our primary research objective is to see if per-instance algorithm selection could be beneficial to MIP-based neural network verification by making it more efficient. We consider two different MIP feature extractors to create different ASlib scenarios for the automatically configured algorithm selection tool AutoFolio. We find that using MIP features to predict running time and thus select a configuration, is not enough to improve on simply using a single solver configuration. Over every scenario tried, the results were either on par or worse than the SBS for both CPLEX and SCIP-based features. A reason for this could be that MIP features can generally predict the performance of a solver configuration, as discussed in **RQ1**, but fail to be precise enough for algorithm selection. It is also possible that it can predict the performance of an instance reasonably well, but is not able to differentiate between the different solver configurations. Further feature selection may lead to slightly improved performance, however, this improvement would likely not be large enough for this method to be significantly better than the SBS.

RQ3: How well do MIP features predict the satisfiability of a verification instance?
A silver lining of this work is that MIP features can predict the satisfiability of a verification instance quite well, with accuracies reaching higher than 97% with the SCIP features found and higher than 90% with the CPLEX-based features, with timeouts removed. An interesting finding is that the timed-out instances decrease these accuracies by about 10% for SCIP and 20% for CPLEX features. This suggests that timeouts negatively impact the prediction of whether an instance is satisfiable. We see that this effect of timeouts hurts the predictions made by the CPLEX features more than the SCIP features. Considering that there are many more SCIP features, these results are not so surprising as they could cover more ground in fully explaining the solvability of an instance.

Looking into **RQ3.1**, which states how this finding can be used in the NNV domain, we can only speculate as to how it can be operationalised. We consider the potential of using a random forest model to obtain fast estimates of the robustness of a verification instance. However, it is unclear how these estimates could be incorporated into a sound verification method. Another option for this work is to use it to understand and explain verification instances better.

7.1 Limitations

Whilst this work has shown many findings that could lead to potentially useful results for neural network verification, we also see some clear limitations. First of all, the number of networks used was 1 to 2. Due to a problem with the mnistnet data retrieved from [19], which we describe in Chapter 5, we could not create the ASlib scenarios and therefore could not get results from AutoFolio. By relying on data produced by prior research we have no control over the solver configurations. This could also be why we are not able to select between the different solver configurations, as they are possibly too similar to one another. The problem of relying on prior research has left us with essentially only 1 network to verify with 1 verification method; MIPVerify. As we only consider MIPVerify as a complete verification tool, we may miss out on other findings for this method for different types of verification approaches. Additionally, we only considered AutoFolio as an algorithm selector, however, it would be interesting to see if other approaches produce different results.

7.2 Future Research

As mentioned above, we find that MIP features alone cannot be used to select a solver configuration from a portfolio. This of course raises the question of if other features might be more successful. Research into other kinds of features may yield better performance. Another possibility is that the differences between the solver configurations are too small to be able to select between them and MIP features do have some predictive power for running time. Experimenting with different solver configurations in the portfolio, and looking at how selection results change, could prove whether this is the reason why algorithm selection doesn't work for these verification instances. Lastly, an alternative to building a portfolio of algorithms is to consider different verification tools as the algorithms to select.

An initially unintentional finding in this work, that of accurate predicting satisfiability, showcases that MIP features still have much to offer. Future work could look into the suggestions that were made in Chapter 6. That is, using the associated probabilities a random forest classifier produces to rank instances from least to most optimal and offering a better understanding of the solvability of verification instances. In these ways, we may be able to operationalise these findings for more efficient verification.

References

[1] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[2] C. Szegedy, W. Zaremba, I. Sutskever, *et al.*, "Intriguing properties of neural networks," in *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

[3] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[4] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of the 37th IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 582–597.

[5] X. Weilin, E. David, and Q. Yanjun, "Feature squeezing: Detecting adversarial examples in deep neural networks," *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS)*, 2018.

[6] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 39–57.

[7] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 274–283.

[8] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security (ASIACCS)*, 2017, pp. 506–519.

[9] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," *Advances in Neural Information Processing Systems 29 (NIPS)*, pp. 2613–2621, 2016.

[10] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV)*, 2010, pp. 243–257.

[11] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *Proceedings of the 29th International Conference on Computer Aided Verification (CAV)*, 2017, pp. 97–117.

[12] G. Katz, D. A. Huang, D. Ibeling, *et al.*, "The marabou framework for verification and analysis of deep neural networks," in *Proceedings of the 31st International Conference on Computer Aided Verification (CAV)*, 2019, pp. 443–452.

[13] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[14] S. Wang, H. Zhang, K. Xu, *et al.*, "Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification," in *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021, pp. 29 909–29 921.

[15] L. Li, T. Xie, and B. Li, "Sok: Certified robustness for deep neural networks," in *Proceedings of the 44th IEEE Symposium on Security and Privacy (SP)*, 2023.

[16] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014, ISSN: 0004-3702.

[17] J. R. Rice, "The algorithm selection problem," in *Advances in computers*, vol. 15, 1976, pp. 65–118.

[18] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, pp. 1–25, 2009.

[19] M. König, H. H. Hoos, and J. N. v. Rijn, "Speeding up neural network robustness verification via algorithm configuration and an optimised mixed integer linear programming solver portfolio," *Machine Learning*, vol. 111, no. 12, pp. 4565–4584, 2022.

[20] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[21] I. E. Grossmann, "Review of nonlinear mixed-integer and disjunctive programming techniques," *Optimization and Engineering*, vol. 3, pp. 227–252, 2002.

[22] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener, "Efficient verification of relu-based neural networks via dependency analysis," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, vol. 34, 2020, pp. 3291–3299.

[23] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018, pp. 6369–6379.

[24] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019, pp. 1310–1320.

[25] L. Weng, H. Zhang, H. Chen, *et al.*, "Towards fast computation of certified robustness for ReLU networks," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, vol. 80, 2018, pp. 5276–5285.

[26] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2017.

[27] C. Tjandraatmadja, R. Anderson, J. Huchette, W. Ma, K. K. Patel, and J. P. Vielma, "The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification," in *Advances in Neural Information Processing Systems 33 (NeurIPS)*, vol. 33, 2020, pp. 21 675–21 686.

[28] A. Raghunathan, J. Steinhardt, and P. S. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," in *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018, pp. 10 900–10 910.

[29] S. Dathathri, K. Dvijotham, A. Kurakin, *et al.*, "Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming," in *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020, pp. 5318–5331.

[30] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, 2019.

[31] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018, pp. 4944–4953.

[32] K. Xu, H. Zhang, S. Wang, *et al.*, "Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[33] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu, "First three years of the international verification of neural networks competition (vnn-comp)," *International Journal on Software Tools for Technology Transfer*, pp. 1–11, 2023.

[34] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2023.

[35] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2017, pp. 251–268.

[36] R. Bunel, I. Turkaslan, P. H. S. Torr, M. P. Kumar, J. Lu, and P. Kohli, "Branch and bound for piecewise linear neural network verification," *Journal of Machine Learning Research*, vol. 21, no. 1, 2020, ISSN: 1532-4435.

[37] B. A. Huberman, R. M. Lukose, and T. Hogg, "An economics approach to hard computational problems," *Science*, vol. 275, no. 5296, pp. 51–54, 1997, ISSN: 00368075, 10959203.

[38] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *Journal of Artificial Intelligence Research*, vol. 32, pp. 565–606, 2008.

[39] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham, "Understanding random sat: Beyond the clauses-to-variables ratio," in *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP)*, 2004, pp. 438–452.

[40] M. T. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub, "AutoFolio: an automatically configured algorithm selector," *Journal of Artificial Intelligence Research*, vol. 53, pp. 745–778, 2015.

[41] L. Xu, H. H. Hoos, and K. Leyton-Brown, "Hydra: Automatically configuring algorithms for portfolio-based selection," in *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, 2010, pp. 210–216.

[42] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming," in *Proceedings of the 18th RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA)*, 2011, pp. 16–30.

[43] C. Ferrari, M. N. Mueller, N. Jovanović, and M. Vechev, "Complete verification via multi-neuron relaxation guided branch-and-bound," in *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

[44] H. Zhang, S. Wang, K. Xu, *et al.*, "General cutting planes for bound-propagation-based neural network verification," 2022, pp. 1656–1670.

[45] L. Xu, H. H. Hoos, and K. Leyton-Brown, "Predicting satisfiability at the phase transition," in *Proceedings of the 26th International Conference on Artificial Intelligence (AAAI)*, 2012.

[46] A. Biere, M. Heule, and H. van Maaren, Eds., *Handbook of Satisfiability* (Frontiers in Artificial Intelligence and Applications volume 336), Second edition. Amsterdam ; Washington, DC: IOS Press, 2021, ISBN: 978-1-64368-160-3.

[47] K. Bestuzheva, M. Besançon, W.-K. Chen, *et al.*, "The SCIP Optimization Suite 8.0," Zuse Institute Berlin, ZIB-Report 21-41, Dec. 2021.

[48] A. Georges, A. Gleixner, G. Gojic, *et al.*, "Feature-based algorithm selection for mixed integer programming," Zuse Institute Berlin, ZIB-Report 18-17, 2018.

[49] A. Gleixner, G. Hendel, G. Gamrath, *et al.*, "Miplib 2017: Data-driven compilation of the 6th mixed-integer programming library," *Mathematical Programming Computation*, vol. 13, no. 3, pp. 443–490, 2021.

[50] H. H. Hoos, M. Lindauer, and T. Schaub, "Claspfolio 2: Advances in algorithm selection for answer set programming," *Theory and Practice of Logic Programming*, vol. 14, no. 4-5, pp. 569–585, 2014.

[51] M. Lindauer, K. Eggensperger, M. Feurer, *et al.*, "Smac3: A versatile bayesian optimization package for hyperparameter optimization," *Journal of Machine Learning Research*, vol. 23, no. 54, pp. 1–9, 2022.

[52] B. Bischl, P. Kerschke, L. Kotthoff, *et al.*, "ASlib: A benchmark library for algorithm selection," *Artificial Intelligence*, vol. 237, pp. 41–58, 2016, ISSN: 0004-3702.

[53] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[54] G. Hendel, D. Anderson, P. Le Bodic, and M. E. Pfetsch, "Estimating the size of branch-and-bound trees," *INFORMS Journal on Computing*, vol. 34, no. 2, pp. 934–952, 2022.

[55] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Appendix A

Additional material

In this section, all additional results, including figures and tables will be presented. These results are additional material and would have otherwise taken up too much space in Chapter 6.

A.1 Running time prediction

The following figures are the leftover running time prediction plots for all four configurations for different sets of features.

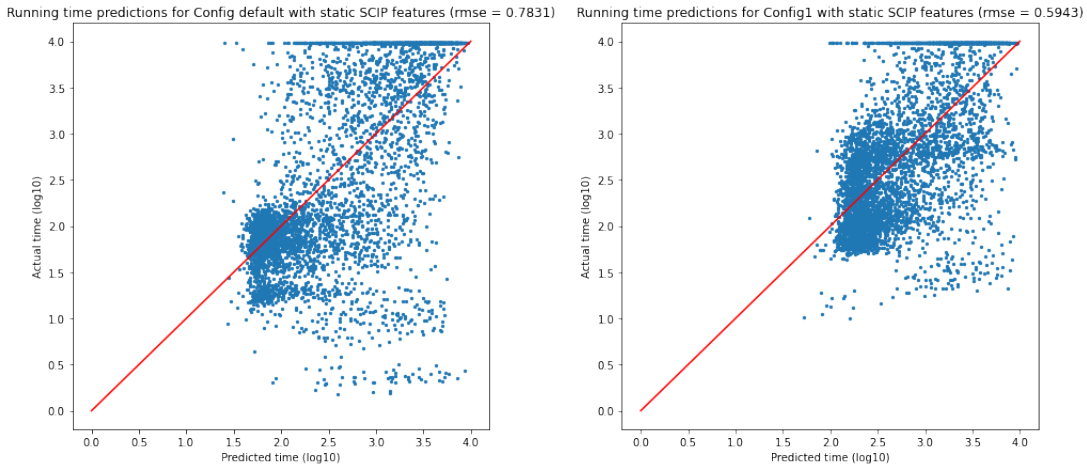

Figure A.1: Running time prediction of the default configuration and configuration 1 of the $\mathrm{SDP_dMLP_A}$ network using static SCIP features.

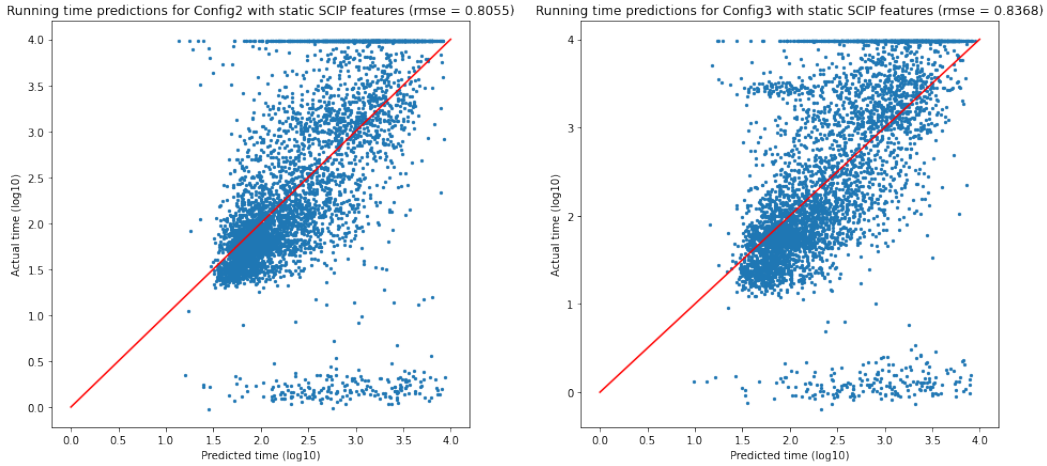

Figure A.2: Running time prediction of configurations 2 and 3 of the $\mathrm{SDP_dMLP_A}$ network using static SCIP features.

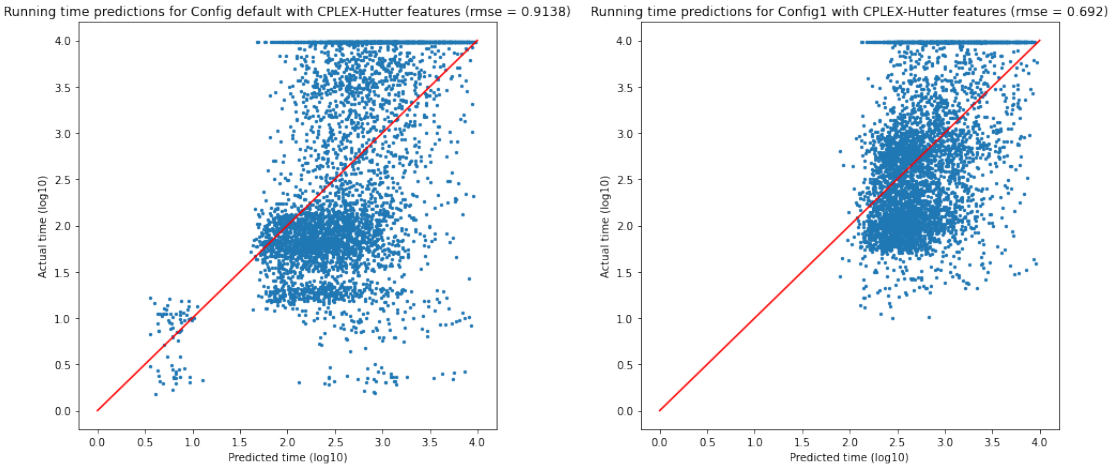

Figure A.3: Running time prediction of the default configuration and configuration 1 of the $\mathrm{SDP_dMLP_A}$ network using the CPLEX-based features.

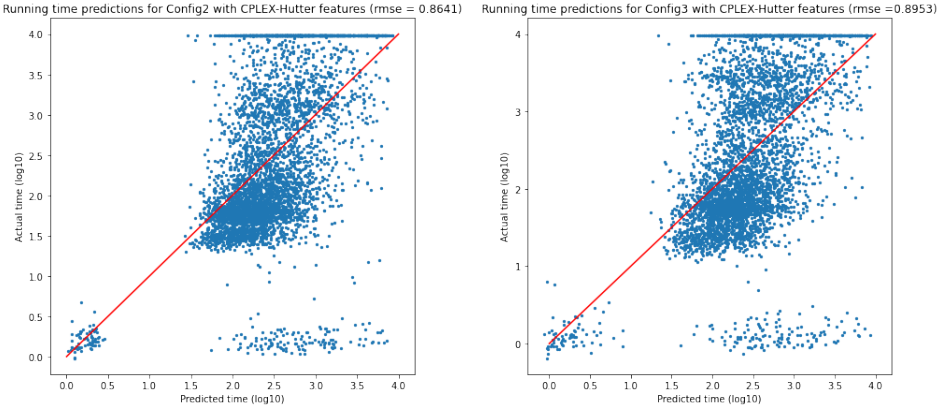

Figure A.4: Running time prediction of configurations 2 and 3 of the $\mathrm{SDP_dMLP_A}$ network using the CPLEX-based features.

A.2 Principal component analysis

Similar to Figure 6.3, we show in Figure A.5 the same plots but for CPLEX features. Note that the found clusters of infeasible versus optimal instances are not as clearly defined.

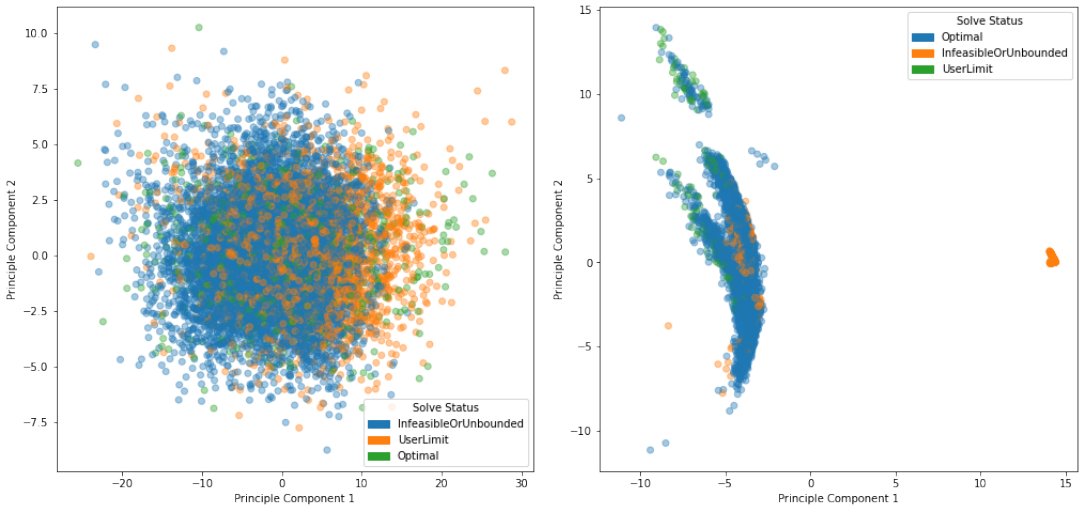

Figure A.5: Reduced dimensions using PCA with two principle components with the CPLEX-based features on both $\mathrm{SDP_dMLP_A}$ (left) and mnistnet (right) for predicting the status of an instance.

A.3 Timeout prediction

During this research project, predicting whether or not an instance will time out was something that we also considered. We present all relevant figures in this section.

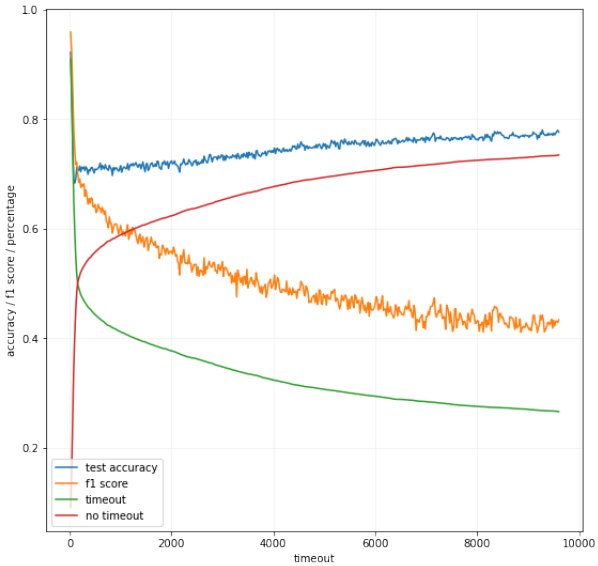

Figure A.6: As running time changes so do the F1 scores and accuracies of our classifier for the $\mathrm{SDP_dMLP_A}$ dataset with CPLEX features on the default configuration, predicting whether or not an instance times out.

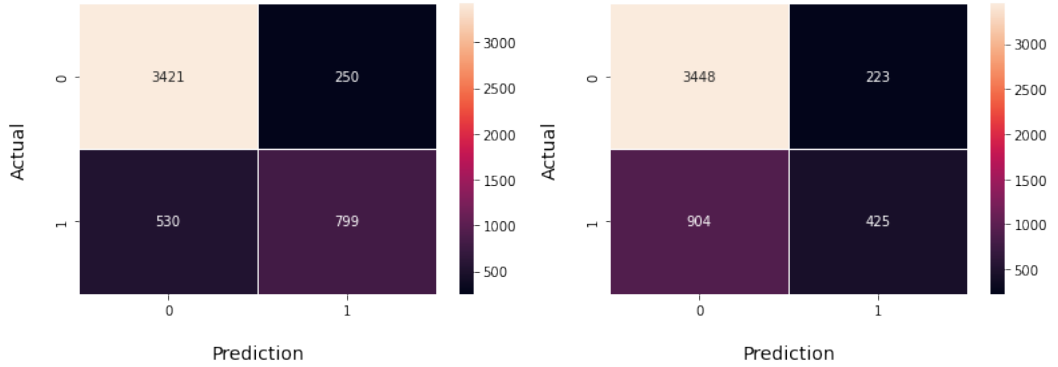

Figure A.7: Confusion matrices of predicting whether the default configuration will timeout on the $\mathrm{SDP_dMLP_A}$ dataset. Left: Using SCIP features we achieve accuracies of 0.844 and an F1 score of 0.672. Right: Using CPLEX features we achieve accuracies of 0.775 and an F1 score of 0.4299.

A.4 AutoFolio results

Further results obtained from AutoFolio can be found here.

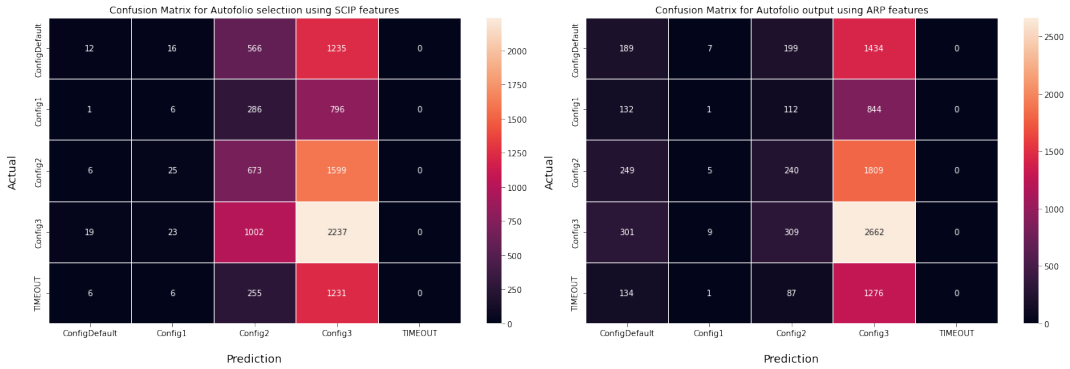

Figure A.8: Confusion matrices depicting the selections made by AutoFolio versus the actual best solver configuration for the $\mathrm{SDP_dMLP_A}$ dataset. Left: Using SCIP features we obtain an accuracy with timeouts of 0.2928 and 0.3444 without. Right: Using CPLEX features we achieve an accuracy of 0.3092 with timeouts and 0.3637, without timeouts.

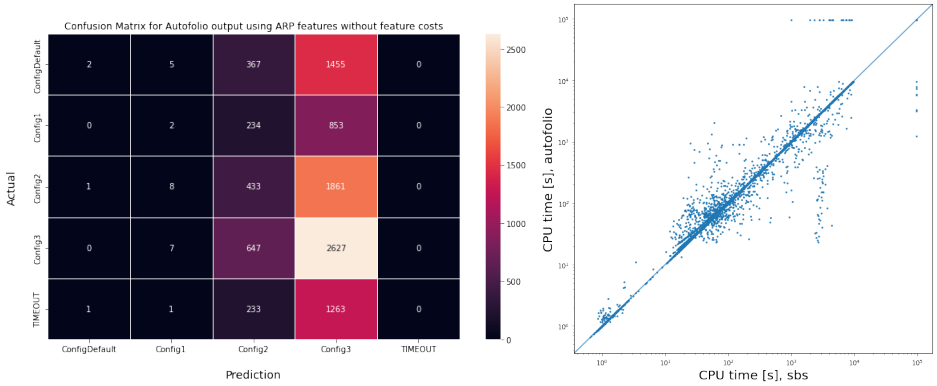

Figure A.9: Autofolio results for CPLEX features without feature costs for the $\mathrm{SDP_dMLP_A}$ dataset. Left: Confusion matrix depicting the selections made by AutoFolio versus the actual best solver configuration. We obtained an accuracy with timeouts of 0.3064 and 0.3604 without. Right: CPU time in seconds showcasing the time it took for the SBS to run for that instance versus AutoFolio.

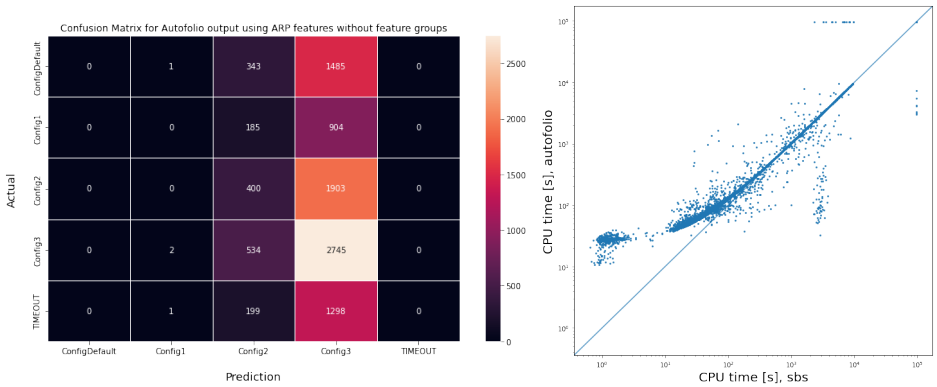

Figure A.10: Autofolio results for CPLEX features without feature groups for the $\mathrm{SDP_dMLP_A}$ dataset. Left: Confusion matrix depicting the selections made by AutoFolio versus the actual best solver configuration. We obtained an accuracy with timeouts of 0.3145 and 0.3699 without. Right: CPU time in seconds showcasing the time it took for the SBS to run for that instance versus AutoFolio.

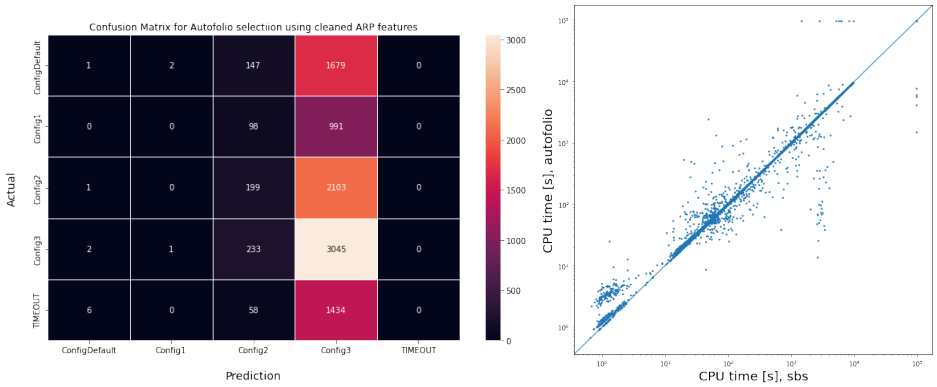

Figure A.11: AutoFolio results for cleaned CPLEX features (81) for the $\mathrm{SDP_dMLP_A}$ dataset. Left: Confusion matrix depicting the selections made by AutoFolio versus the actual best solver configuration. We obtained an accuracy with timeouts of 0.3245 and 0.3817 without. Right: CPU time in seconds showcasing the time it took for the SBS to run for that instance versus AutoFolio.

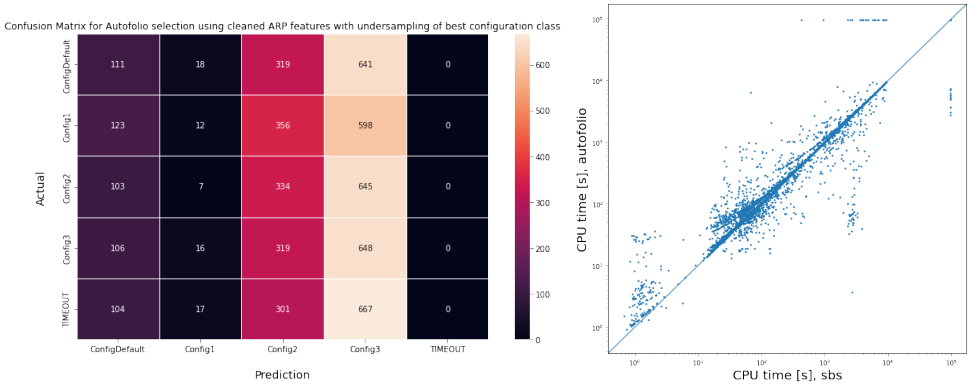

Figure A.12: AutoFolio results for cleaned CPLEX features (81) for the $\mathrm{SDP_dMLP_A}$ dataset with undersampling on the best configuration classes. Left: Confusion matrix depicting the selections made by AutoFolio versus the actual best solver configuration. We obtained an accuracy with timeouts of 0.2029 and 0.2538 without timeouts. Right: CPU time in seconds showcasing the time it took for the SBS to run for that instance versus AutoFolio.

A.5 Satisfiability prediction

In addition to reporting results on predicting satisfiability of the $\mathrm{SDP_dMLP_A}$, we do the same for the mnistnet dataset. Moreover, we show the binary classification confusion matrices.

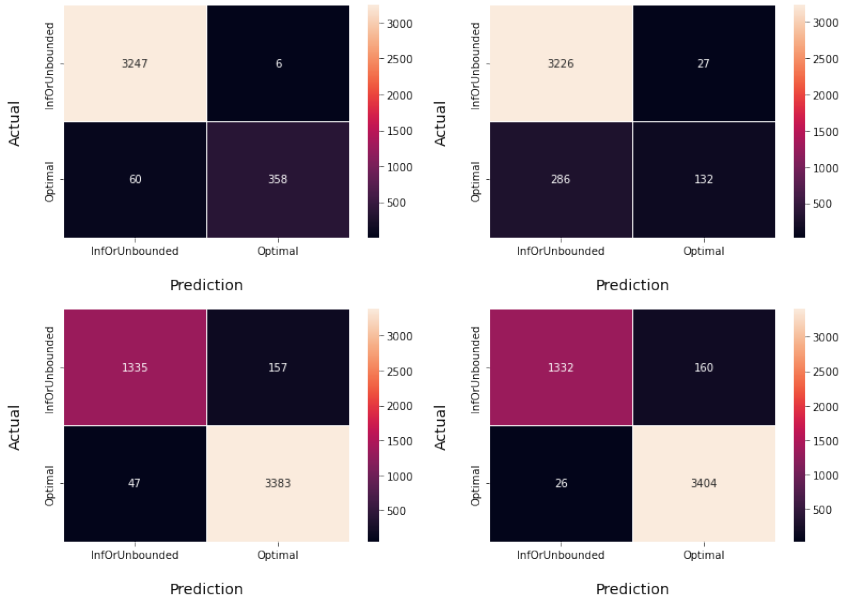

Figure A.13: Confusion matrices of satisfiability prediction of the default configuration with timeouts removed. Top: on the $\mathrm{SDP_dMLP_A}$ network. Bottom: on mnistnet network. With SCIP on the left and CPLEX features on the right.

SCIP features	Config	1	Default	**CPLEX features**	Config	1	Default
	Accuracy	0.96211	0.9454		Accuracy	0.96713	0.96363
	F1	0.9728	0.9705		F1	0.9765	0.9744
	Accuracy with timeouts	0.9232	0.95815		Accuracy with timeouts	0.9282	0.9494

Table A.1: Accuracies and F1 scores with timeouts removed, as well as accuracies including timeouts as a predictive class for both configurations of the mnistnet dataset. We show results for both feature extractors.

A.6 Histograms

The following plots convey the same information as the densities in Figure 6.7, but instead show the distributions through a histogram.

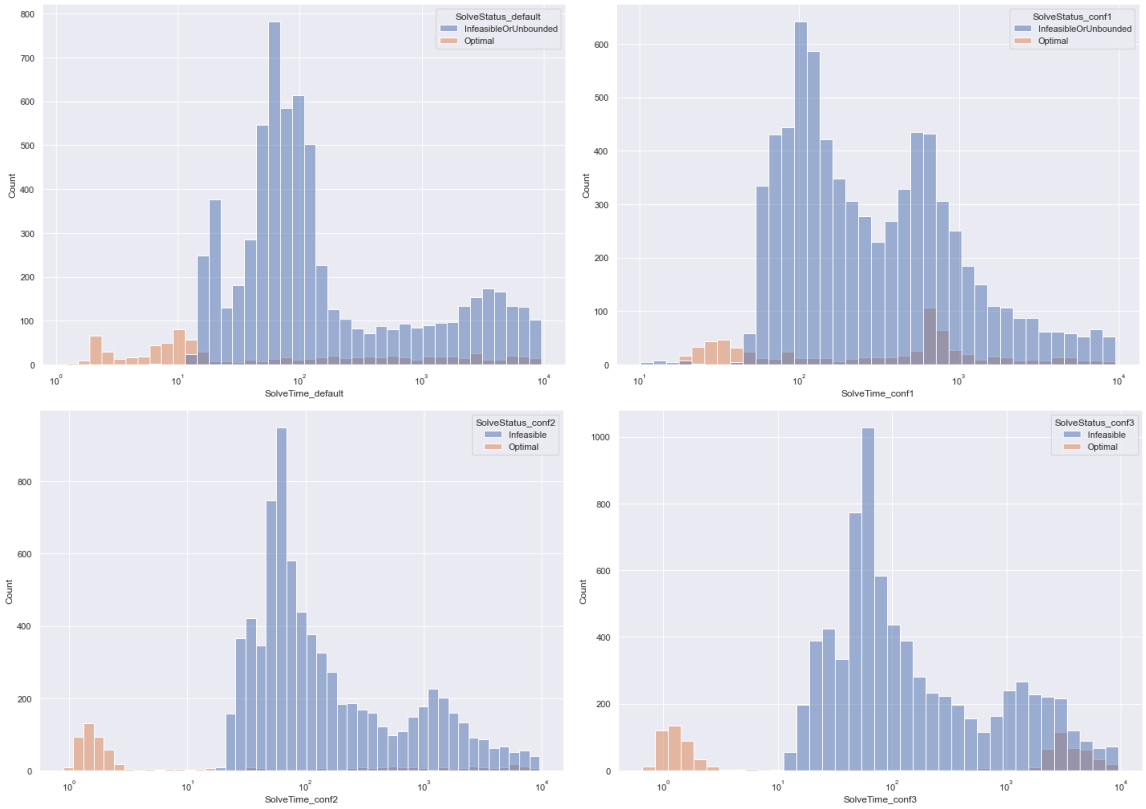

Figure A.14: Histograms of running time over both sat- and unsatisfiable instances.

A.7 Top features

In this section, we present the top 10 features found for both feature extractors on both networks for all four configurations. These top features were computed using a random forest model.

Top n features	1	2	3	4	5	6	7	8	9	10
SCIP, Config Default	LOWERBOUNDS STD presolve default	LOWERBOUNDS MIN presolve default	LOWERBOUNDS MIN presolve off	Gap	LP dualLP Iter/call	BOUNDRANGE MAX presolve default	LH CONSTR MAX presolve default	avgswitch length	CONSTR MAX presolve default	Separators gomory FoundCuts
SCIP, Config 1	LOWERBOUNDS MIN presolve default	LOWERBOUNDS STD presolve default	LOWERBOUNDS MIN presolve off	LP dualLP Iter/call	PrimalBound	Presolvers milp ChgBounds	LH CONSTR MAX presolve default	BOUNDRANGE MAX presolve default	CONSTR MAX presolve default	ConflictAnalysis pseudosolution Calls
SCIP, Config 2	LOWERBOUNDS STD presolve default	Gap	LOWERBOUNDS MIN presolve off	LOWERBOUNDS MIN presolve default	ConflictAnalysis pseudosolution Calls	Presolvers milp ChgBounds	PrimalBound	LH CONSTR MAX presolve default	LP dualLP Iter/call	CONSTR MAX presolve default
SCIP, Config 3	LOWERBOUNDS STD presolve default	ConflictAnalysis pseudosolution Calls	LOWERBOUNDS MIN presolve default	LOWERBOUNDS MIN presolve off	Presolvers milp ChgBounds	LP dualLP Iter/call	avgswitch length	LH CONSTR MAX presolve default	BOUNDRANGE MAX presolve default	CONSTR MAX presolve default
CPLEX, Config Default	rhs_c_1_avg	rhs_c_0_avg	mipgap	lp_l2_avg	rhs_c_1_varcoef	vcg_var_weight1_avg	rhs_c_2_avg	vcg_var_weight1_varcoef	rhs_c_0_varcoef	cplex_prob_time
CPLEX, Config 1	vcg_var_weight1_varcoef	rhs_c_1_avg	vcg_var_weight1_avg	rhs_c_0_avg	lp_l2_avg	rhs_c_0_varcoef	rhs_c_2_avg	clique table	rhs_c_1_varcoef	cplex_prob_time
CPLEX, Config 2	mipgap	rhs_c_1_avg	vcg_var_weight1_varcoef	vcg_var_weight1_avg	rhs_c_0_avg	lp_l2_avg	rhs_c_0_varcoef	rhs_c_1_varcoef	rhs_c_2_avg	vcg_constr_weight2_varcoef
CPLEX, Config 3	mipgap	rhs_c_1_avg	vcg_var_weight1_varcoef	rhs_c_0_avg	vcg_var_weight1_avg	rhs_c_0_varcoef	lp_l2_avg	clique table	rhs_c_1_varcoef	rhs_c_2_avg

Table A.2: The most important features for each of the configurations for both feature extractors for the $\mathrm{SDP_dMLP_A}$ dataset. Feature rankings were obtained using a random forest regression model for running time prediction.

Top n features	1	2	3	4	5	6	7	8	9	10	Accuracy	f1 score
SCIP, Config Default with Timouts	CONSTR MAX presolve default	LH CONSTR MAX presolve default	LOWERBOUNDS STD presolve default	BOUNDRANGE MAX presolve default	LOWERBOUNDS MIN presolve default	Presolvers milp ChgBounds	LP dualLP Iter/call	LP strongbranching 2 Iterations	LP strongbranching 2 Calls	LP strongbranching 2 ItLimit	0.8554	-
SCIP, Config Default without Timeouts	GapLastSol	BranchingRules relpscost ExecTime	ConstraintTimings integral1 TotalTime	LP strongbranching 2 Iterations	FirstSolution	Constraints benders 1 Check	LP strongbranching 2 Calls	LH CONSTR MAX presolve default	Solutionsfound	PrimalBound	0.9831	0.9209
CPLEX, Config Default with Timeouts	lp_l2_avg	lp_avg	vcg_var_weight1_avg	rhs_c_0_avg	cplex_prob_time	vcg_var_weight1_varcoef	rhs_c_0_varcoef	rhs_c_1_varcoef	rhs_c_1_avg	vcg_var_weight2_avg	0.7308	-
CPLEX, Config Default without Timeouts	mipgap	lp_l2_avg	cplex_prob_time	lp_avg	rhs_c_0_varcoef	vcg_var_weight1_varcoef	vcg_var_weight1_avg	rhs_c_0_avg	vcg_constr_weight0_avg	vcg_var_weight0_avg	0.9147	0.4575

Table A.3: The most important features for the default configuration for both feature extractors for the $\mathrm{SDP_dMLP_A}$ dataset, with and without timeout instances. Feature rankings were obtained using a random forest classification model for satisfiability prediction. Additionally, we report the corresponding accuracies and F1 scores of each of the models.

Top n features	1	2	3	4	5	6	7	8	9	10	Accuracy	f1 score
SCIP, Config Default with Timouts	Integrals primal-dual Avg%	PrimalHeuristics intshifting ExecTime	PrimalHeuristics alns Calls	Gap	Integrals primal-dual Total	PrimalHeuristics zirounding Calls	Separators cutpool FoundCuts	PrimalHeuristics feaspump ExecTime	PrimalHeuristics intshifting Calls	Constraints varbound 1 Separate	0.9444	-
SCIP, Config Default without Timeouts	PrimalHeuristics alns Calls	FinalDualBound	PrimalHeuristics intshifting Calls	PrimalHeuristics aln ExecTime	Constraints integral Children	PrimalHeuristics intshifting ExecTime	Integrals primal-dual Avg%	Separators mcf Calls	Integrals primal-dual Total	RootLPEstimate	0.9586	0.9707
CPLEX, Config Default with Timeouts	new_col	vcg_constr_weight1_avg	new_row	lp_linf	vcg_var_weight1_avg	vcg_constr_weight1_varcoef	cplex_prob_time	lp_objval	time_relax	vcg_var_weight2_avg	0.9472	-
CPLEX, Config Default without Timeouts	new_row	new_col	lp_objval	vcg_var_weight2_avg	vcg_constr_weight2_avg	cplex_prob_time	vcg_constr_weight1_avg	lp_linf	vcg_constr_weight1_varcoef	vcg_var_weight1_avg	0.9622	0.9734

Table A.4: The most important features for the default configuration for both feature extractors for the $\mathrm{mnistnet}$ dataset, with and without timeout instances. Feature rankings were obtained using a random forest classification model for satisfiability prediction. Additionally, we report the corresponding accuracies and F1 scores of each of the models.