TU WIEN Informatics

Algorithm Selection using Machine Learning for Sudoku Puzzles

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Rajwardhan Kumar
Matrikelnummer 11936024

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Nysret Musliu

Wien, 27. Oktober 2021

Rajwardhan Kumar	Nysret Musliu

TU WIEN Informatics

Algorithm Selection using Machine Learning for Sudoku Puzzles

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Rajwardhan Kumar
Registration Number 11936024

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.-Doz. Dr. Nysret Musliu

Vienna, 27th October, 2021

Rajwardhan Kumar	Nysret Musliu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Rajwardhan Kumar

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27. Oktober 2021

Rajwardhan Kumar

Acknowledgements

I would like to extend my thanks to my friends and family for their unconditional support and love. I would like to to extend my thanks to DI Felix Winter for his recommendations on Gurobi solver and the running of the Hybrid Sudoku solver in the initial stages of the thesis. I would also like to thank Matthias Steckert and Nicola Nasi for their help in grammatically correcting the German version of the Abstract. Most importantly, I would like to extend my special thanks to my supervisor Priv.-Doz. Dr. Nysret Musliu for his recommendations, directions and support throughout the thesis work.

Kurzfassung

Sudokus sind kombinatorische Zahlenrätsel, welche regelmäßig in Zeitschriften, Zeitungen und auf Webseiten erscheinen und zum Brainstorming gut geeignet sind. Diese kombinatorischen Zahlenrätsel sind als eine quadratische Matrix mit Zeilen-, Spalten- und Untergitterbeschränkungen definiert, wobei die Untergitter als die Quadratwurzel der Matrixgröße definiert sind. Im Allgemeinen sind Sudoku-Rätsel Spezialfälle von sogenannten lateinischen Quadraten.

Sudoku-Rätsel sind rechnerisch NP-komplette Probleme, d.h. nach dem No-Free-Lunch-Theorem gibt es keinen einzigen Algorithmus, der alle Instanzen der Sudoku-Rätsel am effizientesten löst. Daraus ergibt sich die Frage zur Algorithmenauswahl, bei der nicht für jedes Rätsel ein neuer Algorithmus geschrieben werden muss, sondern der beste Algorithmus aus einer gegebenen Liste von Algorithmen mit guter Leistung ausgewählt werden kann. Darüber hinaus ist es möglicherweise auch wichtig, die Laufzeit von Algorithmen für jedes Sudoku-Rätsel abzuschätzen.

In dieser Arbeit untersuchen wir das Problem der Algorithmenauswahl und Laufzeitvorhersage für Sudoku-Rätsel mit Methoden des maschinellen Lernens. Wir identifizieren vier Sudoku-Solver nach aktuellem Stand der Technik, um die Problemstellung der Algorithmenauswahl und Laufzeitvorhersage anzugehen. Zusätzlich wurde die Berechnung von 70 Merkmalen durchgeführt. Für jede Fragestellung wurden fünf Klassifizierungs- und sechs Regressionsmethoden, zusammen mit der Merkmalselektion, der Datendiskretisierung und verschiedenen Parametereinstellungen untersucht.

Abschließend lässt sich sagen, dass die von uns erstellte Umgebung - bezogen auf die Frage der statischen Algorithmenauswahl und Laufzeitvorhersage für Sudoku-Rätsel – robust in Bezug auf Genauigkeit der Algorithmenauswahl und gut bei der Vorhersage der Laufzeit ist.

Abstract

Sudokus are interesting puzzles for brainstorming and appear regularly in magazines, papers and websites. These puzzles are defined as a square matrix having a row, column and sub-grid constraints, where the sub-grids are the square root of the matrix size. In general, Sudoku puzzles are special cases of Latin squares.

Computationally, Sudoku puzzles are NP-complete problems, i.e., according to the No Free Lunch Theorem, there is no single algorithm that solves all instances of the Sudoku puzzles most efficiently. This gives rise to the question of algorithm selection, where there is no requirement of writing a new algorithm for every puzzle but the best performing algorithm can be chosen from a given set of good performing algorithms. Moreover, it may also be of importance knowing how long to run an algorithm.

In this thesis, we investigate the algorithm selection and run-time prediction problem for Sudoku puzzles using machine learning methods. We identify four state-of-the-art Sudoku solvers for algorithm Selection and run-time Prediction. Additionally, identification and computation of 70 features for characterizing Sudoku puzzles distinctively was done. Five classification and six regression methods for each of the problem statements have been investigated along with feature selection method, discretization of data and various parameter settings.

Finally, the environment built by us for algorithm selection and run-time prediction for Sudoku puzzles was robust in terms of accuracy for algorithm selection and good in predicting the run-time.

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
1.1 Objectives . . . 2
1.2 Results . . . 2
1.3 Organisation of Chapters . . . 3

2 Background **5**
2.1 Sudoku . . . 5
2.2 Algorithms for Sudoku Puzzle . . . 7
2.3 Algorithm Selection and Run-time Prediction . . . 11

3 Algorithm Selection and Run-time Prediction for Sudoku Puzzles **15**
3.1 Sudoku Instances . . . 15
3.2 Solvers for Sudoku . . . 16
3.3 Features for Sudoku Puzzle . . . 17
3.4 Method for Algorithm Selection . . . 22
3.5 Method for Run-time Prediction . . . 23

4 Experimental Setup, Solver Evaluation and Data Sets **25**
4.1 Solver Run-time Evaluation . . . 25
4.2 Data Sets for Algorithm Selection and Run-time Prediction . . . 31

5 Evaluation of Algorithm Selection **33**
5.1 Pre-processing and Feature Selection . . . 33
5.2 Algorithm Selection Results . . . 34
5.3 Overall Analysis . . . 42

6 Evaluation of Run-time Prediction **45**
6.1 Pre-processing . . . 45

6.2 Regression Results 46
6.3 Overall Analysis 50

7 Conclusion **55**
7.1 Future Work 56

List of Figures **57**

List of Tables **61**

List of Algorithms **63**

Acronyms **65**

Bibliography **67**

Appendix A **73**
Features 73
Algorithm Selection Feature Selection 73

Appendix B **77**
Run-time Prediction Feature Selection 77

Appendix C **81**
Algorithm Selection for Easy Data Set 81

CHAPTER 1

Introduction

Sudoku puzzles are regularly published in papers, magazines and online. Many mobile apps and websites have been developed for Sudoku puzzles [1] [2] [3]. This puzzle has also grabbed the interest of many mathematicians. Puzzles have always interested mathematicians, for example the seven bridges of Königsberg and Euler [RT12]. Interestingly, origins of Sudoku lie with Euler's development of Latin squares, which is a number puzzle having row and column constraints [Lew07a]. Sudoku has been shown to have uses in binary data encryption [WNA10] and image encryption [WZN+10].

Sudoku as presented in papers, magazines and online are generally of order 3 and are well formed. Typically, a well formed Sudoku puzzle corresponds to one Sudoku board and is logic solvable but it is worth noting that not all puzzles will be logic solvable [YS03, Lew07b]. Sudoku has been proven to be a NP-Complete puzzle [YS03].

Computationally, Sudoku is modelled as a constraint program [Sim05], in conjunctive normal form [LO06], as an integer program [BCLR08], etc. The above mentioned papers have not involved puzzles of higher order, while state-of-the-art meta-heuristic approaches such as [Lew07a] and [MW17] have investigated puzzles up to the order of 5. Though, there are other papers that have also involved puzzles of higher order. Our aim is to investigate puzzles of higher order and also include constraint optimisation and integer programming state-of-the-art solvers like IBM Cplex, Google OR-Tools, Gurobi and Chuffed. The performance of these solvers vary on a case by case basis of instances which justify the *no free lunch theorem* [WM97].

The NP-Completeness of Sudoku leads us to the problem of *algorithm selection* by *no free lunch theorem* [WM97, Ric76], where the need to create new algorithms for each instance of a problem is no longer required and a best performing algorithm can be selected from

[1] http://sudoku.com/
[2] https://www.websudoku.com/
[3] https://www.soduko-online.com/

an existing set of algorithms [Kot16]. The set of best performing algorithms is called the algorithm space [Ric76].

Algorithm space is chosen on the basis of performance by making an evaluation over a set of existing puzzle instances, which is the problem space [Ric76] of Sudoku puzzles. Therefore, the need to generate puzzles. Additionally, puzzles of order 3 to 9 for the purpose has been generated . The puzzles generated are described in [Lew07a].

Based on the problem space, features are computed from each puzzle instance as domain knowledge or meta-learning features, this is the feature space [Ric76]. Additionally, Rice model has a performance space [Ric76]. The use of machine learning is to enhance the performance space using various state-of-the-art machine learning algorithms. The other state-of-the-art algorithm selection papers are [XHHLB08, MS13, Kot16]. Feature space is one of the most important aspect of algorithm selection [Kot16].

Based on the algorithm space, it may also be of importance knowing the run-time of algorithms [HXHLB14]. The application of machine learning techniques has also been applied to predict the run-time of algorithms.

This thesis selects the most efficient algorithm for a certain Sudoku puzzle instance from a set of good performing algorithms.

1.1 Objectives

The main objectives of the thesis are as follows:

- Identification of solvers, computing run-time for each Sudoku puzzle instance w.r.t. each solver and analyzing and comparing the solvers.
- Identify and compute features using domain and meta-learning knowledge for the Sudoku puzzle instances.
- Perform algorithm selection for Sudoku puzzle instances based on supervised machine learning classification techniques using the computed features. Here, we wish to obtain a very high accuracy and the goal would be to select the best performing solver.
- Perform algorithm run-time prediction for Sudoku puzzle instances based on supervised machine learning regression techniques using the computed features.

1.2 Results

Our contribution to algorithm selection and run-time prediction of Sudoku puzzles are as follows:

- We identified six state-of-the-art solvers for Sudoku instances. Based on run-time of Sudoku puzzle instances, we chose four solvers. Additionally, we were experimentally able to show that no solver out performs for every puzzle instance.
- We identified and computed 70 features for algorithm selection and run-time prediction.
- The goal of selecting the best performing solver was achieved for some cases. An accuracy of almost 90% was achieved for some cases.
- Regression techniques showed good results in some cases for predicted run-time of solvers on Sudoku instances.

1.3 Organisation of Chapters

The further chapters of the thesis are organised as follows:

In chapter 2 of the thesis, we describe the background knowledge. The background of Sudoku as a constraint problem is discussed. The state-of-the-art algorithms and modelling techniques that are used to solve Sudoku puzzles have been briefly discussed. Finally, the algorithm selection and run-time prediction method are presented.

In chapter 3 of the thesis, we describe the basic algorithm selection and run-time prediction approach for Sudoku puzzles. The method used for generating the Sudoku puzzle instances has been discussed. The features identified and computed by us are presented. The brief mention of solvers used by us is done. The steps for algorithm selection and run-time prediction using the machine learning approach is mentioned.

In chapter 4 of the thesis, we describe our experimental environment and evaluation of solvers. The evaluation of solvers based on order and empirical hardness has been done. Finally, the data sets used for algorithm selection and run-time prediction has been described.

In chapter 5 of the thesis, we describe the evaluation of algorithm selection approach using five different state-of-the-art machine learning techniques. The chapter includes results based on four data sets, pre-processing, feature selection technique, discretization and parameter tuning. A confusion matrix for each data set has been presented. Finally, an overall analysis has been done.

In chapter 6 of the thesis, we describe evaluation of the run-time prediction. The chapter includes results based on four data sets, their Pre-processing and results using six machine learning techniques. Finally, an overall analysis is presented.

Finally, we conclude with our findings in Chapter 7.

CHAPTER 2

Background

2.1 Sudoku

Sudoku puzzles are generally published in newspapers, magazines and websites due to its popularity and for brainstorming [Lew07a]. Interestingly, Sudoku has found its usage in binary data encryption [WNA10] and image encryption [WZN+10]. Sudoku has been formally defined as a constraint programming problem [Sim05].

2.1.1 Definition

Let us represent $\mathcal{S}_n$ as a Sudoku board and $\mathcal{S}_n^p$ as a Sudoku puzzle of order n. A Sudoku puzzle $\mathcal{S}_n^p$ is a partial Sudoku board $\mathcal{S}_n$. The end goal is to convert the partial Sudoku board into a complete Sudoku board. $\mathcal{S}_n$ is defined as a $n^2 \times n^2$ matrix, where [Lew07a]:

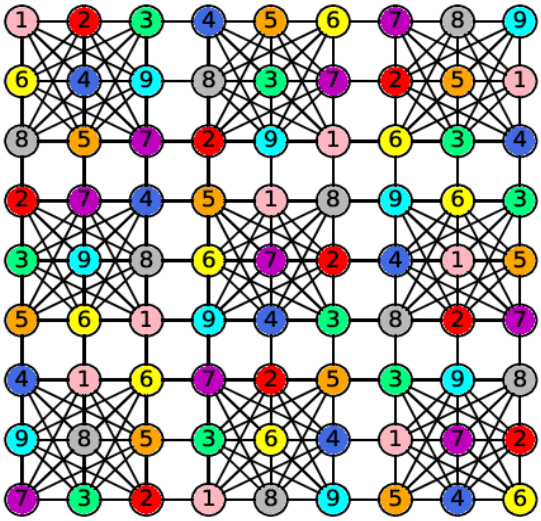
- each row of cells contains each number from $\{1, \ldots, n^2\}$ exactly once,
- each column of cells contains each number from $\{1, \ldots, n^2\}$ exactly once and
- each sub-grid $(n \times n)$ of matrix contains each number from $\{1, \ldots, n^2\}$ exactly once.

The p in $\mathcal{S}_n^p$ represents the probability with which the puzzle has been generated. The puzzle generation method is mentioned in Chapter 3, Section 3.1. A typical Sudoku puzzle has been depicted in the Figure 2.1 (a), where the $\mathcal{S}_n^p$ presented is of order $n = 3$ and $p = \cdot 28$.

A well formed Sudoku puzzle is one that has only one solution and are logic solvable [Lew07a, Sim05]. The goal of logically solving a puzzle is to arrive to a corresponding optimal Sudoku board, that obeys the row, column and sub-grid constraints. It is to be

1						7		9
	4				7	2		
8								
	7			1			6	
3								5
	6			4			2	
								8
		5	3				7	
7		2					4	6

(a) Unsolved Sudoku.



(b) Graph coloring of Sudoku.

Figure 2.1: Sudoku as a Graph Coloring Problem [AC18]

noted that not all Sudoku puzzles are logic solvable and have one solution [Lew07a, YS03]. Additionally, Sudoku has also been proven to be a NP-Complete problem [YS03].

Interestingly, Sudoku can be converted in to a graph coloring problem [HM07]. A graph coloring example for the Sudoku has been depicted in Figure 2.1 (b). Not only can Sudoku be converted into a graph coloring problem but also into a Hamiltonian cycle problem [Hay16], exact cover problem [Kap10], constraint programming problem [Sim05], propositional satisfiability problem [LO06] and integer programming problem [BCLR08].

2.1.2 Solving Sudoku with Pen and Paper

Generally, humans solve logic solvable Sudoku puzzles using constraints. There are various techniques to solve the problem such as the forced cell technique, twins technique, x-wings, etc [RT12].

The forced cell technique has 2 different types; either a cell in the puzzle can always be identified for a candidate value or a row, column or sub-grid can have only one cell available for a given digit [RT12]. The twins technique is similar to the forced cell technique but concerning pair of numbers for 2 and more cells and finally one cell is considered filled by logic [RT12]. These are techniques very similar to constraint programming in computer science.

The other way of solving Sudoku could be, adding the numbers in rows, columns and sub-grids [RT12]. They arrive to the same count for every Sudoku board. Similarly, multiplication could be used [RT12]. In case of addition and multiplication, there could

be a loss of information if the digits are not following the Sudoku constraints in rows, columns or sub-grids.

2.2 Algorithms for Sudoku Puzzle

In this section of the thesis, the state-of-the-art algorithms and modelling techniques used for Sudoku puzzles will be discussed in brief. The Sudoku puzzle problem is known to be a NP-Complete problem [YS03]. Computationally, Sudoku has been popularly known to be modelled with constraint programming [Sim05], as a propositional satisfiability problem (SAT) [LO06] and integer programming [BCLR08].

NP-complete problems are the hardest problems of class NP [Woe03]. Though these problems can be solved by exhaustive search, there are larger instances of problems that can increase the running time of the exhaustive search algorithms and for some instances heuristic methods may be more effective than the exact methods [Woe03].

Sudoku as Constraint Programming

As known by now, Sudoku is a constraint programming problem [Apt03, Sim05] and can be described using the *alldifferent* constraints [vH01]. The *alldifferent* constraint in the case of Sudoku is applicable to each row, each column and each sub-grid, where the domain is $\{1, \ldots, n^2\}$. Throughout the thesis, some solvers used by us use the basic *alldifferent* constraint modelling approach for Sudoku. The approaches and propagation methods to solving will be discussed in the later sub-section.

Sudoku as a SAT

Sudoku has also been modelled as a propositional satisfiability (SAT) problem, where the clauses have been encoded in conjunctive normal form [LO06]. These encoding can be solved using propositional satisfiability inferencing techniques, such as picosat, unit propagation or dpll algorithm[Bie08, DP60]. The paper [LO06] describes two SAT encoding; minimal and extended. The minimal encoding is a sufficient encoding whereas the extended encoding adds redundant clauses [LO06]. The extended encoding described by Lynce and Ouaknine [LO06] is similar to the *alldifferent* approach of constraint programming [Sim05]. The number of assignments for a variable in Sudoku for *alldifferent* constraint is same as the unit propagation [DP60, LO06].

Sudoku as an Integer Program

Sudoku has also been formalized as a binary integer linear program (BILP) [BCLR08]. The decision variable considered for BILP is $x_{ijk} = \{0, 1\}$, where $x_{ijk} = 1$ if element (i, j) of the $n^2 \times n^2$ Sudoku matrix contains the integer $k = \{1, \ldots, n^2\}$ and $x_{ijk} = 0$ otherwise. Moreover, as known to us by Helmut Simonis [Sim05] that Sudoku is a constraint programming problem. BILP's formulation of the constraints are; there is only one k value for each row, there is only one k value for each column, there is only one k value for each sub-grid, every position in the Sudoku matrix must be filled and finally, $x_{ijk} = 1$ [BCLR08]. Since this is a satisfiability problem no objective function

is generally required but for the use in programming such as MATLAB [MAT10] an objective function is defined as $0^T x$, where 0^T is a vector of objective function coefficients [BCLR08]. The objective function is to be minimised.

2.2.1 General Exact Solving Strategies

Helmut Simonis has mentioned several approaches to solve the Sudoku using constraint programming [Sim05], all of them using the *alldifferent* constraint for the Sudoku puzzle. The propagation schemes mentioned are forward checking, forward checking with channeling, *alldifferent* with bound-consistency, bound-consistency with channeling, *alldifferent* with hyper arc-consistency (HAC), HAC with colored matrix, HAC with rows or sub-grids interaction, HAC with colored matrix and same constraints, HAC with same constraints and rows or sub-grids interaction, HAC with colored matrix and rows or sub-grids interaction, *alldifferent* with forward checking plus shaving, *alldifferent* with bound-consistency plus shaving and *alldifferent* with hyper arc-consistency plus shaving [Sim05]. The *alldifferent* survey in [vH01] describes the different consistency methods [Sim05].

Shaving [TL00] is a technique that uses the complete constraint set by trying to set variables to values in the domain of the Sudoku puzzle, i.e. $\{1, \ldots, n^2\}$ [Sim05]. Many inconsistent values are removed before the search begins by shaving, if the assignment has failed. For the Sudoku puzzle all the values of the domain needs to be tested for shaving [Sim05].

NP-Complete problems such as Sudoku can be solved by pruning the search tree [Woe03]. The steps include the identification of the feasible solution for each search space, determining the domain of the space and the enumeration of the branch into several sub-values, like a tree structure. Generally the sub-trees that do not give optimal behaviour are removed from the search space. Every branch and bound algorithm works on this method [Woe03].

Every NP-Complete problem can be solved using a Dynamic programming approach over the subsets [Woe03]. For each set of a chosen subset there are polynomial number of corresponding subsets or the state space. Generally, dynamic programming uses a complexity of $\mathcal{O}(2^n)$ [Woe03]. An extended approach is to pre-process the data, it usually means the reduction of the data before the algorithmic run [Woe03].

Another extensively used method for NP-Complete problems such as Sudoku is the combination of exact (exact exponential time algorithm) approach using local search which looks for a solution in the set of feasible solutions [Woe03]. It moves from one feasible solution to the other in its neighbour. Mathematically, the neighbour is to be found using a certain criteria, for example; in propositional satisfiability (SAT) it uses the truth values $\{0, 1\}$ [Woe03].

Above we have described some of the common methods to solve a NP-Complete constraint problem. One of the most important exact methods in trend is that of lazy clause generation and has proved to out perform other methods [DYS20]. Our thesis investigates Google OR-Tools CP-SAT [1] and Chuffed [CdlBS10] based on the approach of lazy clause generation for Sudoku puzzles. These lazy clause generation solvers based on a hybrid approach of constraint programming and SAT has proved to be more efficient than any state-of-the-art constraint satisfaction problem (CSP) solvers [DYS20]. Lazy Clause Generation is a technique in which finite domain propagation is used for solving constraint programming and is combined with SAT's conflict learning ability [DYS20].

2.2.2 Population-Based Approaches

Genetic Algorithms

Genetic Algorithms (GA) are inspired by Darwin's theory of evolution [Hol92]. There have been evolutionary approaches to the Sudoku puzzle in the computer science community. Moraglio *et al.*[MTL06] have used product geometric crossover coupled with GA to solve Sudoku puzzles. The other method that is more efficient is the approach using than Moraglio *et al.*'s approach is using more straight forward GA approach [MK07]. The method used swap mutations along with a fitness, the fitness function was designed to be penalized on every constraint violation of the Sudoku [MK07] and in the optimal solution all constraints are satisfied and the fitness function becomes zero. Moreover, these algorithms have been tested only for order, $n = 3$ puzzles.

Ant Colony Optimisation

Ant colony optimisation is a population-based search method which has been inspired by the ants behaviour [DDC99]. The approach has been used to solve several computational problems such as the TSP [DG97]. The [LA20] uses a variant of constraint propagation and uses ant colony optimisation rather than using a search based approach. In [LA20] two basic rules have been applied; first is the fix the value that has been prefixed and second is to fix the value that can be fixed by constraints, i.e., the domain set. Due to these other cells could also be fixed recursively. The method used in [LA20] is a variant of the method used in [DG97]. After each iteration, each ant starts with a new puzzle and the aim of the ants to fix as many cells as possible [LA20]. The approach has been generalized and the paper [LA20] have solved puzzles up to the order of $n = 5$.

Artificial Bee Colony

The artificial bee colony population-based method for Sudoku puzzles that we will discuss here is an improvised version of the approach mentioned in [QS08]. In the artificial bee colony method [PSY09] four parameter values have been defined which are, maximum

[1] `https://developers.google.com/optimization/reference/python/sat/python/cp_model`

number of cycles, number of employed bees, number of onlooker bees and number of scout bees. the first three parameters are user defined while the fourth is 0.1 times the number of employed bees. During initialization each bee is given a copy of the Sudoku puzzle and they randomly start filling the empty cells and then the population is evaluated, the cycle continues until a stopping criteria is met [PSY09].

Particle Swarm Optimization

The approach we will discuss here is based on integer particle swarm optimization. It is based on the approach of discrete binary particle swarm optimisation described in [KE97]. Kennedy *et al.*approach was binary where as in this method the approach is multi-valued [HG08].The velocity assumed was different for the dimensional space. The particles had a multi-dimensional velocity vector and each was updated separately [HG08]. The velocity vector had been scaled to a value of 0 to 1. The probability with which the number will be randomly selected from the domain set of the Sudoku and is replaced by the current value. The string of numbers is finally put back and the fitness is calculated [HG08].

2.2.3 Local Search Based Approaches

Simulated Annealing

The meta-heuristic based approach was introduced by Lewis [Lew07a] for solving Sudoku puzzles. The approach uses the Sudoku constraints of sub-grids to solve the puzzle using a random filling of the puzzle with the domain elements $\{1, \ldots, n^2\}$. In order to maintain the sub-grid constraint a swap of cell elements is done for the cells that do not satisfy this constraint. This produces an initial candidate solution that ensures the sub-grid constraint of the puzzle is met. The candidate solution is evaluated using the row and then the column constraint. The cost function is to be evaluated based on the number of constraint violations and they are then fixed, first for the rows and secondly for the column. The cost function for an optimal solution is evaluated to zero. The other paper of Lewis [Lew07b] also discusses the method of simulated annealing.

Hybrid Tabu Search

Here we mention two hybrid Tabu search methods, one uses the *alldifferent* constraint [SCG+15] and the other uses arc-cosistency 3 (AC3) [SCG+13] along with tabu search. The first approach proposes to filter the the variable domain in a pre-processing phase and at every iteration of the Tabu search [SCG+15]. The *alldifferent* constraint is applicable to the rows, columns and sub-grids iteratively. For a full solution a best solution is found or a maximum iteration limit has been reached.

The second method using AC3 and Tabu search, it merges classic tabu search based method with AC3 filtering. The values that do not lead to the solution are removed. The reduction in the number of iterations is done [SCG+13].

Hybrid Iterated Local Search

The hybrid approach uses forward checking for the CP approach along with dynamic variable ordering coupled with iterated local search [MW17].

2.3 Algorithm Selection and Run-time Prediction

In this section, we will discuss the Algorithm Selection (AS) and Run-time Prediction (RP) approach to a given problem. We will introduce the background knowledge of the approaches and also include some important state-of-the-art research. The Rice Framework [Ric76] for AS and the Leyton *et al.*[HXHLB14] approach to RP will be briefly discussed.

2.3.1 Algorithm Selection

Although there are many optimisation and heuristic approaches for NP-complete combinatorial problems, algorithms out perform each other over some instances [WM97]. This theorem is named the *No Free Lunch Theorem* [WM97]. The idea of AS is to reduce the hassle of writing new algorithms for every combinatorial problem instance separately and select the best performing algorithm from the existing set of algorithms. The problem of AS was first introduced in 1976 by Rice [Ric76]. The framework consisted of two models a basic model and a refined model. The model has been presented to have a high algorithmic performance. The basic model of Rice Framework does not introduce the feature space whereas the refined model considers the feature space into the framework. The introduction of the feature space was done to make it simpler and to reduce the dimension from that of the problem space [Ric76].

In the adapted model of Rice [Ric76], set of problem instances is represented by $\mathcal{P}$, set of features by $\mathcal{F}$ (which identifies each problem instance separately), the set of algorithms by $\mathcal{A}$ and $\mathcal{Y}$ represents the performance space. The adapted schematic diagram of the AS is based on [SM09] and is presented in Figure 2.2. The definition of the AS problem is as follows [SM09]:

"For a given problem instance $x \in \mathcal{P}$, with features $f(x) \in \mathcal{F}$, find the selection mapping $S(f(x))$ into algorithm space $\mathcal{A}$, such that the selected algorithm $\alpha \in \mathcal{A}$ maximises the performance mapping $y(\alpha(x)) \in \mathcal{Y}$."

2.3.2 Problem Space and Algorithm Space

For NP-Complete problems, there are generally benchmark problems for instances of the $\mathcal{P}$. These instances tend to cover the varying empirical hardness of $\mathcal{P}$, which generally vary for each algorithm [WM97]. This makes sense for AS to have a varying instance empirical hardness of the $\mathcal{P}$.

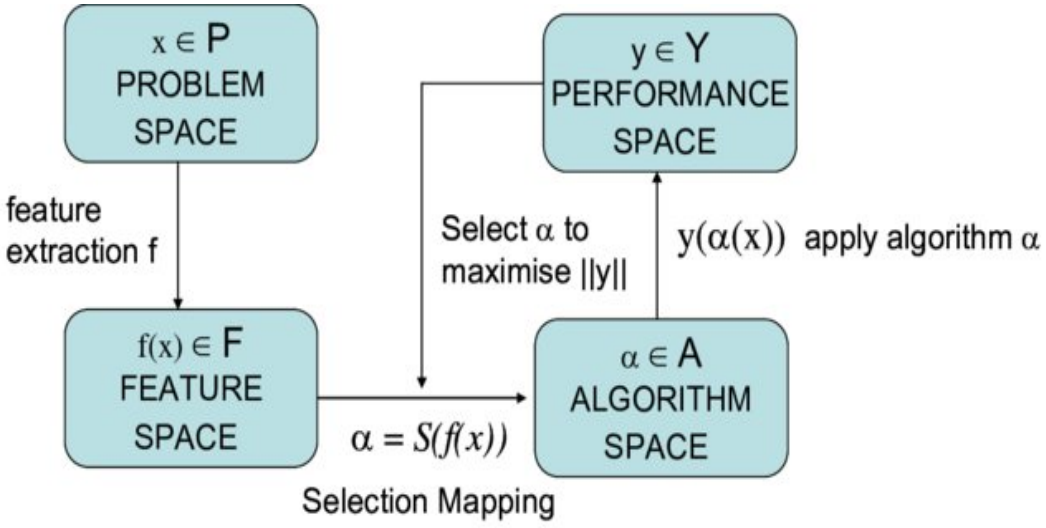


Figure 2.2: The Rice framework as adapted from Kate Smith-Miles [Ric76, SM09]

The other important aspect of the AS problem is the algorithm space $\mathcal{A}$, it is of importance to have good performing set of algorithms. The *no free lunch* theorem is a good motivation to the problem statement of AS [Kot16]. Moreover, in AS the algorithm portfolio can be of two distinct types; static and dynamic [Kot16]. In static portfolio, the designer decides which algorithms to include for the AS approach based on instances and in dynamic portfolio, the composition of algorithms vary for different problem instances [Kot16].

2.3.3 Feature Space

Features are one of the most important aspects of the algorithm selection approach [Kot16, MS13, XHHLB08]. Every $f(x) \in \mathcal{F}$ is to be computed within the average time of the run-time of algorithm over $\mathcal{P}$ and the computation should not cause overhead [MS13, Kot16]. In addition to the computation time of the features, the feature space should be able to detect the phase transitions in instances [Kot16, Ric76].

There are different types of features that have been computed for the approaches of AS. They are the domain knowledge features, which have been implemented by other stat-of-the-art approaches [MS13], SAT based features [XHHLB08] and other *meta*-learning features have been discussed in [XHHLB08].

Feature Selection

Selection of informative features are important in AS [Kot16]. There needs to be a good relation between the features and the performance of algorithms used in the AS approach [Kot16, MS13]. There are various feature selection techniques, manually selecting features [HDH+00], feature selection based on forward selection, backward selection, genetic search, statistical feature selection, etc [GE03, FPHK94, MS13].

2.3.4 Performance Measure

Performance measure of AS is the mapping of an algorithm based on its performance metric and problem, in this case the run-time of the algorithm over an instance [Kot16].

2.3.5 Run-time Prediction

The Run-time Prediction (RP) of an algorithm has continuously been investigated. Algorithmic empirical hardness was first determined by Leyton [LBNS02] for the Winner Determination Problem. The running time of IBM-Cplex was examined [Cpl09]. RP for problems such as SAT, (Mixed Integer Programming) MIP and Travelling Salesperson (TSP) along with several methods and their evaluation has been discussed by Leyton *et al.*[HXHLB14]. The RP for NP-Complete problems [LBHHX14] has also been investigated.

The methodology proposed by Leyton *et al.*[LBNS02, LBHHX14, HXHLB14] has a similar approach and is closely related to AS approach of problem space $\mathcal{P}$, algorithm space $\mathcal{A}$ and feature space $\mathcal{F}$. The data sets are compiled with features along with the algorithmic run-time.

CHAPTER 3

Algorithm Selection and Run-time Prediction for Sudoku Puzzles

In this chapter, the algorithm selection and run-time prediction approach for Sudoku puzzles is discussed. Section 3.1, discusses the Sudoku puzzle instance generation method. Section 3.2, the Sudoku solvers considered for algorithm selection and run-time prediction is mentioned. Section 3.3, a discussion on features identified and computed is done. Finally in Section 3.4, the steps involved for algorithm selection and run-time prediction of Sudoku puzzles is presented.

3.1 Sudoku Instances

For this thesis Sudoku puzzles have been generated by the method described in [Lew07a]. The puzzles generated range from a order of 3 to 9, i.e., $n \in \{3, \ldots, 9\}$. As already mentioned, puzzles that are presented in magazines and websites are generally solvable by logic (having one clue for one position), having only one solution [Lew07a, YS03].

3.1.1 Instance Generation Method

The instance generation method is based on a randomised probabilistic approach [Lew07a]. A Sudoku that is completely filled and satisfies all constraints is an optimal Sudoku board. For a given optimal board permutation of cells of rows and columns is performed to obtain another optimal board.

The mentioned method can generate $n!^{2(n+1)} - 1$ valid Sudoku boards from a single optimal board [Lew07a]. The optimal boards are obtained by taking the row, column

and sub-grid constraints into consideration. This generation method does not produce all the possible boards [Lew07a].

Considering the above mentioned points puzzles are generated by the removal of numbers from boards with a probability of $1 - p$, where $0 \leq p \leq 1$. The removal of numbers are done using a randomised approach. The removal of numbers generate Sudoku puzzles corresponding to optimal Sudoku boards. For higher values of p the puzzles generated will be highly constrained and for low p values generated puzzles will be fairly unconstrained [Lew07a].

In our thesis, we have considered generating puzzle instances from the orders of 3 to 9. The p values considered for generating puzzles lie in between $0 \cdot 05$ and $0 \cdot 95$ with an interval of $0 \cdot 05$. Therefore a total of 19 p values have been considered for each order. Additionally, for each order and for each p value 20 different puzzles have been generated bringing the count of each different $\mathcal{S}_n^p$ instance to 20. The total number of puzzles generated for each order n is 380. The total number of puzzles generated were 2660 for the considered values of n and p.

As mentioned in [Lew07a], higher p value generates highly constrained puzzles and low p values will generate fairly unconstrained puzzles. Therefore, the higher p values will generate puzzles that are fairly easy to solve and may also be logic solvable in many cases. Whereas lower p values in many cases will produce puzzles that are not logic solvable [Lew07a]. Due to its randomised probabilistic nature there may be puzzles generated that are same.

3.2 Solvers for Sudoku

The $\mathcal{S}_n^p$ instances generated require high performing algorithms for solving, specially for the case of higher n values. For the algorithm space of AS and RP problems, the below mentioned exact and *meta*-heuristic approaches have been considered. Several solvers were investigated. After investigation, these solvers were considered due to their ability to solve $\mathcal{S}_n^p$ instances most efficiently. Other Sudoku solvers and algorithms have been mentioned in Chapter 2 of the thesis. *Six* Sudoku solvers were tested during the experimentation phase of our thesis.

3.2.1 Exact Approaches

By now we know that Sudoku can be modelled as a constraint satisfaction problem [Sim05]. There exist several optimisation solvers for solving constraint satisfaction problems. Narrowing the look for solvers we considered 4 of them, namely OR-Tools CP-SAT (ORTL) by Google [1], IBM-Cplex (CPLEX) [Cpl09], Chuffed (CHUF) [CdlBS10] and Gurobi (GUROBI) [GO21].

[1] `https://developers.google.com/optimization/reference/python/sat/python/cp_model`

The modelling of Sudoku constraints was done using the *alldifferent* constraints for all of the above mentioned solvers [vH01, Sim05]. In a set of variables an *alldifferent* constraint implies that no two variables from that set have the same value, for Sudoku the set of variable $v = \{1, \ldots, n^2\}$ [vH01].

In ORTL, the *alldifferent* constraint along with the CP-SAT solver was used. The CP-SAT involves constraint optimisation along with the SAT methods [2]. For CPLEX, CHUF and GUROBI, the *alldifferent* constraint was used for modelling the Sudoku constraint optimisation problem. The $\mathcal{S}_n^p$ instances for the latter were run using the MiniZinc tool [NSB+07, SFS+14].

Both CHUF and ORTL are based on Lazy Clause Generation. These lazy clause generation solvers based on a hybrid approach of constraint programming (CP) and propositional satisfiablity (SAT) has proved to be more efficient than any state-of-the-art constraint satisfaction problem (CSP) solvers [DYS20]. Lazy Clause Generation is a technique in which finite domain propagation is used for solving CP and is combined with SAT's conflict learning ability [DYS20].

3.2.2 Heuristic Approaches

The Simulated Annealing (SIMA) *meta*-heuristic based approache for Sudoku puzzles were introduced in [Lew07a, Lew07b]. SIMA approach introduces the *alldifferent* approach for the third constraint of sub-grids and couples it with a cost function for rows and columns. The solution is optimal if the cost is 0.

The other heuristic approach is a hybrid approach which includes a CP based approach of forward checking with dynamic variable ordering applied in between phases of local search iterations [MW17]. It will be mentioned as HYILS throughout the thesis.

3.3 Features for Sudoku Puzzle

As mentioned earlier, features are one of the most important aspects for AS and RP using machine learning. In this section we will present the features identified that can distinguish one $\mathcal{S}_n^p$ instance from another. The features are broadly related to the domain knowledge of Sudoku instances, their graph coloring variant and flatzinc features from MiniZinc.

A brief recap of terms; a Sudoku, which is completely filled and follows all constraints is a Sudoku board. A Sudoku puzzle is a partially filled Sudoku board.

Factors taken into consideration when computing the features:

- time for computation of features.

[2] https://developers.google.com/optimization/reference/python/sat/python/cp_model

The features presented below are specific to the $\mathcal{S}_n^p$ instances generated by the method mentioned in the above section. For simplicity, we will introduce each feature belonging to a certain class $\mathcal{C}$ and denote it as $\mathcal{C}_x$ where x is the feature. Features related to Sudoku are meaningful using statistical metrics. The features identified and computed here are inspired by papers [MS13, XHHLB08, HXHLB14, HM07, RT12].

3.3.1 Problem Based Features

Order Features

It is known by now that Sudoku puzzles are referred to as $n^2 * n^2$ puzzles, where $n \geq 2$ is an integer. The order ($\mathcal{O}_n$) and size ($\mathcal{O}_{n^2}$) of the puzzles were included in our set of features. The order and size of the puzzle distinguish the puzzles of varying orders in between $3 \leq n \leq 9$.

Sudoku Specific Features

A Sudoku puzzle is a partially filled Sudoku board as introduced in Chapter 2. Summation (a) and Products (p) of Sudoku puzzle rows, columns or sub-grids can be different for puzzles but are same for a Sudoku board of that order n. The total summation and product of a partial Sudoku may also be different in the case of puzzles but is always same in the case of a board for order n. Aggregate functions such as minimum, maximum, mean, ratio (ra), range (r), inter quartile range (iqr for puzzle) and standard deviation (sd) can also be used as attributes. The features that we use have been presented in Table 3.1.

Counting the number of digits with aggregates mentioned above using rows, columns and sub-grids can categorize puzzles distinctively. Similarly, counting the number of empty and full rows, columns or sub-grids along with their aggregates are some other features introduced. The domain set of a Sudoku board is $\{1, \ldots, n^2\}$, it may be of importance knowing the number of elements from the domain set of Sudoku present and missing from the puzzle. These features belong to the class $\mathcal{NS}_x$ where x is the feature name. The features that we use have been presented in Table 3.2.

3.3.2 Graph Coloring Features

Some of the counting strategies mentioned above are similar when a Sudoku board is converted to its graph coloring variant. The notion of a Sudoku board of order n has a proper coloring of n^2. It is known that a graph is called regular if the degree of each vertex is the same, which is the case for every order n Sudoku board [HM07]. The maximum degree of a vertex in a board is given by $3n^2 - 2n - 1$ [HM07]. The Sudoku board graphs are regular hence every node of a Sudoku board has the same degree.

Since the concern is $\mathcal{S}_n^p$ instances, the equivalent graph from a puzzle is a partially colored Sudoku graph or a partial Sudoku graph [HM07]. Some of the features for the partial graph of a Sudoku puzzle has been inspired by [MS13]. They are mentioned as nodes, node degree, edges and clique. The paper [MS13] focuses on graph coloring problems

1	$\mathcal{NS}_{mp}$	mean of puzzle
2	$\mathcal{NS}_{ma}$	mean of board
3	$\mathcal{NS}_{srp}$	range of sum from puzzle to board
4	$\mathcal{NS}_{alr}$	largest sum row
5	$\mathcal{NS}_{alc}$	largest sum of column
6	$\mathcal{NS}_{alsg}$	largest sum of sub-grid
7	$\mathcal{NS}_{sar}$	smallest sum of row
8	$\mathcal{NS}_{sac}$	smallest sum of column
9	$\mathcal{NS}_{sasg}$	smallest sum of sub-grid
10	$\mathcal{NS}_{plr}$	largest product of row
11	$\mathcal{NS}_{plc}$	largest product of column
12	$\mathcal{NS}_{plsg}$	largest product of sub-grid
13	$\mathcal{NS}_{spr}$	smallest product of row
14	$\mathcal{NS}_{spc}$	smallest product of column
15	$\mathcal{NS}_{spsg}$	smallest product of sub-grid
16	$\mathcal{NS}_{sdsr}$	standard deviation for row sum of puzzle from board row sum
17	$\mathcal{NS}_{sdsc}$	standard deviation for column sum of puzzle from board column sum
18	$\mathcal{NS}_{sdssg}$	standard deviation for sub-grid sum of puzzle from board sub-grid sum
19	$\mathcal{NS}_{rmmsr}$	ratio of minimum to maximum sum of row
20	$\mathcal{NS}_{rmmsc}$	ratio of minimum to maximum sum of column
21	$\mathcal{NS}_{rmmssg}$	ratio of minimum to maximum sum of sub-grid
22	$\mathcal{NS}_{rmtsr}$	ratio of minimum puzzle to total row of board sum
23	$\mathcal{NS}_{rmtsc}$	ratio of minimum puzzle to total column of board sum
24	$\mathcal{NS}_{rmtssg}$	ratio of minimum puzzle to total sub-grid of board sum
25	$\mathcal{NS}_{minon}$	minimum occurrence of a digit
26	$\mathcal{NS}_{maxon}$	maximum occurrence of a digit
27	$\mathcal{NS}_{fomin}$	frequency of occurrence for minimum
28	$\mathcal{NS}_{fomax}$	frequency of occurrence for maximum
29	$\mathcal{NS}_{ds}$	domain size
30	$\mathcal{NS}_{mds}$	missing domain size
31	$\mathcal{NS}_{sc}$	is the domain complete? yes/no
32	$\mathcal{NS}_{sp}$	sum of the puzzle

Table 3.1: Table of features related to sum, product and domain of Sudoku puzzle.

which has the requirement of finding the optimal number of colors for a graph. The Sudoku board has a known chromatic number of n^2. Note, while the features computed may be of the same name as that in [MS13], many are different and has a different meaning due to its partial graph nature. Moreover, the computation time for graph coloring features for Sudoku puzzles are very low as the puzzle does not need to be converted into a graph and the features can be obtained from the puzzle itself.

Node Features

1	$\mathcal{NS}_{pp}$	percent of the puzzle filled
2	$\mathcal{NS}_{iqr}$	inter quartile range of puzzle
3	$\mathcal{NS}_{ssr}$	size of smallest row
4	$\mathcal{NS}_{ssc}$	size of smallest column
5	$\mathcal{NS}_{sssg}$	size of smallest sub-grid
6	$\mathcal{NS}_{slr}$	size of largest row
7	$\mathcal{NS}_{slc}$	size of largest column
8	$\mathcal{NS}_{slsg}$	size of largest sub-grid
9	$\mathcal{NS}_{rmmr}$	range from minimum to maximum size row
10	$\mathcal{NS}_{rmmc}$	range from minimum to maximum size column
11	$\mathcal{NS}_{rmmsg}$	range from minimum to maximum size sub-grid
12	$\mathcal{NS}_{rmtr}$	range from minimum to total size row
13	$\mathcal{NS}_{rmtc}$	range from minimum to total size column
14	$\mathcal{NS}_{rmtsg}$	range from minimum to total size sub-grid
15	$\mathcal{NS}_{rammr}$	ratio from minimum to maximum size row
16	$\mathcal{NS}_{rammc}$	ratio from minimum to maximum size column
17	$\mathcal{NS}_{rammsg}$	ratio from minimum to maximum size sub-grid
18	$\mathcal{NS}_{ramtr}$	ratio from minimum to total size row
19	$\mathcal{NS}_{ramtc}$	ratio from minimum to total size column
20	$\mathcal{NS}_{ramtsg}$	ratio from minimum to total size sub-grid
21	$\mathcal{NS}_{sdmtr}$	standard deviation from total board row to puzzle row size
22	$\mathcal{NS}_{sdmtc}$	standard deviation from total board column to puzzle column size
23	$\mathcal{NS}_{sdmtsg}$	standard deviation from total board sub-grid to puzzle sub-grid size
24	$\mathcal{NS}_{nrfc}$	number of filled completely rows
25	$\mathcal{NS}_{ncfc}$	number of completely filled columns
26	$\mathcal{NS}_{nsgfc}$	number of completely filled sub-grids
27	$\mathcal{NS}_{nrce}$	number of completely empty rows
28	$\mathcal{NS}_{ncce}$	number of completely empty columns
29	$\mathcal{NS}_{nsgce}$	number of completely empty sub-grids
30	$\mathcal{NS}_{de}$	number of digits in the puzzle diagonal
31	$\mathcal{NS}_{de}$	puzzle diagonal filled completely, yes/no

Table 3.2: Table of features related to row, column and sub-grid size of Sudoku puzzle.

The feature number of nodes are equivalent to the number of digits on a puzzle. The missing nodes from the partial graph is the number of missing digits from the puzzle. These features belong to the class of graph size $\mathcal{GS}_x$. The features are number of nodes for the puzzle as $\mathcal{GS}_n$, number of missing nodes for the puzzle $\mathcal{GS}_{mn}$ and total nodes for the board $\mathcal{GS}_{tn}$.

Edge Features

As mentioned earlier, the maximal degree of a Sudoku board is $3n^2 - 2n - 1$. The case of a puzzle is a partial graph and hence the maximum degree can be less than $3n^2 - 2n - 1$.

Similarly, number of edges of the partial graph can be computed. The class graph size $\mathcal{GS}_x$ has also been used for the edge features. The edge based features are edges of the puzzle $\mathcal{GS}_e$, maximum number of edges w.r.t. the degree and missing nodes for the puzzle $\mathcal{GS}_{mne}$, number of maximum missing edges w.r.t. the degree and missing nodes for the puzzle $\mathcal{GS}_{nmme}$, total edges of the board $\mathcal{GS}_{te}$ and maximal degree for the Sudoku $\mathcal{GS}_{mdb}$.

Chromatic Number Features
The chromatic number of a Sudoku board graph is always n^2 whereas the chromatic number of a partial Sudoku graph is a number from the domain $\{1, \ldots, n^2\}$. Another important feature that was obtained from here is the minimum condition for a Sudoku puzzle to have one solution. These features belong to the class χ_x. The features are as follows; chromatic number χ_{cn}, missing chromatic number χ_{mcn} and total chromatic number χ_{tcn}.

A Sudoku puzzle having a chromatic number of at least $n^2 - 1$ is a minimal requirement for a Sudoku puzzle to have a single solution. If the chromatic number of a Sudoku puzzle is less than $n^2 - 1$ then we can always swap two of them to obtain another solution [RT12, HM07]. This feature is named as χ_{mc}.

Maximal Clique Features
The maximal clique of a Sudoku puzzle is the maximum of maximal digits in a row, column or sub-grid. The maximal clique can have nodes less than n^2 for a puzzle. For a Sudoku board the maximal clique is always n^2. The computation of maximal clique $\mathcal{MC}_x$ for the puzzle was done for our thesis. The features introduced are as follows; maximal clique for the puzzle $\mathcal{MC}_{mc}$, total clique for the Sudoku board $\mathcal{MC}_{tc}$ and missing clique from the Sudoku board $\mathcal{MC}_{rc}$.

3.3.3 SAT Features

The computation of SAT based features was not computed because of time taken to generate clauses, nevertheless we make a mention of it. The average computation time can be seen in Figure 3.1. For propositional satisfiability (SAT) based features we referred to the conjunctive normal form encoding mentioned in [LO06]. While SATZilla [XHHLB08] mentions many SAT features, it is time consuming to generate the clauses in conjunctive normal form (CNF) for puzzles $\mathcal{S}_8^p$ and $\mathcal{S}_9^p$. The SAT clauses of $\mathcal{S}_6^p$ and $\mathcal{S}_7^p$ puzzles take 4 and 14 seconds respectively whereas, the average time to compute the entire set of features mentioned until now was $3 \cdot 6$ seconds. This time taken to generate clauses can in fact solve many of the puzzles by solvers such as CHUF, ORTL, SIMA and HYILS.

3.3.4 MiniZinc Constraint Programming Features

MiniZinc constraint modelling language [NSB+07] was used for solving Sudoku instances. For solvers CHUF [CdlBS10], GUROBI [GO21] and CPLEX [Cpl09] flatzinc features ($\mathcal{FZ}_x$) were obtained using MiniZinc parameters. The features that were collected are namely, $\mathcal{FZ}_{chi}$, $\mathcal{FZ}_{chc}$, $\mathcal{FZ}_{cht}$, $\mathcal{FZ}_{gi}$, $\mathcal{FZ}_{gc}$, $\mathcal{FZ}_{gt}$, $\mathcal{FZ}_{cpi}$, $\mathcal{FZ}_{cpc}$ and $\mathcal{FZ}_{cpt}$, they are *flatintvar*, *flatintconstraint*, *flattening time* for CHUF, GUROBI and CPLEX, respectively.

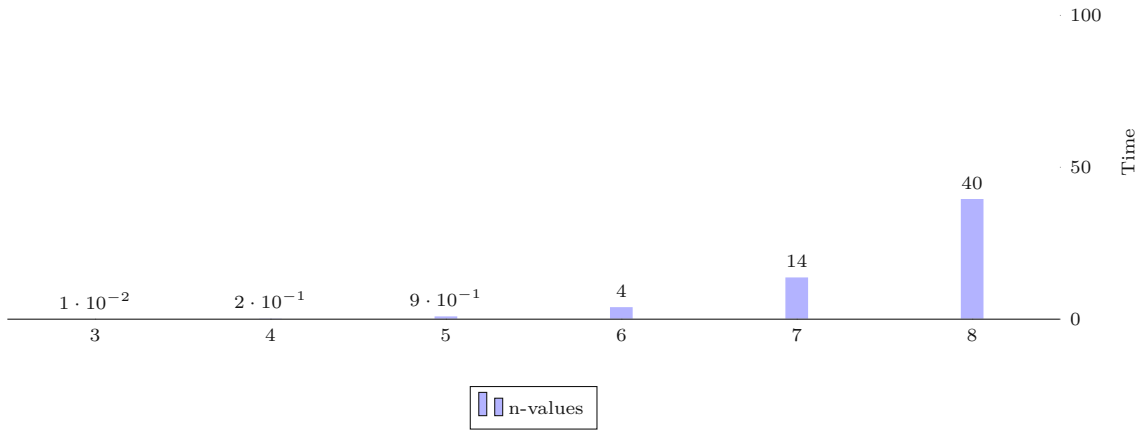


Figure 3.1: Average clause generation computing time for SAT w.r.t. the order of puzzle.

The features for GUROBI and CPLEX were the same and both were not used for any of the further experimentation, due to their excessive computation time. The features from CHUF were used as the computation time was low but only *flatintvar* and *flatintconstraint*.

3.3.5 Feature Space for Algorithm Selection and Run-time Prediction

A total of 89 features were presented in this section. The total features computed for algorithm selection and run-time prediction problems are 70 ($\mathcal{F}_{70}$). Most features not included in the data sets are SAT, CPLEX, GUROBI and multiplication based features. These features were not included due to their excessive computation time or computation overhead in the case of multiplication.

The maximum number of features selected by the feature selection method in the evaluation process of algorithm selection 5 is 25 and for run-time prediction 6 is 70.

3.4 Method for Algorithm Selection

In the above sections we have discussed the problem space (Sudoku instances) $\mathcal{P}$, the algorithm space $\mathcal{A}$ and the feature space $\mathcal{F}$. From Rice's framework 2.2, it is known that the best performing algorithm is to be selected for each Sudoku instance. The selection mapping over the set of features is required and a performance mapping over the puzzle instance.

The features for each puzzle instance $f(\mathcal{S}_n^p) \in \mathcal{F}$ belong to the class of features $\mathcal{C}_x$ and has been discussed in detail in the previous section of Features for the Sudoku puzzle. The best performing algorithm is to be selected $\alpha \in \mathcal{A}$ for each $\mathcal{S}_n^p$ instance. A data set for the Machine Learning (ML) approach is created with the set of features $f(\mathcal{S}_n^p) \in \mathcal{F}_{70}$ along with the best performing algorithm. Moreover, these data sets are used for the ML classification approach.

The ML models are then to be trained such that for the next $\mathcal{S}_n^p$ instance, the model can identify the most efficient algorithm from $\mathcal{A}$. The goal would be to obtain a high accuracy in the case of classification of algorithms.

3.5 Method for Run-time Prediction

Similar to the above approach, algorithm space $\mathcal{A}$, set of features, i.e. the feature space $\mathcal{F}$ and the problem space $\mathcal{P}$ is also be considered for Run-time Prediction (RP). The run-time required to solve a certain problem can help us choose which algorithm to run for a Sudoku instance.

The methodology proposed by Leyton *et al.*[LBNS02, LBHHX14, HXHLB14] has a similar approach and is closely related to the AS approach. The training data sets are created with features $\mathcal{F}_{70}$ and the run-time for an algorithm. The RP is done using ML regression techniques.

CHAPTER 4

Experimental Setup, Solver Evaluation and Data Sets

In this chapter, the evaluation of our system is presented. In Section 4.1, a detail evaluation of Sudoku solvers is presented. Section 4.2, describes the data sets for the machine learning approach of classification and regression for algorithm selection and run-time prediction, respectively.

4.1 Solver Run-time Evaluation

In this section, we present the experimental results of *six* state-of-the-art solvers and their evaluation. Our experiments were performed on Ubuntu 18.04 using Intel-Core I5-3340M 2.70GHz × 4 processor having a 16GB RAM. The same system has been used for the entire computation processes involved in our thesis.

As discussed earlier, the Sudoku instances used are of orders ranging between 3 and 9. They were generated using the methodology used in [Lew07b, Lew07a, MW17]. The method is described in the previous Chapter 3, Section 3.1. Each instance set of order n consists of 380 instances. Therefore, the total number of $\mathcal{S}_n^p$ instances generated were 2660.

4.1.1 Solver Evaluation based on Order

The evaluation of Sudoku solvers are based on their run-time performance. A solver evaluation on the hardness of puzzles has also been discussed. The time limit for each solver on an instance was set to an hour. Note, that all percentages in graphical figures have been rounded to its nearest integer. It can be noted that their are works based on the evaluation of Sudoku solvers only until the order of 5. We present an evaluation until the order of 9.

As discussed in chapter 3 the Sudoku solvers considered are based on exact methods using *alldifferent* constraints and *meta*-heuristics methods. The exact solvers considered are ORTL [1], CPLEX [Cpl09], GUROBI [GO21] and CHUF [CdlBS10]. The *meta*-heuristics methods used are SIMA [Lew07a] and HYILS [MW17]. MiniZinc [NSB+07, SFS+14] tool was used for solvers CPLEX, GUROBI and CHUF. For MiniZinc used solvers, total computation time of a $\mathcal{S}_n^p$ instance is the time taken to generate the MiniZinc (*flatzinc*) constraints and variables along with the solving time.

The Sudoku solvers were run with a time limit of an hour. Each experiment performed over an instance by MiniZinc tool with solvers CPLEX, GUROBI and CHUF was run one time. Solvers SIMA and HYILS were run twice for each instance and an average time was taken. Solver ORTL was run twice and an average taken.

According to the experiments conducted, the Sudoku solvers were very efficient and solved 2096 problems from the problem space of orders $n \in \{3, \ldots, 9\}$. The percent of problems solved is depicted in Figure 4.1 w.r.t. to each solver and order. The higher the order of the puzzle the fewer the number of instances solved for each Sudoku solver.

As mentioned in chapter 3, the $\mathcal{S}_n^p$ instances generated highly constrained puzzles with higher values of p and lower values of p generated fairly unconstrained puzzles [Lew07a]. The puzzle instances $\mathcal{S}_8^p$ with lower p values were hard to solve by all of the solvers, CHUF solved a few of the instances. Similarly, $\mathcal{S}_9^p$ instances with lower p values were not solvable within the time limit of an hour. All instances of order 8 and 9 were solvable most efficiently either by CHUF or ORTL.

However, ORTL and CHUF also could not solve lower p value puzzles and mostly solved instances which were highly constrained. The $\mathcal{S}_6^p$ and $\mathcal{S}_7^p$ instances did have a good number of solved instances for all our solvers. HYILS and SIMA also outperformed the exact approaches of CHUF and ORTL in a few instances. The instances of order $\{3, \ldots, 5\}$ were all solved. The total percent of problems solved for each order $n \in \{3, \ldots, 9\}$ is depicted in Figure 4.2.

The above evaluation describes all puzzles solved according to the benchmarks that were generated using the method described in the previous chapter 3.1. Due to the random and probabilistic nature of the generation, there may be similar puzzles in many cases. Therefore all similar puzzles have been removed from further evaluation.

Based on the instances, 1968 puzzles were distinct for *CHUF*, 1725 were distinct for *ORTL*, 1253 were distinct for *SIMA* and 1795 were distinct for *HYILS*. Therefore, the data sets consist of only puzzle instances that are distinct and solved.

Looking at the percent of problems solved of order $n \in \{8, 9\}$ 4.2, further investigation was done on orders $n \in \{3, \ldots, 7\}$. The efficiency of each solver for orders from $n \in \{3, \ldots, 7\}$ is depicted in Figure 4.3. Therefore, the solvers CPLEX and GUROBI has also been removed from any further analysis as the solvers were not as efficient as the other solvers

[1] https://developers.google.com/optimization/reference/python/sat/python/cp_model

when solving Sudoku puzzle instances 4.3. This could be due to the reason of including the MiniZinc flattening time into the solving time. One would also need to investigate

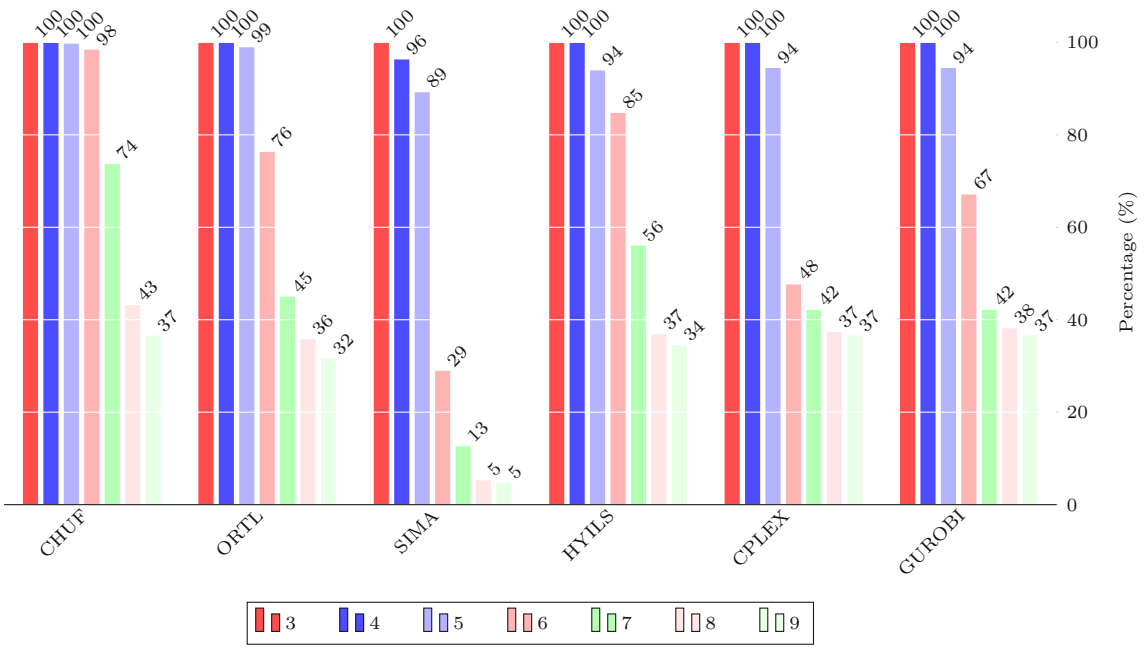


Figure 4.1: Percent of puzzles solved for each order by each solver.

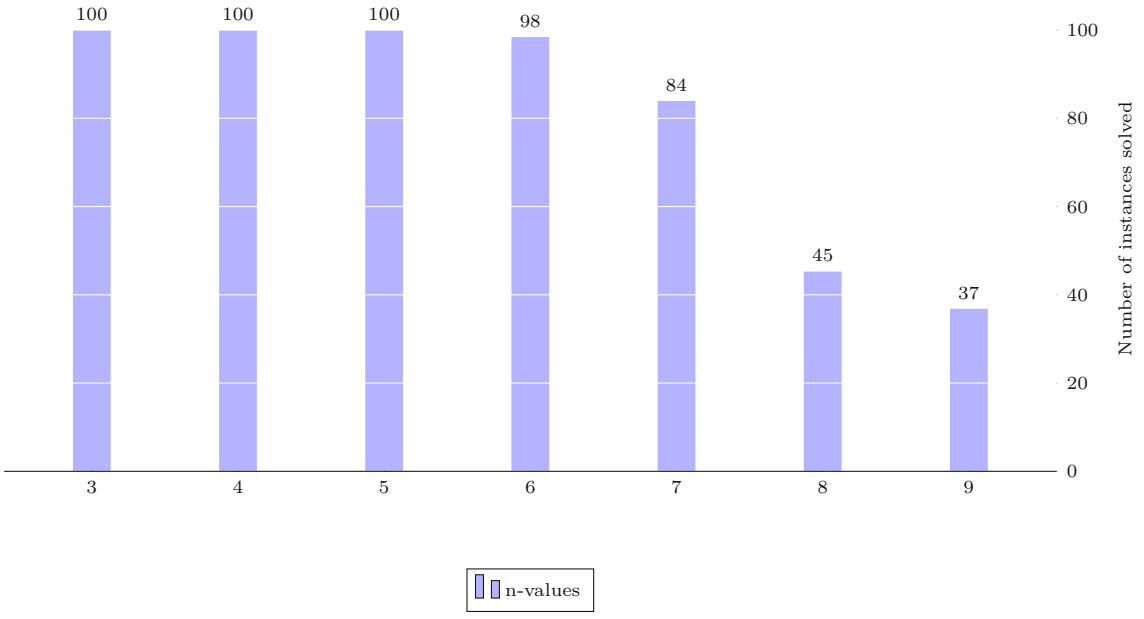


Figure 4.2: Total percent of puzzles solved for each order.

the other aspect of theses solvers, i.e., without the MiniZinc flattening time.

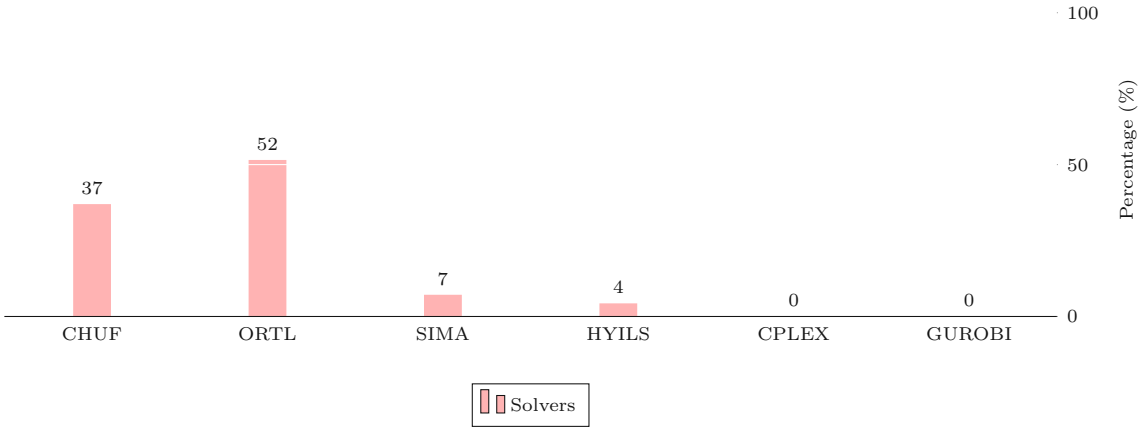


Figure 4.3: Percent of puzzles solved by each solver most efficiently for orders 3 to 7.

Based on the above mentioned results 4.3, a further analysis of solvers and order was conducted. Three sets of algorithms were considered for further analysis. The sets of algorithms are $\mathcal{A}_1 = \{CHUF, ORTL, HYILS\}$, $\mathcal{A}_2 = \{CHUF, ORTL, SIMA\}$ and $\mathcal{A}_3 = \{CHUF, ORTL, HYILS, SIMA\}$. The order sets considered are $n_1 \in \{3, \ldots, 6\}$ and $n_2 \in \{3, \ldots, 7\}$.

The algorithm set $\mathcal{A}_1$ solved 1665 distinct $\mathcal{S}_n^p$ instances for order set n_2. The percent of problems solved most efficiently is depicted in Figure 4.4. Since, the solver SIMA did not solve any $\mathcal{S}_n^p$ instance of order 7 most efficiently, the order set considered for algorithm space $\mathcal{A}_2$ was only $n_1 \in \{3, \ldots, 6\}$. The percent of problems solved by each solver is presented in Figure 4.5.

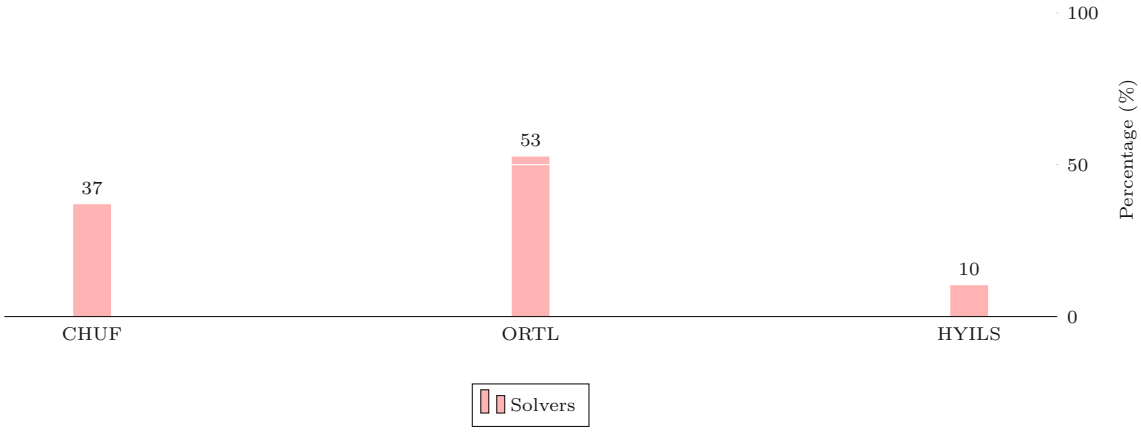


Figure 4.4: Percent of puzzles solved by each solver most efficiently for orders 3 to 7.

Out of the total of 1385 distinct problem instances solved by the algorithm set $\mathcal{A}_2$ of order set n_1, 125 puzzle instances were solved by SIMA most efficiently. The overall percentage for $\mathcal{A}_2$ and n_1 is depicted in Figure 4.5. ORTL solved most of the problems efficiently followed by CHUF and SIMA.

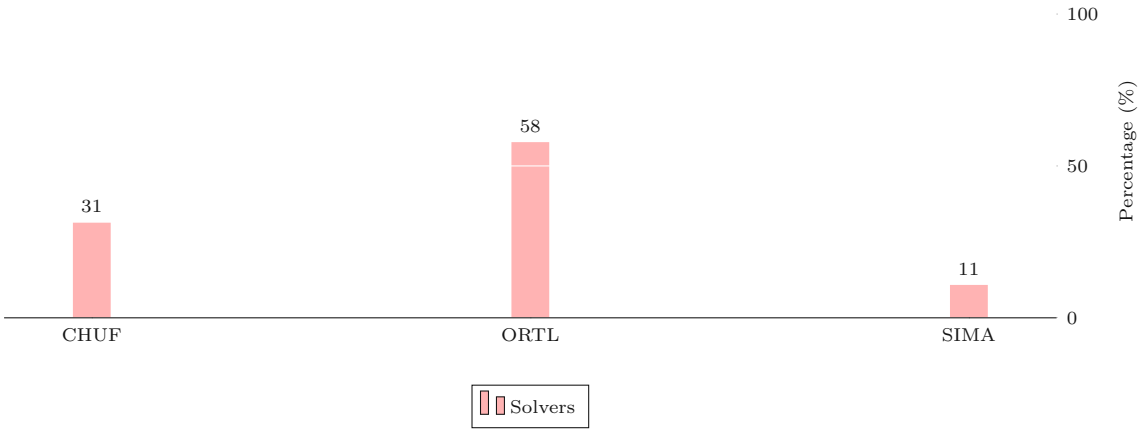


Figure 4.5: Percent of puzzles solved by each solver most efficiently for orders 3 to 6.

Another analysis for the combination of algorithm space of $\mathcal{A}_3 = \{CHUF, ORTL, HYILS, SIMA\}$ and the instance order set n_2 was done. A total of 1665 Sudoku instances solved, i.e. from orders 3 to 7. The most efficient solvers (taking number of puzzles solved) were ORTL, CHUF, SIMA and HYILS, in that order. The percent of Sudoku problem instances solved by each solver efficiently is presented in Figure 4.6.

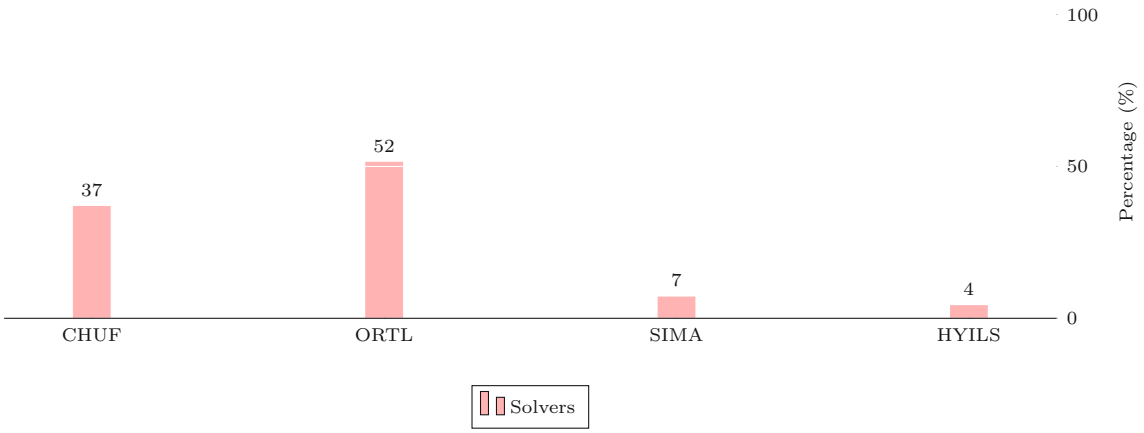


Figure 4.6: Percent of puzzles solved by each solver most efficiently for orders 3 to 7.

The above analysis is the basis for the creation of data sets. They have been described in detail in the next section.

4.1.2 Solver Evaluation Based on Hardness

hardness of Sudoku instances is shown to have a transition of *easy, hard, easy*, based on p values of the instances [Lew07a, MW17]. The above papers [Lew07a, MW17] included orders of $\{3, 4, 5\}$. The instances showed a different transition of hardness *easy, hard, medium*. The unconstrained puzzles were more difficult to solve than the highly constrained puzzles, i.e. the puzzles generated with very low p values were more difficult to solve than the puzzles generated with higher p values.

An evaluation based on hardness of puzzles for order $n \in \{3, \ldots, 7\}$ has been done. The p values have been divided into three parts based on the hardness of the instances. The division of instances based on hardness and p values is depicted in Table 4.1.

	p-values	% Puzzle Filled
Easy	$\cdot 95 \geq p \geq \cdot 65$	65% to 95%
Hard	$\cdot 65 > p > \cdot 40$	40% to 65%
Medium	$\cdot 40 \geq p \geq \cdot 5$	5% to 40%

Table 4.1: Table Depicting hardness of the puzzle instances based on p-values and % puzzle filled with digits.

The percent of Sudoku instances solved based on hardness is depicted in Figure 4.7. In the case of easy puzzles, SIMA and HYILS has solved 25% of the puzzles most efficiently.

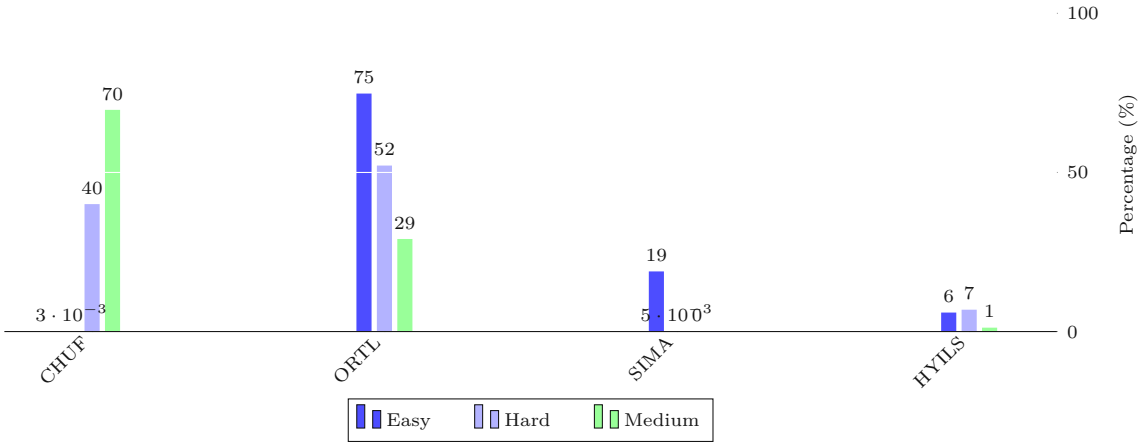


Figure 4.7: Percent of puzzles solved by each solver most efficiently for orders 3 to 7, according to the hardness of puzzles described in Table 4.1.

4.2 Data Sets for Algorithm Selection and Run-time Prediction

Based on the above evaluation of efficient Sudoku solvers, we have created several data sets for AS and RP. The data sets are briefly mentioned in this section. The order of puzzles from 3 to 9 and a mix of algorithms has been considered for our data sets. During our experimentation only puzzles that were highly constrained were mainly solved of order 8 and 9. The puzzles were mainly solved by CHUF and ORTL. Therefore, the order set $n \in \{3, \ldots, 7\}$ has been considered for AS and order set $n \in \{3, \ldots, 9\}$ for RP. The other data set that we have considered for AS is based on hardness *HardSet*, also of order 3 to 7.

4.2.1 Data Sets for Algorithm Selection

The four categories of data division is based on the previous chapter of solver evaluation. The data sets are named *COH37, COS36, COHS37 and HardSet*, where *C* represents CHUF, *O* represents ORTL, *H* represents HYILS and *S* represents SIMA and *36* and *37* represent the problem space of order set n_1 and n_2, respectively. The *HardSet* data set is based on the hardness 4.7 of the puzzles, i.e. the hard puzzles.

The sets of algorithms are $\mathcal{A}_1 = \{CHUF, ORTL, HYILS\}$, $\mathcal{A}_2 = \{CHUF, ORTL, SIMA\}$ and $\mathcal{A}_3 = \{CHUF, ORTL, HYILS, SIMA\}$. The order sets considered are $n_1 \in \{3, \ldots, 6\}$ and $n_2 \in \{3, \ldots, 7\}$.

COH37

The total number of problems solved in the problem space n_2 by the algorithm space $\mathcal{A}_1$ is 1793 out of 1900 Sudoku instances. Out of which, 1665 puzzles were distinct. Therefore, the data set consists of 1665 puzzle instances. The percentage of problems solved by each solver from $\mathcal{A}_1$ is presented in the Figure 4.4.

COS36

The total number of problems solved in the problem space n_2 by the algorithm space $\mathcal{A}_2$ is also 1513 out of 1520 Sudoku instances. Out of which, 1385 puzzles were distinct. Therefore, the data set consists of 1385 puzzle instances. The percentage of problems solved by each solver from $\mathcal{A}_2$ is presented in the Figure 4.5.

COHS37

The total number of problems solved in the problem space n_2 by the algorithm space $\mathcal{A}_3$ is also 1793 out of 1900 Sudoku instances. Out of which, 1665 puzzles were distinct. Therefore, the data set consists of 1665 puzzle instances. The percentage of problems solved by each solver from $\mathcal{A}_3$ is presented in the Figure 4.6. This data set includes the orders $n \in \{3, \ldots, 7\}$ and all four best performing Sudoku solvers.

HardSet

The *HardSet* data set is based on the empirically hard puzzles, the solver evaluation of hardness is depicted in Figure 4.7. The p values considered for hardness is depicted in Table 4.1. A total of 416 were identified solved puzzles from the *HardSet*. Out of which, 393 puzzles were distinct. Therefore the data set consists of 393 puzzle instances. Most were solvable most efficiently by either ORTL or CHUF. HYILS did solve 7% of the hard puzzles and it would be interesting to train the machine learning models based on this data set to see the accuracy results. SIMA solved only 2 puzzles most efficiently of the hard puzzles. Therefore, it has been excluded from the *HardSet*. The $\mathcal{S}_n^p$ instances considered for this data set are $n \in \{3, 4, 5, 6, 7\}$ and $\cdot 65 > p > \cdot 40$.

4.2.2 Data Sets for Run-time Prediction

The order of puzzles investigated for each solver from the algorithm space $\mathcal{A} = \{CHUF, ORTL, SIMA, HYILS\}$ are $n \in \{3, \ldots, 9\}$. For each solver considered, a naming suffix of 39 is considered for the data sets. Four data sets have been constructed for the algorithmic RP namely; *CHUF39*, *ORTL39*, *SIMA39* and *HYILS39*. In the solver evaluation section is shown the percent of problems solved by each Sudoku solver from $\mathcal{A}$ and for each order n 4.1.

CHAPTER 5

Evaluation of Algorithm Selection

In this chapter, the evaluation and analysis of Algorithm Selection (AS) is presented. Section 5.1 introduces the pre-processing and feature selection methods used the ML classification method. In Section 5.2 the results of the machine learning classifiers are presented. Section 5.2.8, presents analysis of the classification based on confusion matrix. Finally, a brief overall analysis is done.

The investigated ML classification methods are Random Forest Classifier (RFC), Decision Trees Classifier (DTC), k-Nearest Neighbour Classifier (kNN), Multi Layer Perceptron Classifier (MLPC) and Linear Support Vector Classifier (LSVC).

5.1 Pre-processing and Feature Selection

As discussed in chapter 4, the data sets are *COH37, COS36, COHS37, HardSet* and include 70 features each, i.e., the feature space $\mathcal{F}_{70}$. Pre-processing involves data discretization. Additionally, the feature selection method is also described.

5.1.1 Discretization

For discretization, we have implemented *KBinsDiscretizer* (KBD) technique from the Python pre-processing library [PVG+11]. It is used to transform continuous data and into intervals. The number of bins used is 15. Data discretization allows certain machine learning techniques to perform better due to the transformation of continuous features to discrete values [1]. A better accuracy for discretized data over non-discretized data was obtained using kNN, MLPC and LSVC classifiers in our case. The method of discretization was implemented along with non-discretized data to analyse the effects of discretization.

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

5.1.2 Feature Selection

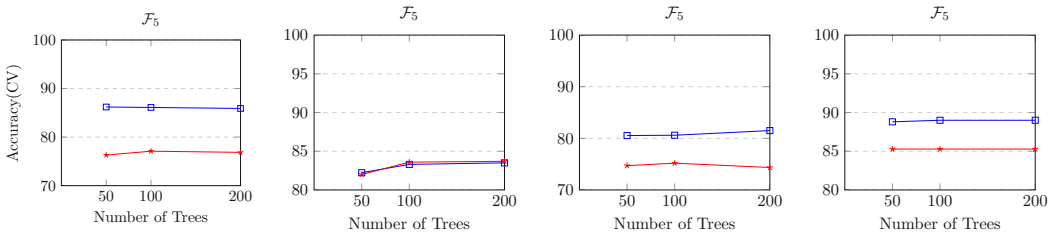
For feature Selection, the implementation of *SelectKBest* (SKB) technique from the Python library [PVG+11] has been implemented. It is a filter based feature selection method. The feature selection method selects the k best features according to statistical feature scoring. Information gain [KSG04, Ros14, PVG+11] is used for feature scoring. The feature scores are done by comparing each feature individually to the target data [KSG04, Ros14, PVG+11]. The results achieved by using SKB will be discussed in the further sections. The experiments below have been conducted using k-values of 5 ($\mathcal{F}_5$), 15 ($\mathcal{F}_{15}$) and 25 ($\mathcal{F}_{25}$).

5.2 Algorithm Selection Results

In this section, the classification results are presented. A Cross Validation (CV) for the entire experimentation process was used with a number of repeats to 10, i.e. 10-fold cross validation.

5.2.1 Random Forest Classifier

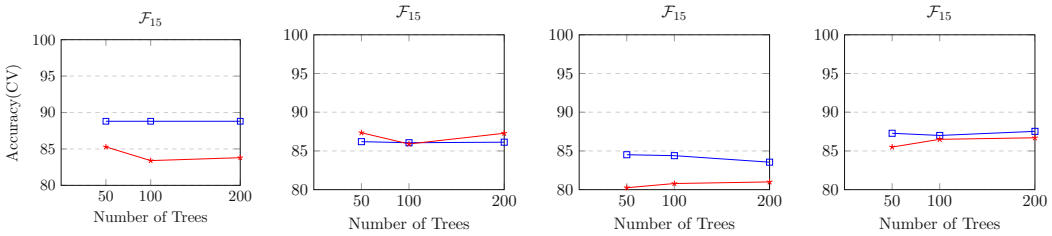
The parameter number of trees was tested for RFC. RFC obtained an overall good accuracy for feature $\mathcal{F}_5$ for the HardSet. The result for which is depicted in Figure 5.1. For discretized data the accuracy was lower than that of non-discretized data for all parameter values. The discretized and non-discretized data of COH37 and COS36 had similar accuracy for all number of trees tested 5.1. The non-discretized data had a continuously higher accuracy for COHS37 5.1.



(a) COH37 (b) COS36 (c) COHS37 (d) HardSet

Figure 5.1: Random forest classifier: Accuracy for non-discretized data is presented in blue and discretized data is presented in red. The k-value for SKB is 5 and the number of trees generated is 50, 100 and 200.

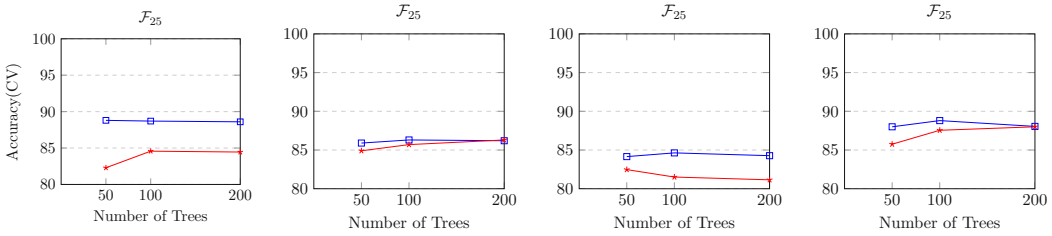
A good accuracy for non-discretized data sets of *COH37, COS36 and COHS37* with $\mathcal{F}_{15}$ was obtained. The overall result can be viewed in Figure 5.2. The discretized data's accuracy was almost similar to the non-discretized for $\mathcal{F}_{15}$ with data sets COS36 and HardSet. The non-discretized data of COH37 and COHS37 had a better accuracy continuously over discretized data.



(a) COH37 (b) COS36 (c) COHS37 (d) HardSet

Figure 5.2: Random forest classifier: Accuracy for non-discretized data is presented in blue and discretized data is presented in red. The k-value for SKB is 15 and the number of trees generated is 50, 100 and 200.

The overall result for $\mathcal{F}_{25}$ is depicted in Figure 5.3. For the other data sets the accuracy for non-discretized data was almost always higher. The exception being, the COS36 and HardSet with 200 trees.



(a) COH37 (b) COS36 (c) COHS37 (d) HardSet

Figure 5.3: Random forest classifier: Accuracy for non-discretized data is presented in blue and discretized data is presented in red. The k-value for SKB is 25 and the number of trees generated is 50, 100 and 200.

Based on the experiments conducted using RFC classifier the non-discretized data set was giving an overall higher accuracy consistently with all tree value, except for the HardSet. The parameter values changes did effect the accuracy of the classifier and so did the $\mathcal{F}_k$ value.

5.2.2 Decision Trees

The parameter value maximum depth of the tree was tested in the case of DTC. The results of DTC is depicted in Figure 5.4. Discretized data sets accuracy was consistently observed to be lower than that of non-discretized data for all data sets. As for the other accuracy observations based on $\mathcal{F}_k$ value, the data sets had varying accuracy.

COH37 data set observed high accuracy for $\mathcal{F}_{25}$. The *COS36* data set observed a high accuracy for $\mathcal{F}_5$. For *COHS37* data set, a high accuracy was observed for $\mathcal{F}_{25}$. The

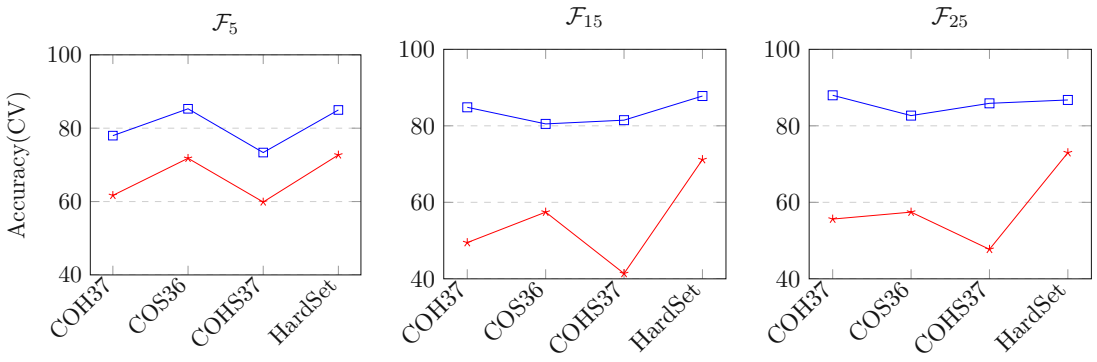


Figure 5.4: Decision trees classifier: Accuracy for non-discretized is presented in blue with depth 3. In red is presented the accuracy for the discretized data with depth 3.

HardSet noted the highest accuracy for $\mathcal{F}_{15}$. There was variation in the accuracy with the change in k-values. The accuracy for each considered case is depicted in Figure 5.4.

5.2.3 K-Nearest Neighbour Classifier

The parameter value number of neighbours was investigated in the case of kNN. The results of kNN is depicted in Figure 5.5. The discretized data showed continuously higher accuracy for all data sets with $\mathcal{F}_{15}$ and $\mathcal{F}_{25}$.

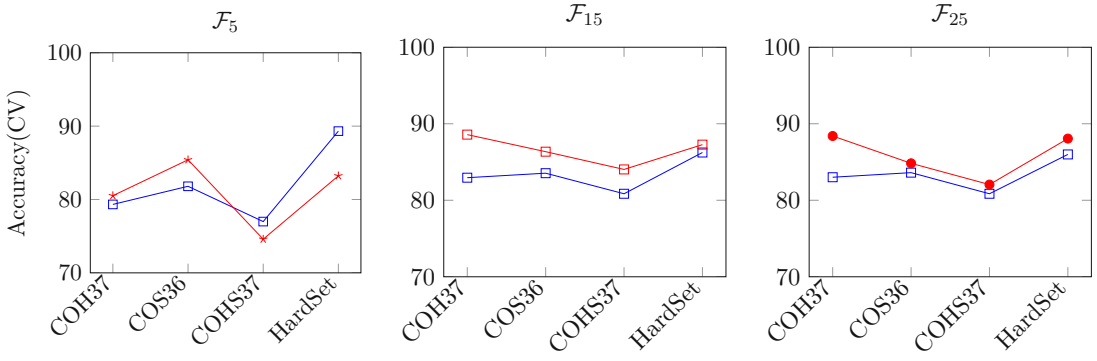


Figure 5.5: kNN classifier : Accuracy for non-discretized data is presented in blue with 3 neighbours. Discretized data accuracy is presented in red star with 3 neighbours.

The *COHS37* and *HardSet* data set observed higher accuracy for $\mathcal{F}_5$ with non-discretized data. Whereas *COS36* and *COH37* data set observed a high accuracy for $\mathcal{F}_5$. The accuracy for each considered case is depicted in Figure 5.5. The discretization of data was beneficial in the case of kNN.

5.2.4 Multi Layer Perceptron Classifier

The MLPC method was examined with parameter value of maximum iterations. The accuracy for 3 discretized data was observed to be higher than that of non-discretized data 5.6.

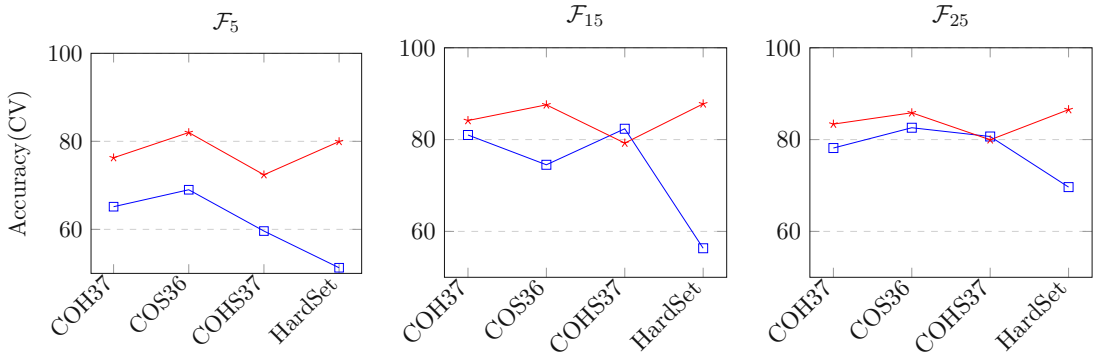


Figure 5.6: MLPC: Accuracy for non-discretized data is depicted in blue with maximum iterations of 1000. In red is depicted the discretized data accuracy with maximum iterations of 1000.

The MLPC method proved very promising for discretized data and for k-value of $\mathcal{F}_{15}$ and $\mathcal{F}_{25}$. The accuracy for non-discretized data sets were low, specially for $\mathcal{F}_5$. The accuracy results are depicted in Figure 5.6. The accuracy for *HardSet*, *COH37 and COHS37* was observed higher with $\mathcal{F}_{15}$. For *COHS37* non-discretized data had higher accuracy for $\mathcal{F}_{15}$ and $\mathcal{F}_{25}$.

5.2.5 Linear Support Vector Classifier

The parameter tested for Linear Support Vector Classifier (LSVC) was maximum iterations. The accuracy for *COH37, COS36 and COHS37* was higher in the case of discretized data. The accuracy was higher for HardSet in the case of non-discretized data. The higher accuracy values was observed in the case of $\mathcal{F}_{15}$. The result of LSVC is depicted in Figure 5.7. The LSVC classifier proved to be very good for non-discretized data of HardSet.

5.2.6 Effects of Discretization

As mentioned, KBD pre-processing having a bin number of 15 was used. In our experiments with discretized and non-discretized data, there was a variation in results. The accuracy for discretized data was higher in the many cases for classifiers kNN, MLPC and LSVC. In LSVC the Hardset observed an accuracy lower in discretized data continuously.

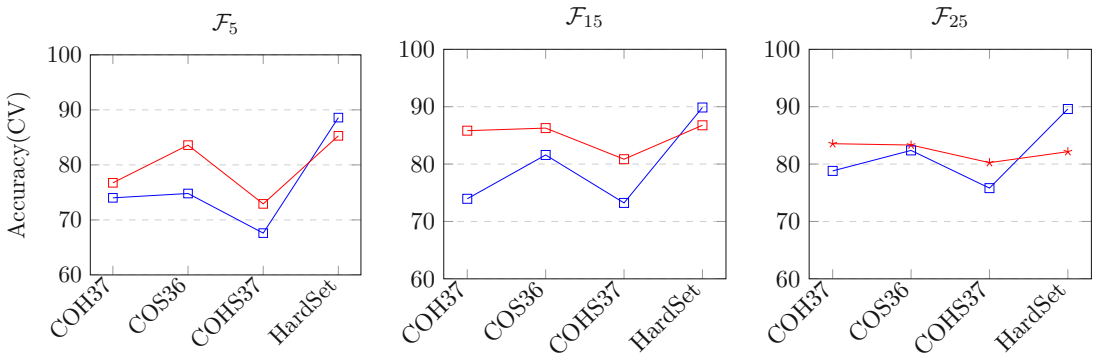


Figure 5.7: LinearSVC accuracy: In blue is depicted the accuracy for non-discretized data with maximum iteration of 1000 and in red for discretized data with maximum iteration of 1000.

The table 5.1 shows the case of MLPC and the percent of change in accuracy due to discretization for $\mathcal{F}_{25}$. The result high performing discretized data is depicted in Figure 5.8. For COHS37 data set the results were different 5.8.

	Discretized Accuracy	Non-Discretized Accuracy	Change in Accuracy
COH37	83.37	78.16	5.21
COS36	85.84	82.58	3.26
COHS37	79.95	80.67	-.72
HardSet	86.52	69.65	16.87

Table 5.1: MLP classifier accuracy changes from non-discretized to discretized data for k-value of 25.

5.2.7 Comparison of Classifiers

Just like it is important to know whether our AS approach is robust, it is of importance to know the efficient classifiers. Here we will present the highest accuracy for each data set corresponding to each classification method. This is presented in the Figure 5.9.

5.2.8 Analysis using the Confusion Matrix

Finally for the analysis of confusion matrix, the non-discretized data sets were considered. The goal of this was to show that our machine learning approach selected the best performing algorithm in most number of cases. The data sets were split into a training, testing set of 70% and 30%, respectively.

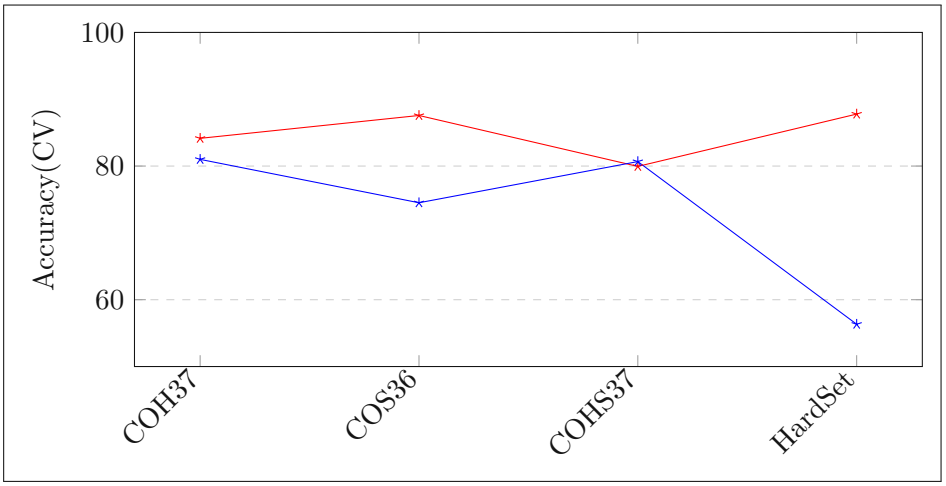


Figure 5.8: Highest accuracy for discretized vs non-discretized data with MLPC: The highest accuracy for discretized data is presented in red. Accuracy for the same parameters for non-discretizeed data is presented in blue.

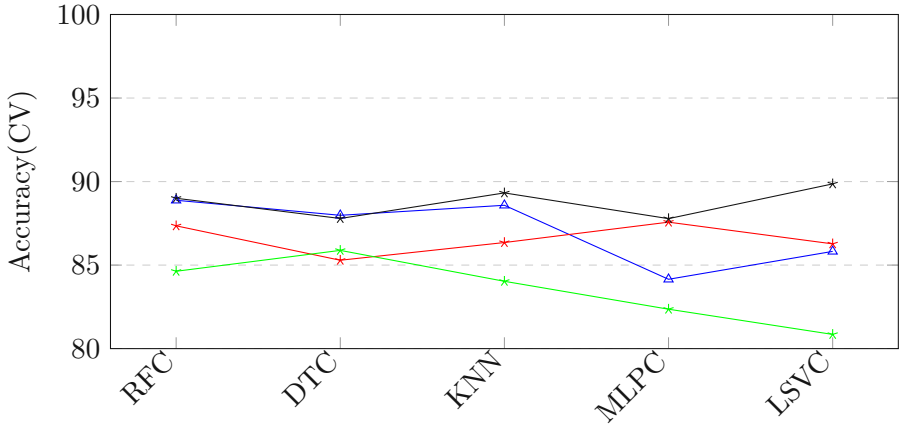


Figure 5.9: Highest accuracy for all machine learning classifiers for data *COH37* in blue, *COS36* in red, *COHS37* in green and *HardSet* in black. The results presented are with $\mathcal{F}_5$, $\mathcal{F}_{15}$ or $\mathcal{F}_{25}$.

5.2.9 COH37

The RFC with number of trees 100 was tested. An accuracy of 92% was observed, which included a high precision and recall value for all solvers. which was above 90% for CHUF and ORTL and a recall value 0f 83% for HYILS. The confusion matrix is presented in Table 5.2.

Figure 5.10 shows that our RFC was able to select the best algorithm in most number of cases.

	CHUF	HYILS	ORTL
CHUF	172	0	10
HYILS	2	53	9
ORTL	15	4	239

Table 5.2: Confusion matrix using RFC for the data set COH37.

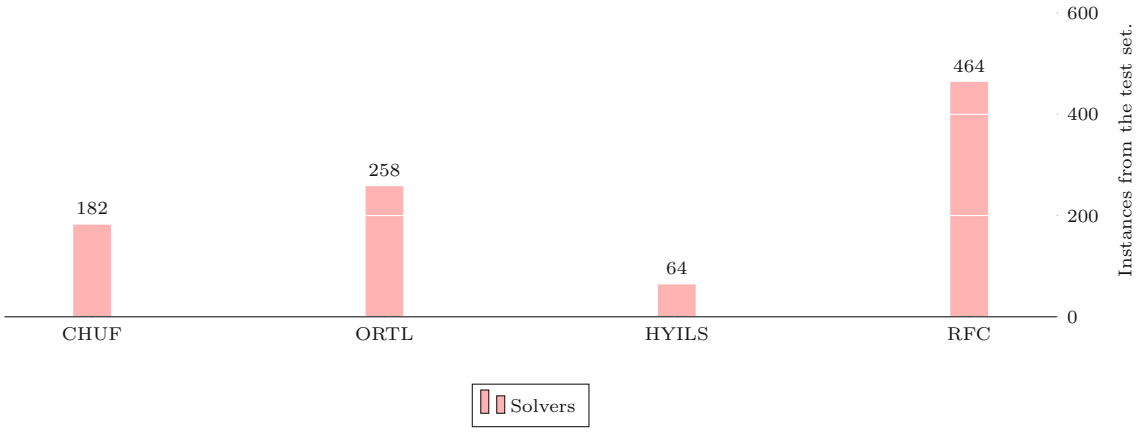


Figure 5.10: Instances solved most efficiently for the solvers and number of times best solver selected by AS approach using RFC for the test case of COH37.

5.2.10 COS36

The DTC with a maximum depth of 3 was investigated. An accuracy of 94% was observed. A high precision and recall value for all solvers was achieved, above 90% for CHUF, ORTL and SIMA. The confusion matrix can be seen in the Table 5.3.

	CHUF	ORTL	SIMA
CHUF	134	17	0
ORTL	5	221	4
SIMA	0	4	34

Table 5.3: Confusion matrix using DTC for the data set COS36.

Similarly, Figure 5.11 shows that DTC was able to select the best Sudoku solver in most number of cases.

5.2.11 COHS37

The DTC with maximum depth of 5 was investigated. The accuracy obtained was 93%. For ORTL, CHUF and SIMA the precision and recall was over 90%, HYILS obtained a precision and recall over 61%. The confusion matrix can be seen in the Table 5.4.

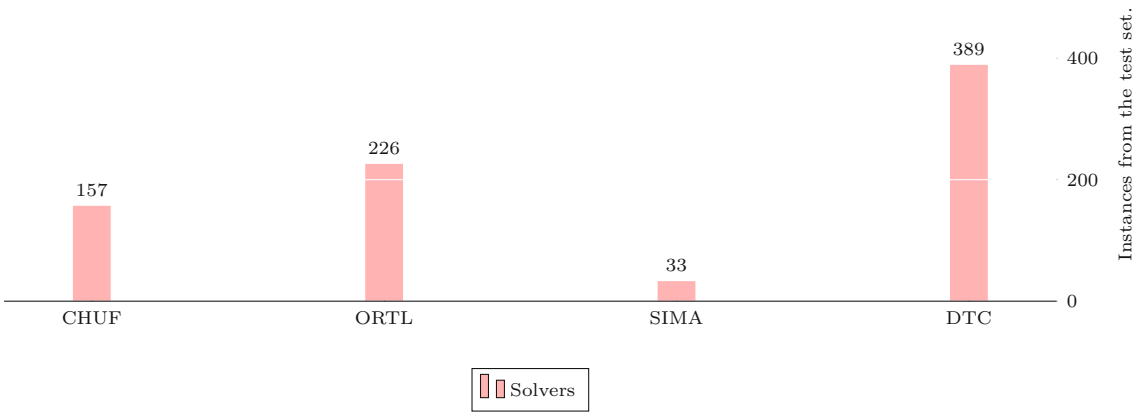


Figure 5.11: Instances solved most efficiently for the solvers and number of times best solver selected by AS approach using DTC for the test case of COS36.

	CHUF	HYILS	ORTL	SIMA
CHUF	178	0	14	0
HYILS	0	11	6	1
ORTL	11	3	243	0
SIMA	0	1	0	32

Table 5.4: Confusion matrix using DTC for the data set COHS37.

Similarly, Figure 5.12 shows that DTC was able to select the best performing algorithm for most number of Sudoku instances.

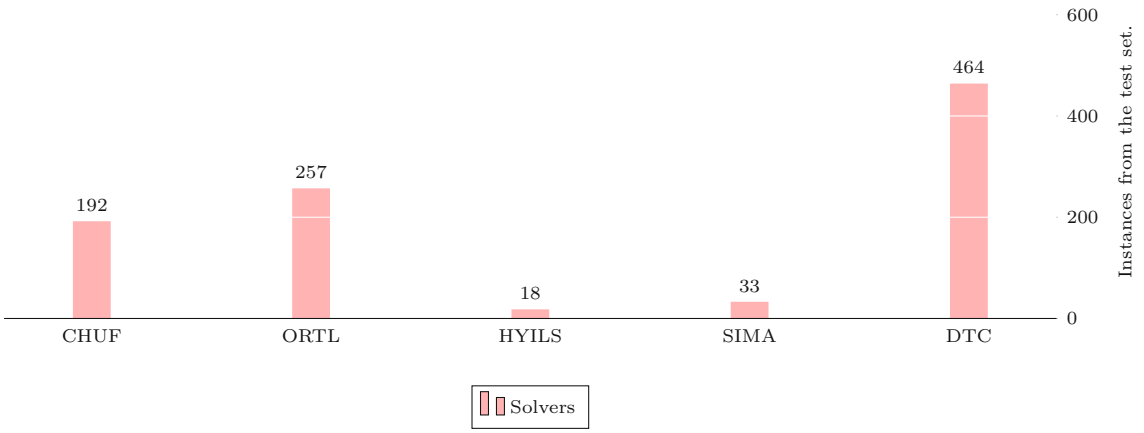


Figure 5.12: Instances solved most efficiently for the solvers and number of times best solver selected by AS approach using DTC for the test case of COHS37.

5.2.12 HardSet

The non-discretized data set *HardSet* using the RFC with number of trees as 100 was considered. The accuracy obtained was 92%. The precision and recall for HYILS was over 64%. For ORTL and CHUF the precision and recall was over 90%. The confusion matrix can be seen in the Table 5.5.

	CHUF	HYILS	ORTL
CHUF	39	0	3
HYILS	0	7	1
ORTL	1	4	63

Table 5.5: Confusion matrix using RFC for the HardSet.

Similarly, Figure 5.13 shows that our RFC was able to select best performing algorithm for most number of number of instances.

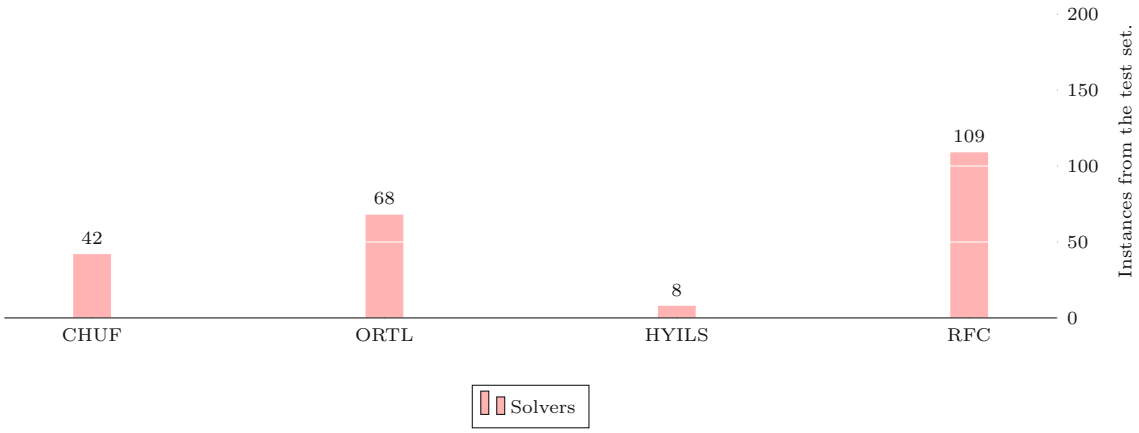


Figure 5.13: Instances solved most efficiently for the solvers and number of times best solver selected by AS approach using RFC for the test case of HardSet.

5.3 Overall Analysis

The experiments with different $\mathcal{F}_k$ values noted a variation in accuracy over all data sets. The accuracy values for $\mathcal{F}_5$ were low as compared to the other k-values. The effects of discretization was seen to be good for kNN, MLPC and LSVC classifier.

The 15 highest scoring features for classification based on SKB are presented in Table 5.6. The features are in the Table 5.6 is for COHS37. The $\mathcal{F}_{15}$ features for the COS36 and COH37 were similar to COHS37 data set using SKB except for one exception. HardSet also had the feature importance of $\mathcal{MC}_{rc}$. Overall the features were included from each

class of features $\mathcal{C}_x$ such as graph coloring features, Sudoku board features and flatzinc from MiniZinc.

The result of the best classifiers has been depicted in Figure 5.9. The accuracy indicates that the ML method was able to select the best performing algorithm in many cases.

	Feature Name		Feature Name
1.	$\mathcal{GS}_{mne}$	9.	$\mathcal{NS}_{pp}$
2.	$\mathcal{GS}_{nmme}$	10.	$\mathcal{NS}_{sdsc}$
3.	$\mathcal{NS}_{ramtr}$	11.	$\mathcal{NS}_{sdsr}$
4.	$\mathcal{NS}_{ramtc}$	12.	$\mathcal{NS}_{sdssg}$
5.	$\mathcal{NS}_{ramtr}$	13.	$\mathcal{NS}_{srp}$
6.	$\mathcal{NS}_{ramtsg}$	14.	$\mathcal{FZ}_{chi}$
7.	$\mathcal{GS}_{mn}$	15.	$\mathcal{NS}_{alr}$
8.	$\mathcal{GS}_{n}$		

Table 5.6: $\mathcal{F}_{15}$ features for algorithm selection of *COHS37*.

Figure 5.10 shows that AS using RFC technique selects the best performing algorithm in most number of cases. Similarly, Figures 5.11, 5.12 and 5.13 shows high accuracy for AS.

CHAPTER 6

Evaluation of Run-time Prediction

In this chapter, the algorithm Run-time Prediction (RP) of the solvers based on six different regression methods is presented. The analysis of the RP will be done based on algorithms from the algorithm space $\mathcal{A} = \{CHUF, ORTL, SIMA, HYILS\}$, in that order. Data sets have been formed for each individual solver from $\mathcal{A}$ to analyse the RP. They are based on the order of the puzzles $n \in \{3, \ldots, 9\}$.

The chapter is arranged into 3 sections; Pre-processing, Regression Results and Overall analysis.

The pre-processing step includes outlier detection and removal. Thereafter, the regression based Machine Learning (ML) methods are investigated. The regression methods investigated are Random Forest Regressor (RFR), Decision Tree Regressor (DTR), Linear Supprt Vector Regressor (LSVR), Extreme Gradient Boosting Regressor (XGBR), Ridge Regressor (RR) and Gradient Boosting Regressor (GBR).

6.1 Pre-processing

Pre-processing for regression involves outlier detection and removal. Two automatic outlier detection methods from the python library [PVG+11] had been tried and result of one has been presented.

6.1.1 Outliers: Detection and Removal

For outlier detection, we chose to investigate between two automatic methods; isolation forest [LTZ08] and local outlier factor [BKNS00]. The methods have been implemented

in python library [PVG+11] as class IsolationForest [1] and LocalOutlierFactor [2].

Isolation factor is a tree based outlier detection method and is computationally not expensive for data sets having large feature space. Outliers in isolation factor are detected based on whether they are few in number and having attribute values that are different from normal instances [LTZ08]. Local outlier factor is useful for data sets having lower feature space and outliers are detected based on nearest neighbours and their degree of outlier-ness [BKNS00].

Since, the feature space for Sudoku puzzles computed for the purpose of RP is not large, computing time is not a concern. Therefore, we decided to use local outlier factor as it is useful for outlier detection in data sets with lower feature space. The results presented are based on local outlier factor.

6.2 Regression Results

In this section, the regression results of our data sets are presented. An evaluation of the regression methods is done based on Root Mean Square Error (RMSE), Pearson's Correlation Coefficient (PCC) and scatter plots. The ML methods were expected to predict the computation time with minimum error (RMSE), a RMSE value of 0 shows over fitting. The PCC score is expected to in between 0 and 1, the better predictions show value closer to 1 (exactly 1 is a over fitting situation). The presented results for regression are obtained with all the features, i.e., $\mathcal{F}_{70}$. The data sets were split into a training, testing set of 70% and 30%, respectively.

6.2.1 CHUF39

The CHUF solver's RP based on RMSE scores are depicted in Figure 6.1. Six different regression techniques were tested, RFR and GBR show low RMSE scores. Higher RMSE scores were noted for DTR.

The predicted time versus test time was plotted using scatter plots and PCC was also observed. The scatter plot of predicted time vs test time for all regression techniques used can be seen in Figure 6.2. The PCC score is depicted in Figure 6.3.

The PCC scores were not good for RP of CHUF. A maximum PCC was observed using XGBR with score of 0·435.

6.2.2 ORTL39

The RMSE scores for the RP of ORTL are depicted in Figure 6.4. Six different regression techniques were tested. LSVR observed the highest RMSE score, whereas RFR observed

[1] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

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

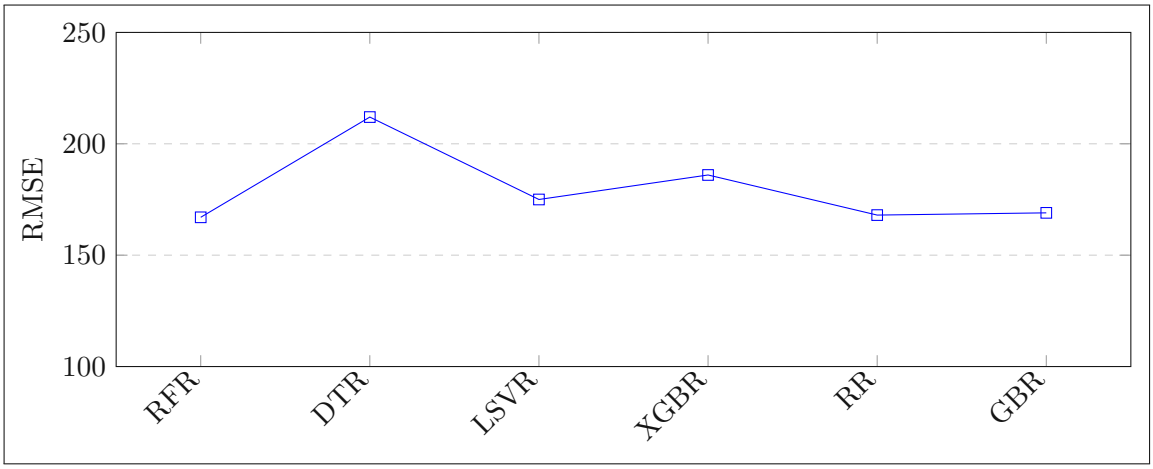
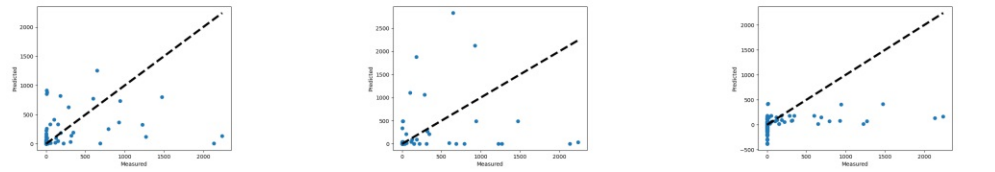
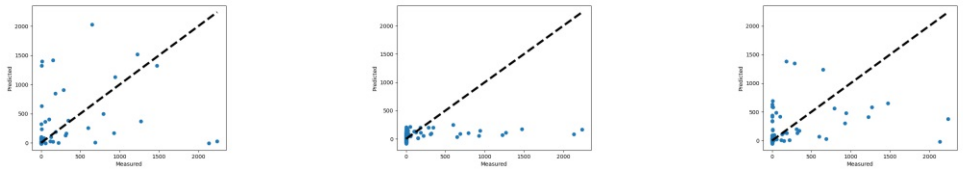


Figure 6.1: CHUF39 RMSE scores: The number of trees for RFR was 100, maximum depth for DTR, 3 and maximum iteration of 1000 for LSVR.



(a) Random Forest (b) Decision Tree (c) Linear SV Regressor



(d) XGBoost (e) Ridge Regressor (f) Gradient Boost

Figure 6.2: Scatter plot for CHUF39: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR.

the lowest RMSE score.

ORTL's predicted versus test scatter plot shows correlation with the RMSE scores. This can be observed with the RMSE score of LSVR and the scatter plot of LSVR. The scatter plot of predicted time versus test time for all regression techniques is depicted in Figure 6.5. The PCC scores can be observed in Figure 6.6. The maximum PCC score for ORTL was obtained using RFR with score of 0·614.

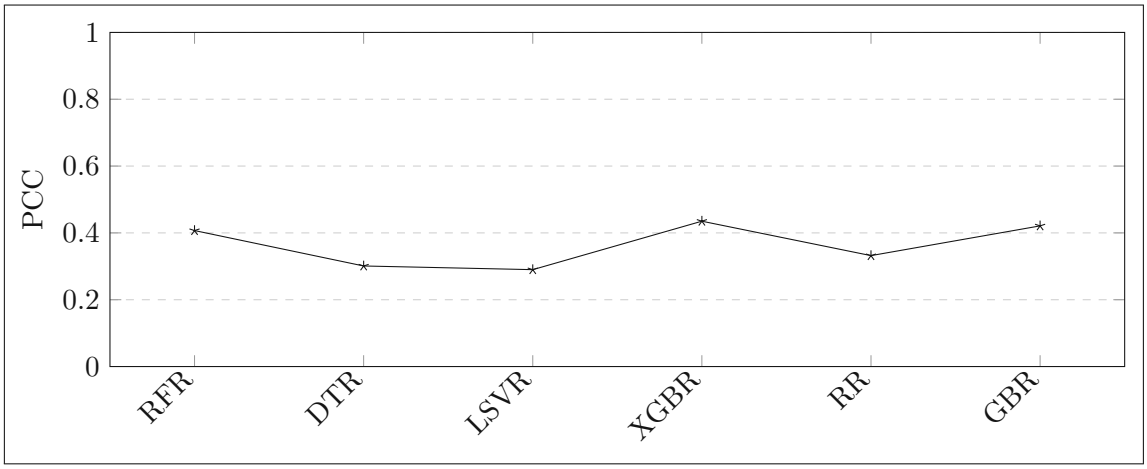


Figure 6.3: PCC scores for CHUF39: The PCC score was above 0·4 for RFR, XGBR and GBR. The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR.

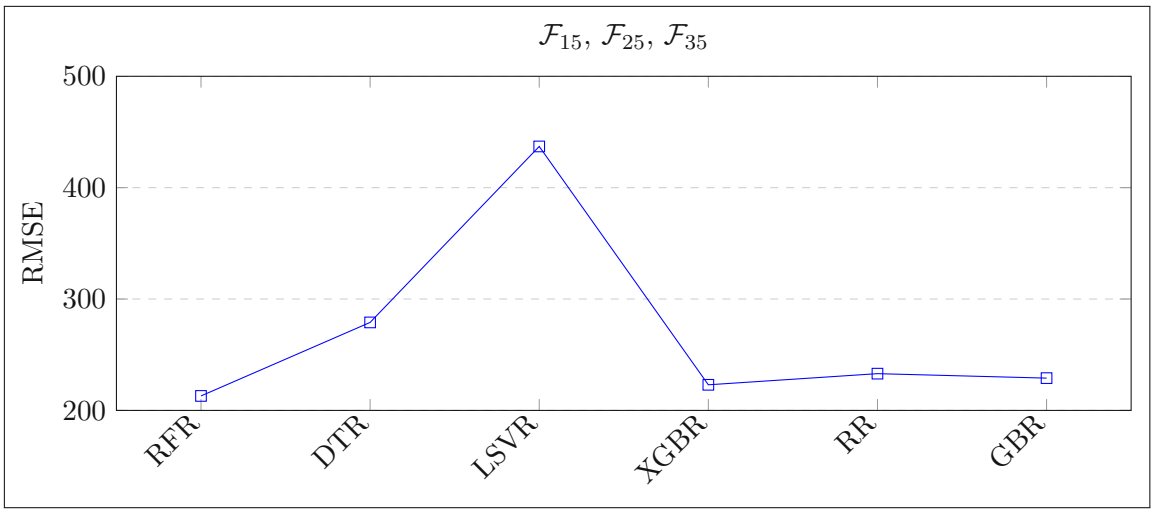
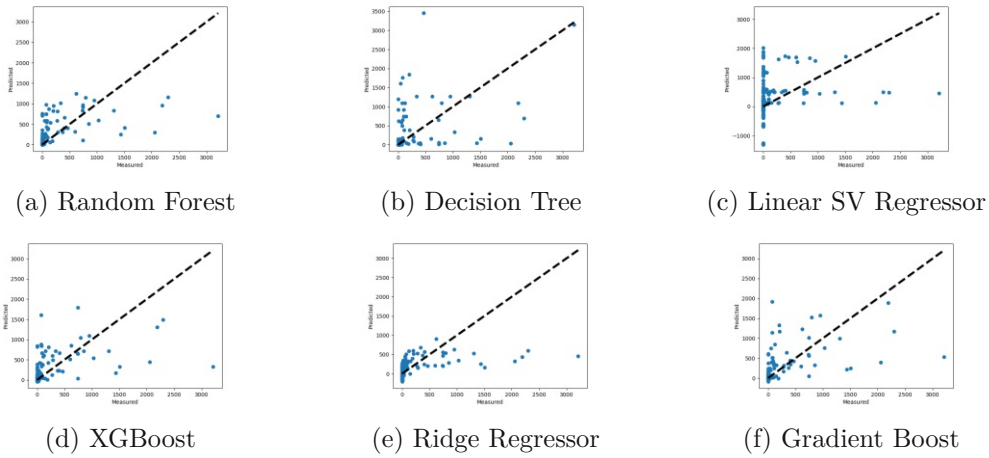


Figure 6.4: ORTL39 RMSE scores: The number of trees for RFR was 100, maximum depth for DTR, 3 and maximum iteration of 1000 for LSVR.

6.2.3 SIMA39

SIMA's RMSE scores are depicted in Figure 6.7. Six different regression techniques were tested, RFR observed the lowest RMSE score. The RMSE score of LSVR was highest.

SIMA's predicted vs test scatter plot shows good correlation between RMSE scores and scatter plots. The scatter plot of predicted versus test time for all regression techniques used can be seen in Figure 6.8. The PCC can be observed in Figure 6.9. A good PCC score of above 0·90 was observed throughout the experimentation of SIMA39. There was



(a) Random Forest (b) Decision Tree (c) Linear SV Regressor

(d) XGBoost (e) Ridge Regressor (f) Gradient Boost

Figure 6.5: Scatter plot ORTL solver: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR.

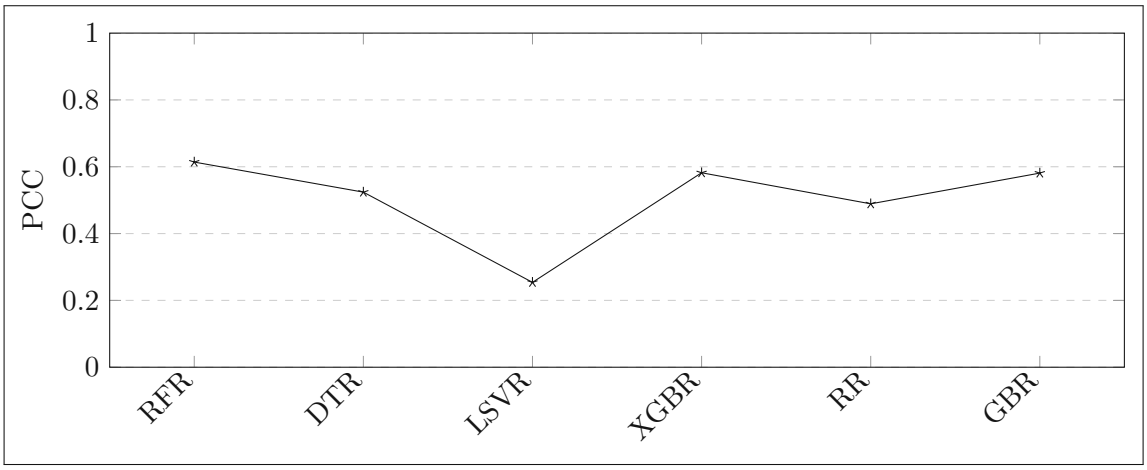


Figure 6.6: ORTL's PCC scores: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR.

good correlation between the scatter plots and PCC scores.

6.2.4 HYILS39

RMSE scores for HYILS are depicted in Figure 6.10. Six different regression techniques were tested out of which RFR observed the lowest RMSE score. LSVR observed the highest RMSE score.

The scatter plot of predicted versus test time for all regression techniques is depicted in

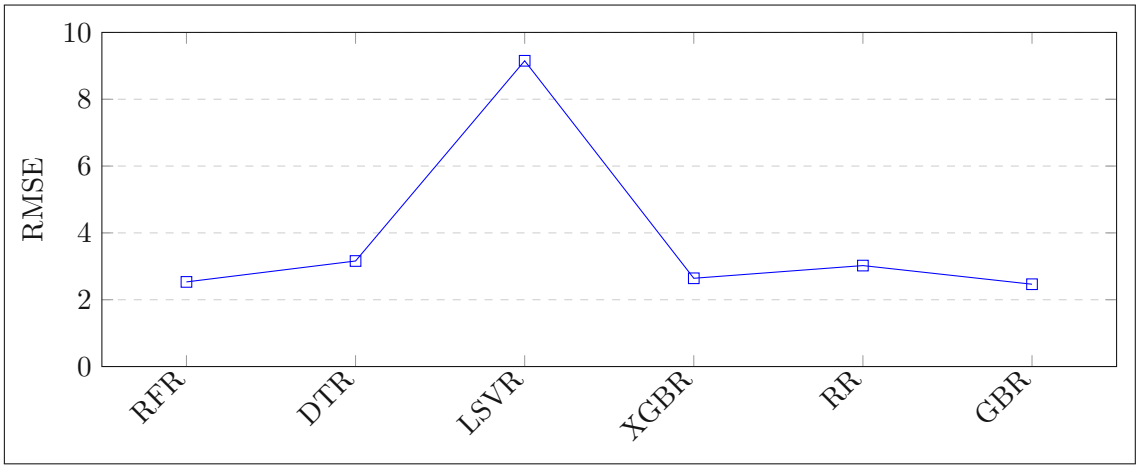
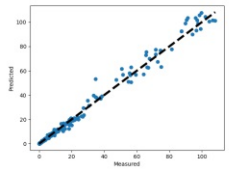
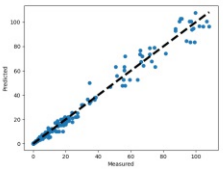


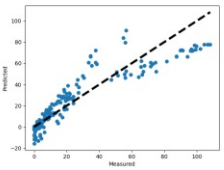
Figure 6.7: SIMA39 RMSE scores: The number of trees for RFR was 100, maximum depth for DTR, 3 and maximum iteration of 1000 for LSVR.



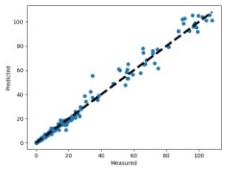
(a) Random Forest



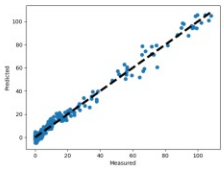
(b) Decision Tree



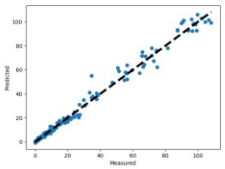
(c) Linear SV Regressor



(d) XGBoost



(e) Ridge Regressor



(f) Gradient Boost

Figure 6.8: Scatter plot SIMA39: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR.

Figure 6.11. The PCC can be observed in Figure 6.12. The PCC score was high for RFR, XGBR and GBR. There was correlation between the scatter plots and PCC scores.

6.3 Overall Analysis

The RMSE, PCC scores and scatter plots presented above are the basis for our analysis. The scatter plots show a good relationship between the predicted versus test computation time for SIMA39 and HYILS39 to a good extent. The PCC scores for CHUF39 was not

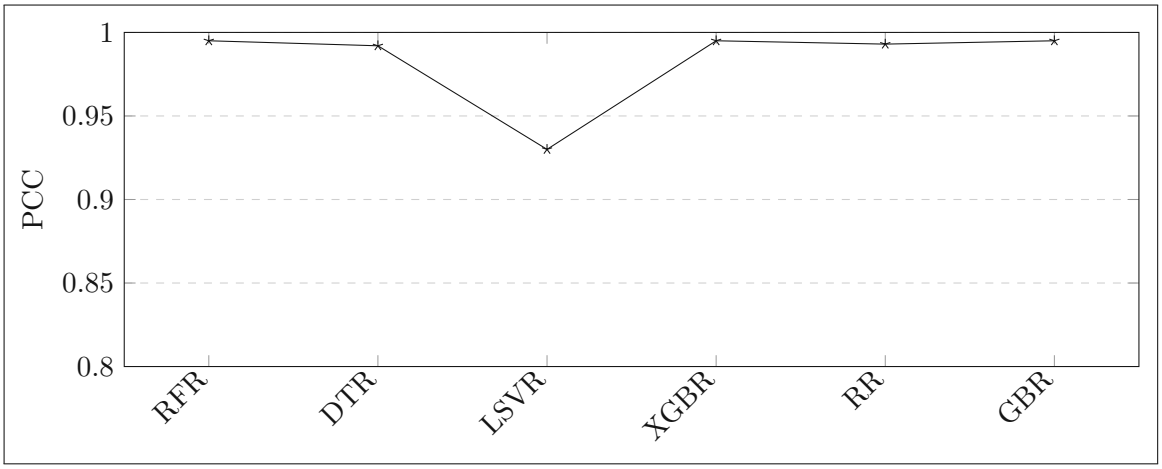


Figure 6.9: PCC Scores for SIMA39: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DT and maximum iterations of 1000 for LSVR.

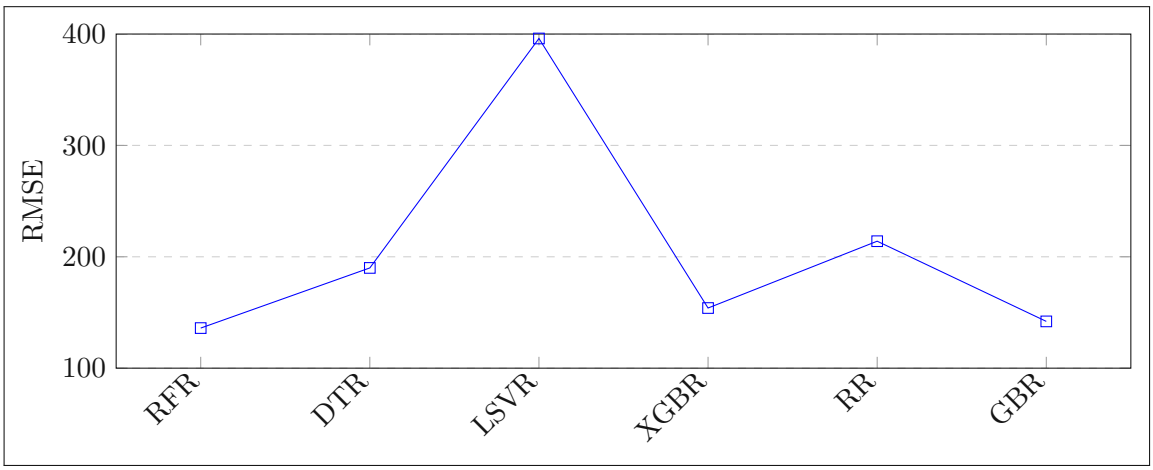
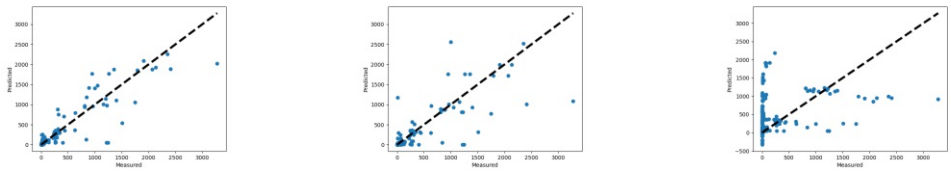


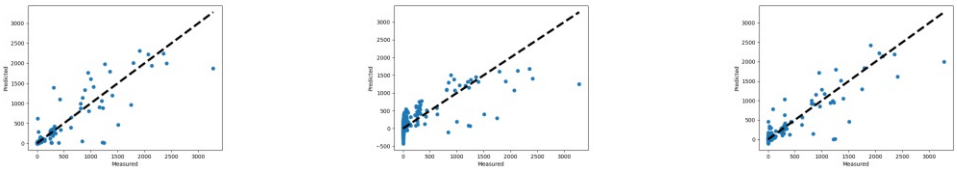
Figure 6.10: HYILS39 RMSE Scores: The number of trees for RFR was 100, maximum depth for DTR, 3 and maximum iteration of 1000 for LSVR.

good for any of the regression methods and always remained below 0·5. The PCC scores for ORTL39 was satisfactory and was highest for RFR.

The regression technique that performed well were XGBR, GBR and RFR consistently, based on low RMSE scores. The PCC values depicted a correlation with the scatter plots.



(a) Random Forest (b) Decision Tree (c) Linear SV Regressor



(d) XGBoost (e) Ridge Regressor (f) Gradient Boost

Figure 6.11: Scatter plot for HYILS39: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR.

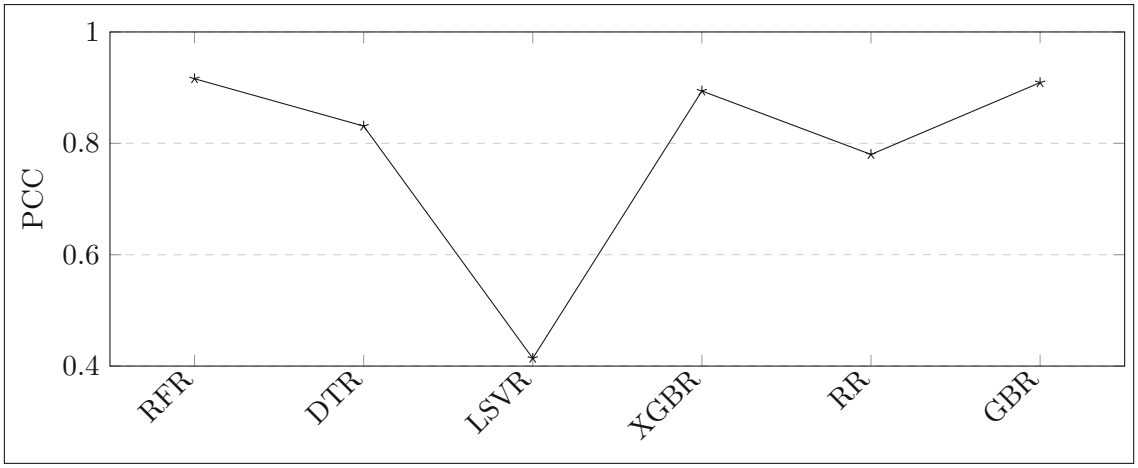


Figure 6.12: HYILS39 PCC : LSVR obtained the lowest PCC score and the correlation can be noticed in the scatter plot in Figure 6.11 (c), where as RFR obtained the highest PCC score.

6.3.1 Feature Selection

As we know from the previous chapter that SKB selects the k best features according to feature scoring. Feature scoring is done using mutual information regression [KSG04, Ros14, PVG+11]. The regression models are not presented based on feature selection but we will present the 25 highest scoring features using SKB. We will denote it as $\mathcal{F}_{25}$.

The $\mathcal{F}_{25}$ for our data set CHUF39 is presented in Table 6.1. From the table it can be seen that the top 25 features were selected from most of the class of features that we

described in Chapter 3, such as graph size ($\mathcal{GS}$), Sudoku puzzle features ($\mathcal{NS}$), order ($\mathcal{O}$), chromatic number (χ), maximal clique ($\mathcal{MC}$) and flatzinc ($\mathcal{FZ}$).

	Feature Name		Feature Name
1.	$\mathcal{NS}_{alc}$	14.	$\mathcal{NS}_{alr}$
2.	$\mathcal{NS}_{alsg}$	15.	$\mathcal{NS}_{iqr}$
3.	$\mathcal{NS}_{slc}$	16.	$\mathcal{NS}_{slr}$
4.	$\mathcal{NS}_{slsg}$	17.	χ_{mc}
5.	$\mathcal{MC}_{mc}$	18.	$\mathcal{GS}_{mdb}$
6.	$\mathcal{GS}_{mne}$	19.	$\mathcal{FZ}_{chi}$
7.	$\mathcal{GS}_{mn}$	20.	$\mathcal{GS}_{te}$
8.	$\mathcal{GS}_{nmme}$	21.	$\mathcal{FZ}_{chc}$
9.	$\mathcal{NS}_{ma}$	22.	$\mathcal{NS}_{sp}$
10.	$\mathcal{NS}_{mp}$	23.	$\mathcal{O}_{n^2}$
11.	$\mathcal{NS}_{nd}$	24.	$\mathcal{NS}_{maxon}$
12.	$\mathcal{GS}_{n}$	25.	$\mathcal{NS}_{srp}$
13.	$\mathcal{O}_{n}$		

Table 6.1: Top 25 features: Run-time Prediction for CHUF39.

CHAPTER 7

Conclusion

In this thesis, we have developed an approach towards algorithm selection of Sudoku puzzles. Run-time prediction for Sudoku puzzles using machine learning has also been investigated. For the purpose we have computed features based on domain knowledge of Sudoku puzzle, their graph coloring equivalence and flatzinc. Identification and computation of 70 features from Sudoku puzzle instances for algorithm selection and run-time prediction was done.

We investigated several Sudoku solvers and selected four state-of-the-art solvers for our further analysis. Four data sets for algorithm selection and four for run-time prediction were created. The data sets were based on evaluation of solvers, the order of the puzzles and hardness of the puzzle instances.

The machine learning methods investigated for classification of algorithms were RFC, DTC, kNNC, MLPC and LSVC. The algorithm selection approach selected the best performing algorithm in many cases.

The machine learning methods investigated for regression were RFR, DTR, LSVR, XGBR, RR and GBR. The four data sets were based on four Sudoku puzzle solvers. The predicted versus test time scatter plot showed good relationship in some of the cases, specifically for Sudoku solvers SIMA and HYILS. The RMSE scores were low for some regression techniques.

Finally, the algorithm selection method introduced for Sudoku puzzles was able to select the best performing Sudoku solver on many instances. The run-time prediction errors showed good correlation between predicted and test computation time for some machine learning methods.

7.1 Future Work

To the best of our knowledge, algorithm selection and run-time prediction for Sudoku puzzles were not inspected before. This thesis lays a foundation for considering other puzzles such as N-Queens, Latin squares and several other versions of Sudoku.

A deeper analysis on impact of proposed features and the development of additional features for Sudoku could also be considered in the future work.

List of Figures

2.1 Sudoku as a Graph Coloring Problem [AC18] 6
2.2 The Rice framework as adapted from Kate Smith-Miles [Ric76, SM09] . . 12

3.1 Average clause generation computing time for SAT w.r.t. the order of puzzle. 22

4.1 Percent of puzzles solved for each order by each solver. 27
4.2 Total percent of puzzles solved for each order. 27
4.3 Percent of puzzles solved by each solver most efficiently for orders 3 to 7. 28
4.4 Percent of puzzles solved by each solver most efficiently for orders 3 to 7. 28
4.5 Percent of puzzles solved by each solver most efficiently for orders 3 to 6. 29
4.6 Percent of puzzles solved by each solver most efficiently for orders 3 to 7. 29
4.7 Percent of puzzles solved by each solver most efficiently for orders 3 to 7, according to the hardness of puzzles described in Table 4.1. 30

5.1 Random forest classifier: Accuracy for non-discretized data is presented in blue and discretized data is presented in red. The k-value for SKB is 5 and the number of trees generated is 50, 100 and 200. 34
5.2 Random forest classifier: Accuracy for non-discretized data is presented in blue and discretized data is presented in red. The k-value for SKB is 15 and the number of trees generated is 50, 100 and 200. 35
5.3 Random forest classifier: Accuracy for non-discretized data is presented in blue and discretized data is presented in red. The k-value for SKB is 25 and the number of trees generated is 50, 100 and 200. 35
5.4 Decision trees classifier: Accuracy for non-discretized is presented in blue with depth 3. In red is presented the accuracy for the discretized data with depth 3. 36
5.5 kNN classifier : Accuracy for non-discretized data is presented in blue with 3 neighbours. Discretized data accuracy is presented in red star with 3 neighbours. 36
5.6 MLPC: Accuracy for non-discretized data is depicted in blue with maximum iterations of 1000. In red is depicted the discretized data accuracy with maximum iterations of 1000. 37

5.7 LinearSVC accuracy: In blue is depicted the accuracy for non-discretized data with maximum iteration of 1000 and in red for discretized data with maximum iteration of 1000. 38
5.8 Highest accuracy for discretized vs non-discretized data with MLPC: The highest accuracy for discretized data is presented in red. Accuracy for the same parameters for non-discretizeed data is presented in blue. 39
5.9 Highest accuracy for all machine learning classifiers for data *COH37* in blue, *COS36* in red, *COHS37* in green and *HardSet* in black. The results presented are with $\mathcal{F}_5$, $\mathcal{F}_{15}$ or $\mathcal{F}_{25}$. 39
5.10 Instances solved most efficiently for the solvers and number of times best solver selected by AS approach using RFC for the test case of COH37. . . 40
5.11 Instances solved most efficiently for the solvers and number of times best solver selected by AS approach using DTC for the test case of COS36. . . . 41
5.12 Instances solved most efficiently for the solvers and number of times best solver selected by AS approach using DTC for the test case of COHS37. . . 41
5.13 Instances solved most efficiently for the solvers and number of times best solver selected by AS approach using RFC for the test case of HardSet. . 42

6.1 CHUF39 RMSE scores: The number of trees for RFR was 100, maximum depth for DTR, 3 and maximum iteration of 1000 for LSVR. 47
6.2 Scatter plot for CHUF39: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR. 47
6.3 PCC scores for CHUF39: The PCC score was above 0·4 for RFR, XGBR and GBR. The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR. 48
6.4 ORTL39 RMSE scores: The number of trees for RFR was 100, maximum depth for DTR, 3 and maximum iteration of 1000 for LSVR. 48
6.5 Scatter plot ORTL solver: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR. 49
6.6 ORTL's PCC scores: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR. 49
6.7 SIMA39 RMSE scores: The number of trees for RFR was 100, maximum depth for DTR, 3 and maximum iteration of 1000 for LSVR. 50
6.8 Scatter plot SIMA39: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR. 50
6.9 PCC Scores for SIMA39: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DT and maximum iterations of 1000 for LSVR. 51

6.10 HYILS39 RMSE Scores: The number of trees for RFR was 100, maximum depth for DTR, 3 and maximum iteration of 1000 for LSVR. 51
6.11 Scatter plot for HYILS39: The parameter configuration used for regression methods were; 100 trees for RFR, maximum depth of 3 for DTR and maximum iterations of 1000 for LSVR. 52
6.12 HYILS39 PCC : LSVR obtained the lowest PCC score and the correlation can be noticed in the scatter plot in Figure 6.11 (c), where as RFR obtained the highest PCC score. 52

1 Feature Importance according to the table 1 for COH37 data set. 75
2 Feature Importance according to the table 1 for COS36 data set. 75
3 Feature Importance according to the table 1 for COHS37 data set. 76
4 Feature Importance according to the table 1 for HardSet data set. 76

5 Feature Importance according to the table 1 for CHUF39 data set. 77
6 Feature Importance according to the table 1 for ORTL39 data set. 78
7 Feature Importance according to the table 1 for SIMA39 data set. 78
8 Feature Importance according to the table 1 for HYILS39 data set. 79

9 Instances Solved most efficiently w.r.t. the Solvers and Algorithm Selection approach using RF ($\mathcal{F}_{25}$) for the test set of Easy data set. 82

List of Tables

3.1 Table of features related to sum, product and domain of Sudoku puzzle. . 19
3.2 Table of features related to row, column and sub-grid size of Sudoku puzzle. 20

4.1 Table Depicting hardness of the puzzle instances based on p-values and % puzzle filled with digits. 30

5.1 MLP classifier accuracy changes from non-discretized to discretized data for k-value of 25. 38
5.2 Confusion matrix using RFC for the data set COH37. 40
5.3 Confusion matrix using DTC for the data set COS36. 40
5.4 Confusion matrix using DTC for the data set COHS37. 41
5.5 Confusion matrix using RFC for the HardSet. 42
5.6 $\mathcal{F}_{15}$ features for algorithm selection of *COHS37*. 43

6.1 Top 25 features: Run-time Prediction for CHUF39. 53

1 Features: The features as computed for the data sets. 74

2 Confusion Matrix using RF with k-value 15 and number of trees 100 for the data set EasySet. 81

List of Algorithms

Acronyms

AS Algorithm Selection.

CHUF Chuffed.

CPLEX IBM-Cplex.

CV 10-Fold Cross Validation.

DTC Decision Tree Classifier.

DTR Decision Tree Regressor.

GBR Gradiant Boost Regressor.

GUROBI Gurobi.

HYILS Hybrid Iterated Local Search.

kNN K-Nearest Neighbour Classifier.

LSVC Linear Support Vector Classsifier.

LSVR Linear Support Vector Regressor.

ML Machine Learning.

MLPC Multi Layer Perceptron Classifier.

ORTL OR-Tools.

PCC Pearson's Correlation Coefficient.

RFC Random Forest Classifier.

RFR Random Forest Regressor.

RMSE Root Mean Square Error.

RP Run-time Prediction.

RR Ridge Regressor.

SIMA Simulated Annealing.

XGBR Extreme Gradiant Booster Regressor.

Bibliography

[AC18] Francisco J. Aragón Artacho and Rubén Campoy. Solving graph coloring problems with the douglas-rachford algorithm. *Set-Valued and Variational Analysis*, 26(2):277–304, January 2018.

[Apt03] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[BCLR08] Andrew C. BartlettTimothy, P. Chartier, Amy Nicole Langville, and Timothy D. Rankin. An integer programming model for the sudoku problem. In *Journal of Online Mathematics and its Applications*, Vol. 8, Article ID 1798, 2008.

[Bie08] Armin Biere. Picosat essentials. *J. Satisf. Boolean Model. Comput.*, 4(2-4):75–97, 2008.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 93–104, New York, NY, USA, 2000. Association for Computing Machinery.

[CdlBS10] Geoffrey Chu, Maria Garcia de la Banda, and Peter J. Stuckey. Automatically exploiting subproblem equivalence in constraint programming. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 71–86, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Cpl09] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[DDC99] M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 2, pages 1470–1477 Vol. 2, 1999.

[DG97] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.

[DYS20] R. V. Driel and N. Yorke-Smith. Towards unsatisfiable core learning for chuffed. In *CP'20 Workshop on Progress Towards the Holy Grail*, 2020.

[FPHK94] F.J. Ferri, P. Pudil, M. Hatef, and J. Kittler. Comparative study of techniques for large-scale feature selection* *this work was suported by a serc grant gr/e 97549. the first author was also supported by a fpi grant from the spanish mec, pf92 73546684. In Edzard S. GELSEMA and Laveen S. KANAL, editors, *Pattern Recognition in Practice IV*, volume 16 of *Machine Intelligence and Pattern Recognition*, pages 403–413. North-Holland, 1994.

[GE03] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3(null):1157–1182, March 2003.

[GO21] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[Hay16] M. Haythorpe. Reducing the generalised sudoku problem to the hamiltonian cycle problem. *AKCE Int. J. Graphs Comb.*, 13:272–282, 2016.

[HDH+00] Adele E. Howe, Eric Dahlman, Christopher Hansen, Michael Scheetz, and Anneliese von Mayrhauser. Exploiting competitive planner performance. In *Recent Advances in AI Planning*, pages 62–72. Springer Berlin Heidelberg, 2000.

[HG08] James M. Hereford and Hunter Gerlach. Integer-valued particle swarm optimization applied to sudoku puzzles. In *2008 IEEE Swarm Intelligence Symposium*, pages 1–7, 2008.

[HM07] Agnes Herzberg and Ram Murty. Sudoku squares and chromatic polynomials. *Internationale Mathematische Nachrichten*, 54, 06 2007.

[Hol92] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[HXHLB14] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, January 2014.

[Kap10] A. Kapanowski. Python for education: the exact cover problem. *ArXiv*, abs/1010.5890, 2010.

[KE97] J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 5, pages 4104–4108 vol.5, 1997.

[Kot16] Lars Kotthoff. *Algorithm Selection for Combinatorial Search Problems: A Survey*, pages 149–190. Springer International Publishing, Cham, 2016.

[KSG04] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E*, 69(6), Jun 2004.

[LA20] Huw Lloyd and Martyn Amos. Solving sudoku with ant colony optimization. *IEEE Transactions on Games*, 12(3):302–311, 2020.

[LBHHX14] Kevin Leyton-Brown, Holger H. Hoos, Frank Hutter, and Lin Xu. Understanding the empirical hardness of np-complete problems. *Commun. ACM*, 57(5):98–107, May 2014.

[LBNS02] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, CP '02, page 556–572, Berlin, Heidelberg, 2002. Springer-Verlag.

[Lew07a] Rhyd Lewis. Metaheuristics can solve sudoku puzzles. *Journal of Heuristics*, 13(4):387–401, May 2007.

[Lew07b] Rhydian Lewis. On the combination of constraint programming and stochastic search: The sudoku case. In Thomas Bartz-Beielstein, María José Blesa Aguilera, Christian Blum, Boris Naujoks, Andrea Roli, Günter Rudolph, and Michael Sampels, editors, *Hybrid Metaheuristics*, pages 96–107, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[LO06] I. Lynce and J. Ouaknine. Sudoku as a sat problem. In *ISAIM*, 2006.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[MAT10] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[MK07] Timo Mantere and Janne Koljonen. Solving, rating and generating sudoku puzzles with ga. In *2007 IEEE Congress on Evolutionary Computation*, pages 1382–1389, 2007.

[MS13] Nysret Musliu and Martin Schwengerer. Algorithm selection for the graph coloring problem. In Giuseppe Nicosia and Panos Pardalos, editors, *Learning and Intelligent Optimization*, pages 389–403, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[MTL06] A. Moraglio, J. Togelius, and S. Lucas. Product geometric crossover for the sudoku puzzle. *2006 IEEE International Conference on Evolutionary Computation*, pages 470–476, 2006.

[MW17] Nysret Musliu and Felix Winter. A hybrid approach for the sudoku problem: Using constraint programming in iterated local search. *IEEE Intell. Syst.*, 32(2):52–62, 2017.

[NSB+07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[PSY09] Jaysonne A. Pacurib, Glaiza Mae M. Seno, and John Paul T. Yusiong. Solving sudoku puzzles using improved artificial bee colony algorithm. In *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pages 885–888, 2009.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[QS08] Haiyan Quan and Xinling Shi. On the analysis of performance of the improved artificial-bee-colony algorithm. In *2008 Fourth International Conference on Natural Computation*, volume 7, pages 654–658, 2008.

[Ric76] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[Ros14] Brian C. Ross. Mutual information between discrete and continuous data sets. *PLoS ONE*, 9(2):e87357, February 2014.

[RT12] J. Rosenhouse and L. Taalman. *Taking Sudoku Seriously: The Math Behind the World's Most Popular Pencil Puzzle*. Oxford University Press, United Kingdom, 2012.

[SCG+13] Ricardo Soto, Broderick Crawford, Cristian Galleguillos, Eric Monfroy, and Fernando Paredes. A hybrid ac3-tabu search algorithm for solving sudoku puzzles. *Expert Systems with Applications*, 40(15):5817–5821, 2013.

[SCG+15] Ricardo Soto, Broderick Crawford, Cristian Galleguillos, Fernando Paredes, and Enrique Norero. A hybridalldifferent-tabu search algorithm for solving sudoku puzzles. *Computational Intelligence and Neuroscience*, 2015:1–10, 2015.

[SFS+14] Peter J. Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The MiniZinc challenge 2008–2013. *AI Magazine*, 35(2):55–60, June 2014.

[Sim05] H. Simonis. Sudoku as a constraint problem. In *Workshop Modeling and Reformulating Constraint Satisfaction Problems*, pages 13–27, 2005.

[SM09] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1), January 2009.

[TL00] P. Torres and P. Lopez. Overview and possible extensions of shaving techniques for job-shop problems. In *Proceedings of the Second International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 181–186, 2000.

[vH01] Willem Jan van Hoeve. The alldifferent constraint: A survey. *CoRR*, cs.PL/0105015, 2001.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.

[WNA10] Yue Wu, Joseph P. Noonan, and Sos Agaian. Binary data encryption using the sudoku block cipher. In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 3915–3921, 2010.

[Woe03] Gerhard J. Woeginger. *Exact Algorithms for NP-Hard Problems: A Survey*, pages 185–207. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[WZN+10] Yue Wu, Yicong Zhou, J. Noonan, K. Panetta, and S. Agaian. Image encryption using the sudoku matrix. In *Defense + Commercial Sensing*, 2010.

[XHHLB08] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 06 2008.

[YS03] T. Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 86-A:1052–1060, 2003.

Appendix A

Features

The features presented in the Table 1 are total features that have been utilised in the entire experimentation process of the thesis. A total of 70 features has been utilised for the entire experiments of Algorithm Selection and Run-time prediction but a maximum of only 35 were selected by the feature selection method. The features have been presented as in the data sets for Algorithm Selection and Run-time Prediction. By presenting this table of features, feature importance for each data set will be presented in the case of Algorithm Selection and Run-time Prediction, respectively.

Algorithm Selection Feature Selection

COH37

Feature importance for COH37, the feature scores has been depicted in Figure 1. The missing values from the puzzle show more important, like missing nodes, missing edges, etc.

COS36

Feature importance for COS37, the feature scores has been depicted in Figure 2.

COHS37

Feature importance for COS37, the feature scores has been depicted in Figure 3.

HardSet

While the above three data sets for Algorithm Selection had almost similar data importance. The HardSet features were slightly different. The feature scores has been depicted in Figure 4. Although the missing values from the Sudoku puzzle depicted similar importance.

0	:	columncomplete			
1	:	columnempty			
2	:	highestcolumnsum	37	:	minratiocolumn
3	:	highestmaxcolumn	38	:	minratiorow
4	:	highestmaxrow	39	:	minratiosubgrid
5	:	highestmaxsubgrid	40	:	missingdomain
6	:	highestoccpuzzle	41	:	missingnodes
7	:	highestrowsum	42	:	numdomain
8	:	highestsubgridsum	43	:	numnodes
9	:	iqrpuzzle	44	:	orderpuzzle
10	:	largestcolumn	45	:	percentpuzzle
11	:	largestrow	46	:	puzzlediagonal
12	:	largestsubgrid	47	:	rangemaxclique
13	:	leastcolumnsum	48	:	rangeminmaxsubgrid
14	:	leastconditionbin	49	:	rangemintotalsubgrid
15	:	leastconditionforsolution	50	:	rowcomplete
16	:	leasthighestcolumn	51	:	rowempty
17	:	leasthighestrow	52	:	sdcolumnadd
18	:	leasthighestsubgrid	53	:	sdrowadd
19	:	leastmaxcolumn	54	:	sdsubgridadd
20	:	leastmaxrow	55	:	sdsizerow
21	:	leastmaxsubgrid	56	:	sdsizesubgrid
22	:	leastrowsum	57	:	domaincover
23	:	leastsubgridsum	58	:	sizediagonal
24	:	lowestoccpuzzle	59	:	sizepuzzle
25	:	maxclique	60	:	smallestcolumn
26	:	maxdegree	61	:	smallestrow
27	:	maxnumedges	62	:	smallestsubgrid
28	:	maxoccurence	63	:	subgridscomplete
29	:	maxmissingnumedges	64	:	subgridsempty
30	:	maxratiorow	65	:	sumrangepuzzle
31	:	meanadd	66	:	sumratiopuzzle
32	:	meanpuzzle	67	:	totaledges
33	:	minoccurence	68	:	flatintvarschuf
34	:	minmaxcolumn	69	:	flatintconstraintschuf
35	:	minmaxrow			
36	:	minmaxsubgrid			

Table 1: Features: The features as computed for the data sets.

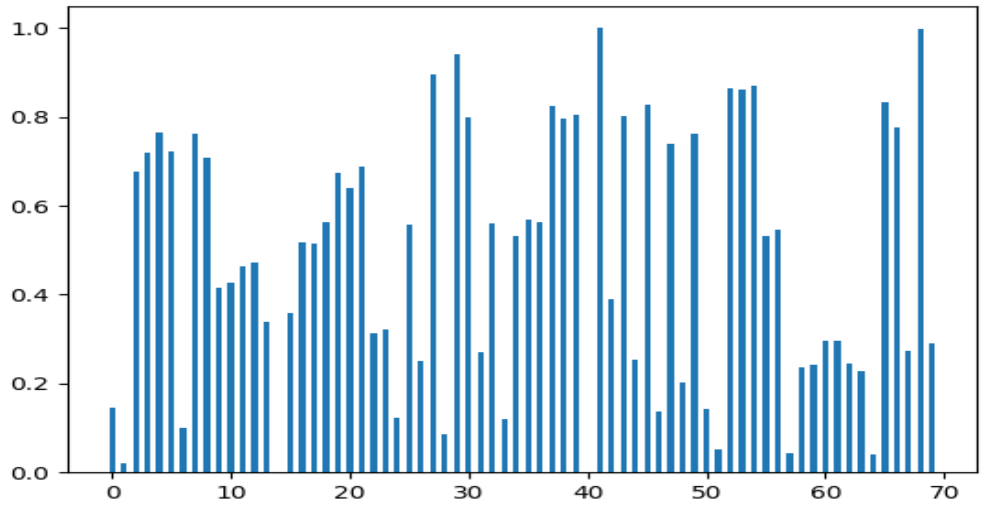


Figure 1: Feature Importance according to the table 1 for COH37 data set.

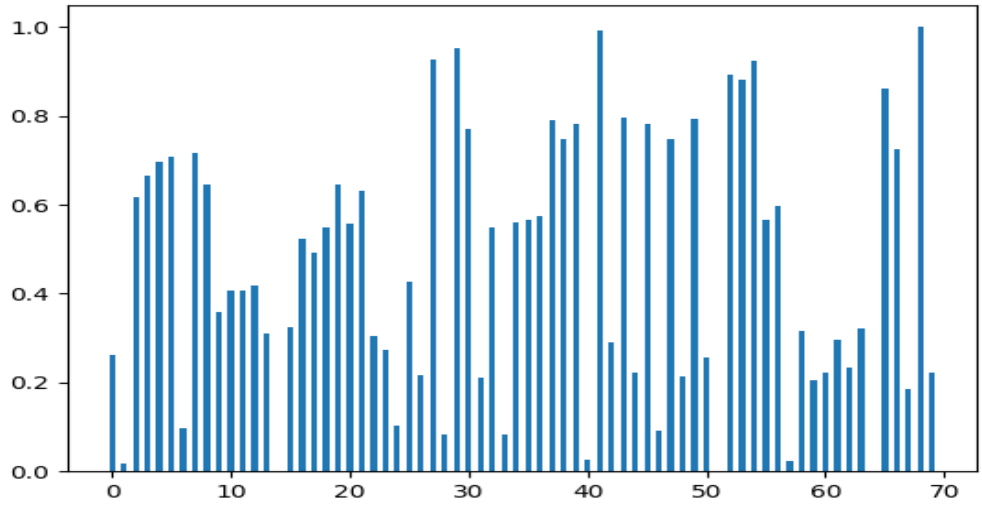


Figure 2: Feature Importance according to the table 1 for COS36 data set.

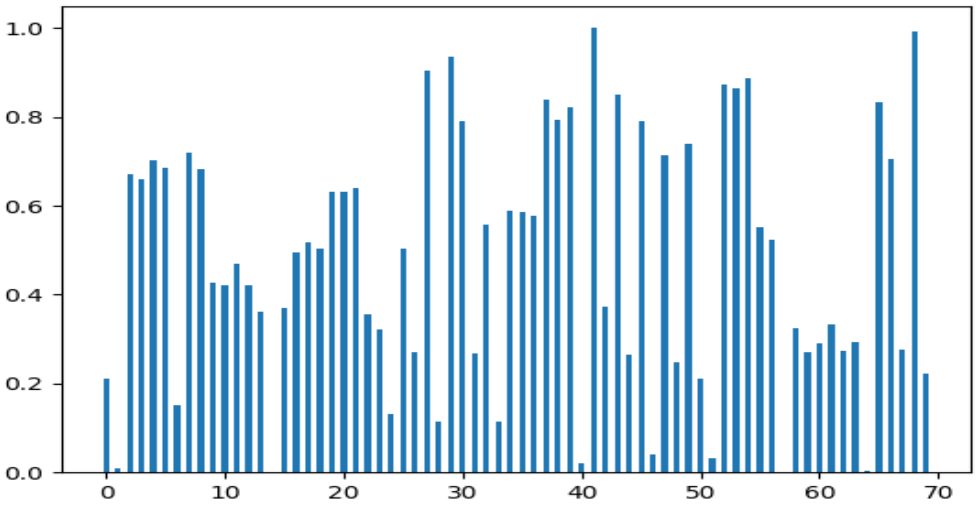


Figure 3: Feature Importance according to the table 1 for COHS37 data set.

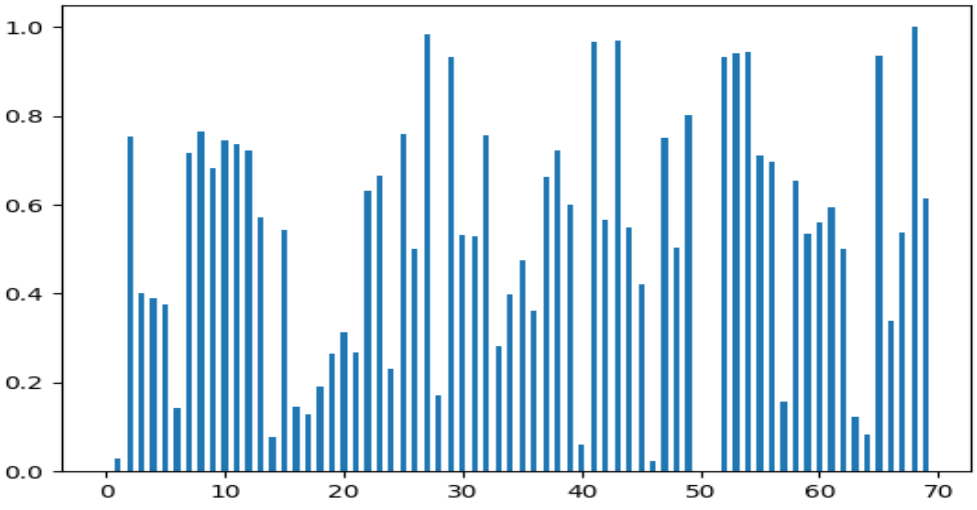


Figure 4: Feature Importance according to the table 1 for HardSet data set.

Appendix B

Run-time Prediction Feature Selection

CHUF39

Feature importance for CHUF39 shows that order and size of the puzzle does have some importance for the regression of this data. The most important feature is flattening time, as the addition of the flattening time was done to the solving time. The feature importance has been depicted in Figure 5.

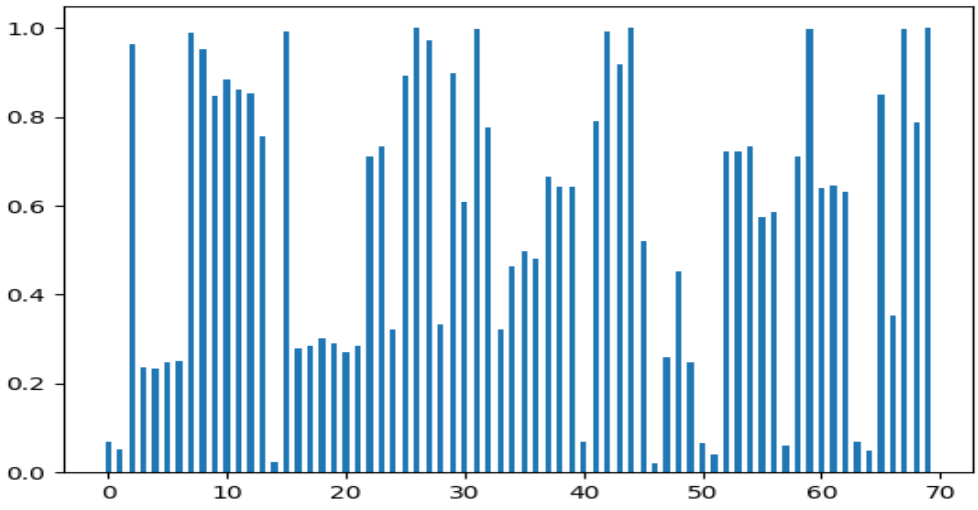


Figure 5: Feature Importance according to the table 1 for CHUF39 data set.

ORTL39

The most important feature for ORTL39 using regression using SKB was number of nodes on the Sudoku puzzle. The feature importance has been depicted in Figure 6.

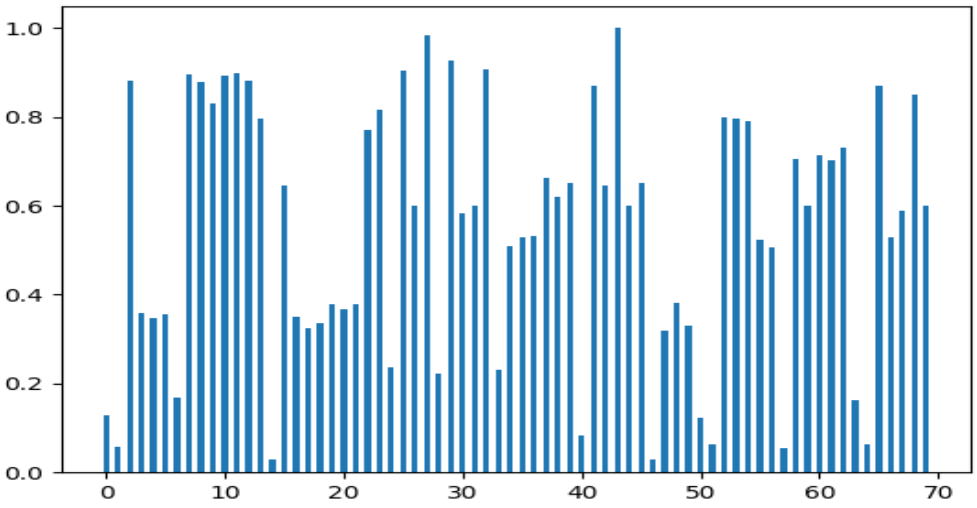


Figure 6: Feature Importance according to the table 1 for ORTL39 data set.

SIMA39

The most important feature for SIMA39 with regression using SKB was the flattening variables of chuffed. The feature importance has been depicted in Figure 7.

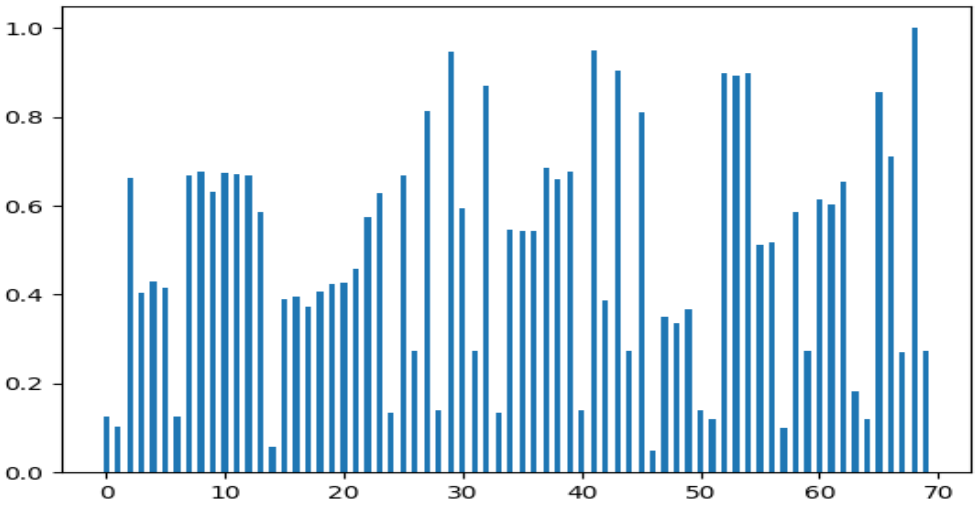


Figure 7: Feature Importance according to the table 1 for SIMA39 data set.

HYILS39

Similar to ORTL39, HYILS39's most important feature was number of nodes on the Sudoku puzzle. The feature importance has been depicted in Figure 8.

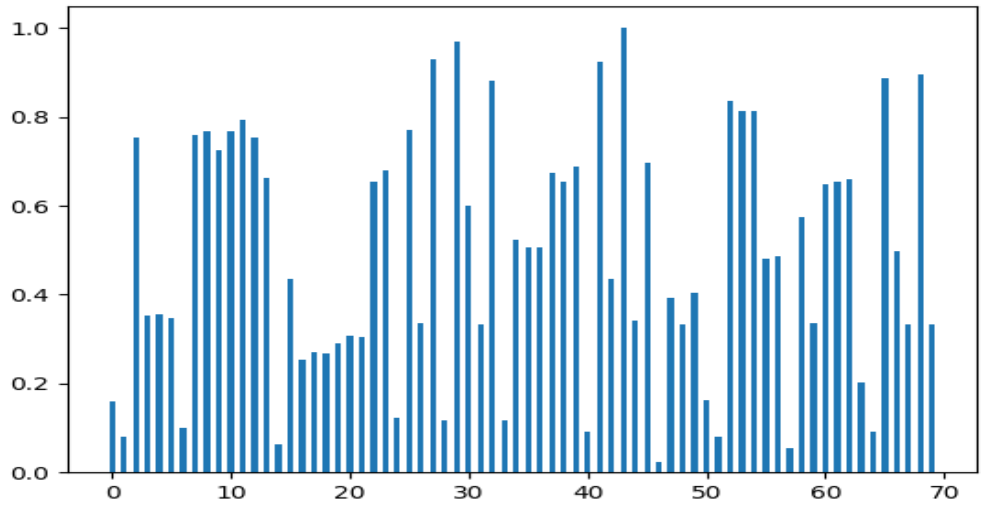


Figure 8: Feature Importance according to the table 1 for HYILS39 data set.

Appendix C

Algorithm Selection for Easy Data Set

This section we will present the result of the easy empirically easy data set. The empirical hardness of the Sudoku instances are presented in Table 4.1 and Figure 4.7. The accuracy achieved using RF was 94%. The RF machine learning technique out performs the best performing algorithm, i.e. ORTL. This is depicted clearly by Figure 9.

	HYILS	ORTL	SIMA
HYILS	4	1	7
ORTL	0	140	2
SIMA	2	0	34

Table 2: Confusion Matrix using RF with k-value 15 and number of trees 100 for the data set EasySet.

Figure 9 shows that our RF classifier performed much better than any other solver. The RF classifier was able to correctly classify many instances showing the robust nature of our algorithm selection approach.

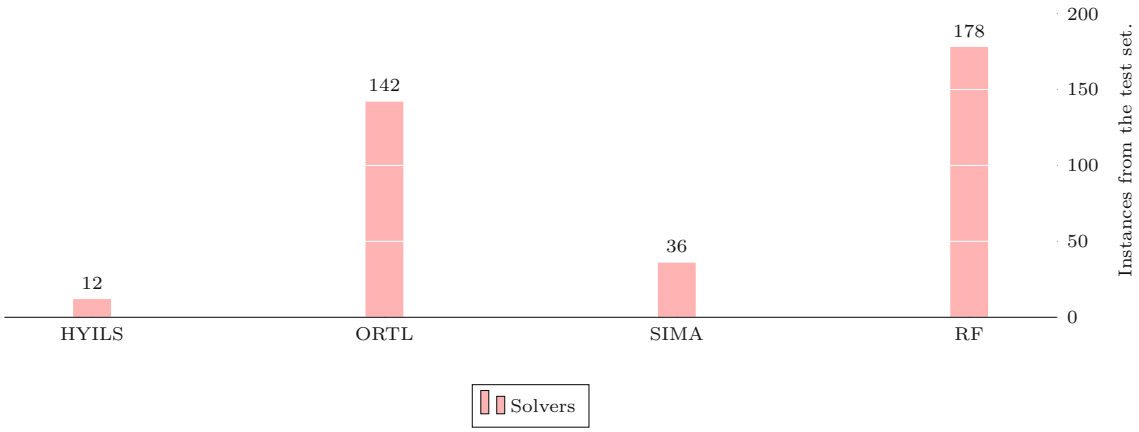


Figure 9: Instances Solved most efficiently w.r.t. the Solvers and Algorithm Selection approach using RF ($\mathcal{F}_{25}$) for the test set of Easy data set.