*Division of Computing Science and Mathematics*

*Faculty of Natural Sciences*

*University of Stirling*

Project Title

Predicting SAT Runtime

Author Name

Shardul Bajirao Mole

**Dissertation submitted in partial fulfilment for the degree of   
Master of Science in Big Data**

**September 2022**

Abstract

Bollean satisfiability problem has become a very crucial and unsolvable problem from a very long time. To understand the problem in a better way, we used a SAT solver to predict the time required to get the desired output. So while passing the SAT instances through the SAT solver we noticed that the time taken by each instance was not same. Some would generate output within seconds whereas some took 8 to 10 minutes(some take time more than that also but for our convenience we limited the time frame to 10 minutes). It made difficult to predict the exact time taken by each instance to get the output. Thus, this problem needed a solution.

So, to find a solution for this, we took some SAT instances and processed them through the SAT solver and found out the average time taken by those instances which would help us predict the runtime of the new SAT instances. This made predicting the time taken by new SAT instances to run and generate the output easy by comparing the existing training data and the newly formed test data. From this we need to build a new model which would help us to predict the runtime of new SAT instances without passing them through the SAT solver. This would help us to get a brief knowledge of which instances would take more time and which would take less time when passed them through the SAT solver.

Attestation

I understand the nature of plagiarism, and I am aware of the University’s policy on this.

I certify that this dissertation reports original work by me during my University project.

**Signature** *Shardul Bajirao Mole* **Date:- 07/05/2022**

Acknowledgements

I would like to express sincere thanks to Dr. Patrick Maier for his valuable guidance throughout my dissertation project. I got instant help from him whenever I got any queries/difficulties in various phases of completion of this project.

I would also like to express my gratitude to the Computing Science and Mathematics Department for giving me an opportunity to work on the project of Predicting SAT Runtime.

Table of Contents

Abstract i

Attestation ii

Acknowledgements iii

Table of Contents iv

Introduction 1

Background and Context 4

Related Work 5

miniSAT solver 8

The Constraint Satisfication Problem 9

Boolean Satisfiability Problem(SAT) 10

Machine Learning and Constrain Solving 11

Data Cleaning and Processing 12

PyCaret 14

Statistical measures in Modelling 16

Linear Regression……….…………………………………………………………………………………………..18

Random Forest…….…………………………………………………………………………………………………19

Decision Tree Regressor…………….……………………………………………………………………………19

K-Nearest Neighbor (k-NN)…….……………………………………………………………………………….20

Conclusion 22

Future Work 23

References 24

Installation guide 29

# Introduction

Boolean Satisfiability problem is nothing but a problem which helps to determine whether a Boolean formula is satisfiable or unsatisfiable. The problem is satisfiable if we assign a Boolean variables in such a way that the formula turns out to be TRUE. SAT solving problem has been a issue since many years. There have been lots of researches and practices who have tried solving this problem for past decades. There have been various types of solvers created by various researchers to overcome this issue. There were some modern solvers as well which are robust in nature but still they give different runtimes when run repeatedly on a solver. This report helps to check whether there is any model which can predict the runtime of a solver for a given instance. To do so we need to take into consider various models and then compare their output with each other so that we can come to a conclusion as to which model is preforming best for the given SAT instances.

The scope of the project is to built a model that would help us to predict the expected time to run a SAT instance in a better way. For this we used the machine learning tools and compared various models to understand which model would predict the most accurate runtime of an instance. This helped us to better understand that is there any model that can predict most accurate runtime for a given instance.

To perform this task, we had to set up a separate machine that can handle such huge tasks. The operating system must be capable to run big files(instances) in a very smooth manner. Time required should also be less and machine should be monitored for overheating. We needed to find such a operating system that would help us work efficiently and be user friendly. It should be capable of doing multitasking, help in data cleaning and processing and the files should be stored in a safe way and easily accessible whenever needed. There have been cases where researchers were running huge instances that would take hours to run and generate output from the SAT solver and sometimes the machine would crash. This happened because the machines were weak and didn’t had the capacity to load such huge data files. So this also was taken to be in consideration before choosing the operating system.

Now coming to data cleaning, the data which we got from the websites ad SAT competitions was of very mixed kind. Some instances were huge that took hours and days to run while some were very small and took few seconds to run. Actually, it was a bit difficult to illustrate as to which instances would take less time and which would take more time just by looking at the file. The data were mostly stored in zip folders in an unordered manner and thus sorting the files which were to be used became difficult. To solve this issue we needed to download all the zip folders onto our machine and then unzip in our folder in which the files were to be stored.

At its most basic level, training a portfolio entails collecting each solver's runtime performance against a set of benchmarks. Machine learning algorithms are then used to select the best solvers depending on instance features. Existing research in this area has only looked at a single data point of the solver time for each instance, assuming that the actions are consistent over multiple solver runs. However, as we will show, a solver's performance might vary by orders of magnitude when performed repeatedly. [Gomes et al., 2000] discovered that this behaviour is associated to heavy-tailed distributions. Estimating a solver runtime aims at developing an empirical difficulty model that can be used to estimate a solver's performance based on instance-specific characteristics.

Thus, the contributions of this paper are as follows:

– we show that in practise, large runtime differences occur, which might impact the result of empirical analyses like the annual SAT competitions,

– we show that in areas like solvers runtime prediction, portfolios, including automated configuration, runtime variation must be taken into account,

– around 100 industrial SAT cases spending almost 6 hours of CPU time are used in an empirical evaluation, and

– we present a statistical approach that may be used to compare solver performance from the perspective of runtime distribution.

The Boolean Satisfiability problem is a NP-complete problem. This fact was first stated by the Cook-Levin theorem considering the complexity theory of computation. Does a Boolean expression withinside the conjunctive everyday shape assign a Boolean variable in order that the expression evaluates to real while expressed as a Boolean expression? These are various traditional NP-complete troubles due to the fact the Boolean SAT solver can not be scaled withinside the feel of NP-complete. The SAT solver is typically used to resolve conditions that comprise hundreds of thousands of Boolean variables. By lowering the NP trouble to a Boolean satisfiability trouble, many practitioners use the cutting-edge SAT solver to locate the answer [11, 12, 19]. As a result, SAT answers have turn out to be an critical characteristic in diverse industries. Effective propagation, warfare analysis, preprocessing / processing, and restart are a number of the important thing ideas of the SAT solver with Competitive-Driven Phrase Learning (CDCL) evolved through the SAT community [18]. Running the CDCL-SAT solver could also be notion of as a tree of nodes with output edges categorized actual and false, very similar to a standard backtracking seek (showing variable cost assignments). The solver is restarted frequently. That is, discard the modern-day seek tree and start over (even though all discovered terms and variable hobby are preserved). Rebooting the SAT solver can also appear inefficient, however it is empirically quicker than a solver that does not reboot.

There are some suggestions by the academics which state that the distribution in the runtimes must not be constant. This helps in keeping the ambiguity to the restart and performance of the solver. These ideas were, however, not justifiable as to why the restarts are important in CDCL SAT solvers. Modern SAT solvers are using CDCL(Conflict driven clause learning) instead of the simple and easy DPLL backtracking search procedure. The main motto of CDCL is 1. Helps in deciding truth value to an unassigned variable 2. Perform unit propagation 3. when a conflict (a falsified clause) is obtained, analyse the reason for the conflict and learn a new clause that forbids this and possibly multiple similar conflict situations from occurring in the future 4. Back jump, possibly over several irrelevant decisions, based on the conflict analysis.

The improvement of the thought of literal block distance (LBD) through Gilles Audemard and Laurent Simon [7] in 2009 became the contemporary development withinside the contemporary SAT solver. The described distance is blocked through 3 literal (LBD) S.t. Literals are partitioned with admire to the selection stage given clause C and therefore the literal walls into n subsets in step with the mapping at hand. The LBD for C is precisely to make a decision the fee of a discovered sentence, LBD has grow to be an excellent method for comparing headwords and selecting headwords that want to be retained instead of older activity-primarily based totally measurements. The more the LBD decreases, the upper the headword becomes.

The greater great terms the solver seems at, the far more likely it is to get a answer quickly. On the other hand, restarting too regularly could also be steeply-priced thanks to the very fact the hunt tree is consistently structured. Managing restart frequency is critical to enhance LBD at an equivalent time as avoiding immoderate overhead. Frequent restarts can also cause the solver to overheat, which may lead in irregular shutdowns, data loss, machine to act irregular, and even cause long term damage. Due to these reasons we skip the LBD in our further modelling.

MiniSAT can deal with SAT problems which is perhaps appreciably more annoying and sophisticated than those it faced within the course of education via running more message-passing iterations. MiniSAT uses just some dozen iterations within the course of education, but hundreds, if not thousands, of iterations within the course of checking out to seek out answers to more complicated problems. instead of a traditional classifier, the going to know manner superior a fashion which will be used indefinitely to settle on out answers to situations of various complexity.

MiniSAT is considered as a simple SAT solver as it also provides a C++ interface which uses 3 vocabulary types:

Minisat::Solver - Implementation of the core solver and its algorithms.

Minisat::Var - Representation of a variable.

Minisat::Lit - Representation of a concrete (positive or negative) literal of a variable.

One utility type is also used in MiniSat's interface: Minisat::vecT> is a container for passing clauses to the solver, analogous to std::vector.

The SAT solving algorithms are to be developed simultaneously and to do so there are two different approaches. Using the portfolio strategy, one SAT instance can be solved using many SAT solvers. SAT solvers communicate information when solving SAT problems (usually conflict clauses). The original SAT instance is partitioned into numerous independent instances using a partitioning method. It is natural to use parallel or distributed computing to solve examples from the acquired family of problems. For a single SAT instance, many different partitionings are feasible. The concern here is how to evaluate partitioning and compare it to other partitioning. From a practical point of view, it can be rephrased as follows. How can you quickly find a reasonably good partition? The answers to these questions are in our report. The next section provides more details.

The Propositional Satisfiability Problem (SAT) is a logic and computer science problem that evaluates whether an interpretation of a Boolean expression can satisfy a formulation. It is generally believed that a single solution does not adequately address all SAT issues. Therefore, it is difficult to predict the solver execution time. In some situations it may take a moment, in another it may take hours or even days. The goal is to develop a model that analyzes the data provided by the SAT solver during the solver process to estimate the solver's execution time (for example, the number of conflicts, the number of restarts, the memory used Amount etc.).

## 

## Background and Context

Prediction of the runtime of a solver has always been a very fascinating and interesting thing to the researchers from decades. There have been made several attempts by famous scientists to create a machine learning model that would help in prediction of the runtime of a SAT solver. There were numerous comparisons of general machine learning methods which were applied to the runtime prediction e.g. (49). Though, these methods just helped in tuning the mathematical solvers in a better way (49). However, more studies are being made in tuning the existing frameworks to make the solver more advanced for a specific problem which would lead to tuning of mathematical solvers and feature selection (50). There are Machine learning problems where the data is produced by running the algorithm itself and this helps us to choose the method through which the data is to be produced. Such kind of method are used to active learning and are used in the scenarios where there is inadequate or unbalanced data (51).

The strategy of random data generation is used by most of the applications that help in runtime prediction. But the recent studies have indicated that they have found more success when an approach of novelty search was executed (52). Though this study was done by using just the Neural networks that were artificial in nature. We will be extending this study by the doing the comparison of different data generation techniques that will be executed on different machine learning algorithms. The prediction of runtime that has good output gives chance to the user to choose the algorithm parameters that the user expects. This is mostly related to a quicker runtime while taking into consideration the quality of the solution(53). In the following report, our main goal is to increase the accuracy of prediction of runtime of a SAT solver by simultaneously making the use of machine learning methods along with the strategies used for the data generation.

**Related Work**

The use of gadget studying to supply example-precise predictions approximately problem attributes has piqued the hobby of the combinatorial optimization network withinside the ultimate decades. Many authors had been inquisitive about forecasting the time it might take a solver to run (54). These techniques have proven to be all of sudden powerful throughout a huge variety of problems, solvers, and example distributions. The reality that exponentially strolling algorithms can be foreseen withinside the worst state of affairs is counterintuitive to maximum people(55) proposed functions are observed with the aid of using the maximum considerably utilised methods to studying to cause approximately SAT problems.

Researchers have found these heuristics (such as percentages of phrases and variables that look positive or negative), manageable subclasses (such as those close to the horn formula), and other measurements of problem complexity (such as LP mitigation statistics). Created 84 predictors of solver performance using SAT issues and time-limited execution progress of SAT solver statistics). These features manage random forest models and are combined with many machine learning models of random forest models studied by(56). (Example: LP mitigation of problem size or near cube). In 2014, Hutter et al.(57) Created an algorithm portfolio by manually creating attributes to estimate the execution time for a particular case.

To set up a operating system there are numerous available operating system in the market. Be it the renowned ones like Windows, Linux, Unix which support almost all the required technologies for our modelling. The operating system should be based on various important constraints. Stability and robustness is one of the important factors to consider before selecting an operating system. The operating system required for modelling should be stable as the tasks performed on it are very complex and huge. So if the machine is stable it will easily handle such datasets and working of the machine will be better. Similarly, the machine must be robust and strong so that when huge datasets are made to be run in the solver on the machine, the machine will work smoothly and effortlessly. Memory management plays a vital role in making the machine efficient for doing multi-tasking.

Taking into consideration of all the above things the most convenient and easily available operating system is ubuntu. One of the foremost popular Linux distributions is Ubuntu. Created by Mark Shuttleworth of Canonical Lab. it's also the foremost well-known open source technology, meaning that each one its features and applications are available for free of charge. And it's an indisputable fact that if an application is free, its popularity will grow automatically. Because Ubuntu is open source, it releases new updates almost twice a year, and future Support (LTS) releases security fixes that are updated every two years. Core, desktop, and server are the three main distribution categories. The core version is primarily aimed toward IoT and robotics developers. The desktop version is meant for everyday office use and programming applications. The server version is designed solely for the client-server architecture and is primarily intended for industrial applications.

It's always beneficial to work on a virtual machine instead on the main machine as virtual machine provides the flexibility to work remotely and smoothly without any efforts on the main machine. The data can be backed up easily onto the main machine which would help to recover data incase of the virtual machine getting crashed. There are some more benefits of virtual machine which made us use them instead of loading a operating system on our main machine. Prior to virtualization, organizations spent most of their IT budget buying physical servers to host their applications. As virtualization became more important, companies began investing in more powerful hardware that could support more virtual machines. This is significantly cheaper than buying additional hardware. In addition, you can use virtual machines to extend the life of your old software. Physical servers damaged by a disaster can take hours or days to exchange. Virtualized environments, on the opposite hand, are often duplicated or duplicated at any time and may be restored in minutes. IT teams are more productive because they only got to maintain the host machine. Hardware maintenance is harder in traditional data centers that haven't yet migrated to virtualized environments. The impact of maintenance activities on the assembly environment is negligible. When doing maintenance work, downtime is usually not required. additionally, virtual machines simplify the testing and development of applications and websites. VMWare machine was installed and used to perform modelling. VMWare are known for backup and data protection methods that are more robust, lowering the chance of data loss. It provides more applications then other virtual machines and the downtime of the machine is generally less. Incase of data loss, the data recovery time is very less.

Work related to the bottom al. [22] is a more popular alternative to an official SAT solver, both in research and in applications. The holistic approach is gaining popularity due to its excellent performance in real-world applications. Furthermore, its ability to determine the satisfaction level makes it suitable for solving simple problems commonly encountered in many modern applications. In recent years, global approaches such as the DPLL baseline finding algorithm have become central to the study of SAT solvers. Many of the goals of these studies are to improve branching algorithms and heuristics. On-demand learning and non-temporal sequence recovery have both made great strides in this area [27] and are particularly well-suited to the orderly nature of real-world applications. Clause learning is the process of analysing and storing knowledge about conflicts that arise in SAT solvers in order to avoid them in the future.

Simultaneous literal clauses are often used to learn propositions. On the other hand, non-chronological backtracking refers to conflict resolution through conflict analysis rather than backtracking to the final decision-making level when the conflict arises. Another major achievement of the SAT solver was the invention of Chaff, the SAT solver introduced in 2001. This solution had a huge impact on the region at the time due to its excellent performance. This performance is supported by the variable state independent attenuation summing (VSIDS) method, first developed by Chaff. Essentially, the VSIDS heuristic assigns each letter in the initial search algorithm an operation score that increases by division as statements containing that word are added during the search.

A second study was conducted to handle the difficulty of inadequate early branching. this will take a substantial amount of your time to resolve in some situations. To counter this problem. it had been first proposed this idea [16] and it's widely accepted to restart the solver in an exceedingly random way. Random restart discards the variables assigned to the present search tree, but preserves the clauses within the clause creation in order that the solver can inspect the new search tree without raising the identical error.

One more study was conducted to simplify the SAT instances by distributing them into different classes based on the time taken by them to run in seconds. This classification model was created considering the time taken to generate the output should be within the similar time frame. The time frames were classified on the 50 seconds gaps. For the first time frame was 0 to 50 seconds, 50 to 100 seconds, 100 to 150 seconds and so on. In the beginning, this model seemed to be efficient as all the instances were easily able to fit in this pattern and there were no instances left. Machine learning algorithms are commonly used to recognize and classify things. This is called classification and can divide large amounts of data into individual values ​​such as 0/1, true / false, or a predefined output label class. The process of recognizing objects and thoughts, interpreting them, and classifying them into predetermined groups (often called "subpopulations") is called classification. Machine learning programs use a variety of algorithms to use these pre-categorized training datasets to place future datasets into appropriate and relevant categories. The machine learning classification algorithm uses input training data to predict whether the following data falls into one of the specified categories: The classification in our situation is based on the time it takes for the machine to produce output and the sort of solver execution time. In other words, classification is a kind of "pattern recognition". In this case, the classification algorithm applied to the training data will recognize the same pattern in future datasets (the same time the instance is processed by the solver). Dive deeper into the classification algorithm and how the Boolean satisfiability problem solver performs tasks such as sentiment analysis used to classify unstructured text based on opinion polarity (positive, negative, neutral, etc.).

Though classification model seemed to be useful and efficient there were some drawbacks as well. The classification only allows in achieving moderate accuracy. For an instance we don’t get to know the exact runtime taken by the solver to generate the output but instead we get the time frame. This makes the classification model moderately accurate with the accuracy percentage of 80-90%. The data required which we had over here was a mixture of structured and semi structured type. The classification model requires high quality data in order to produce better output. High quality data makes the classification easy specially in artificial neural network and KNN modelling. To build a classification model, large datasets are required. There should be 100-1000 of labelled training events for each category. Moreover, the total number of datasets which we have taken while modelling is 100. That’s very less considering the modelling requirement of a classification model. Some of the datasets of the Boolean satisfiability problem are very huge which makes it computationally intensive to build the model. The machine would have struggled to process these huge datasets in a classification model. Also, the energy taken to perform these tasks would be more. Considering all of these terms it was difficult to build a classification model that would help in the prediction of the runtime of a SAT solver.

From our references [11], [12], [13], and [14] all focus on QoS prediction. The author of [11] uses the Markov Arrival Process (MAP) and the MAP / MAP / 1 queuing model to predict server performance. The smart grid case study uses the M / M / 1 queuing model, but it is unconstrained and you can change the QoS prediction approach as needed. The prediction-based resource evaluation proposed in [12] uses neural networks and linear regression to predict future resource needs. Based on numerical estimation, the regression model in [14] is also used to predict run-time service level goals (SLOs). PREvent is a regression classifier that predicts injuries according to [13]. However, no information about its performance is provided. [15, 16, 17] addresses QoS requirements by optimizing the QoS requirements at run time. [15] uses linear programming to set up a runtime adaptive framework that meets the QoS requirements of a service-oriented system. [16] and [17] represent the challenges of a multipurpose rating of for the development of QoS adaptive provider systems. [18] provides a collaborative approach to predicting the performance of cloud components based on user feedback. While this technique is suitable for user-oriented QoS indicators, it is not suitable for most business-oriented QoS situations. QoS prediction using the model checking approach is described in [19] and [20]. [19] describes a technique called ATOP (prism model from an abstract description of a service configuration) that generates a probabilistic model from an abstract description of the service configuration and feeds it to the evaluation phase (activity diagram). This theory doesn’t apply to the modelling done to predict runtime of a SAT solver. In contrast to our solution, this is a way to evaluate your system during the design phase.

Similar to our work, [20] proposes a two-step method, including monitoring and prediction, to monitor the reliability of a configured web service that exhibits unpredictable behaviour at run time. This method involves probabilistic model validation, but primarily deals with reliability assessments using DTMC-based Markov models. On the other hand, it provides a universal CTMC probabilistic model of parameterized counters for both states and transitions, creating a model that can be modified at run time. So the model which we have built can be easily adjustable during the runtime and we can make the necessary changes required on the go without waiting for the process to end.

The Boolean satisfiability problem and the constraint satisfaction problem have existed for centuries. However, the terminology is relatively new. Boolean satisfiability is based on logical satisfiability. All propositional expressions contain an instance of the Boolean Satisfiability Problem (SAT). As a result, the terms "satisfaction" and "satisfaction" are sometimes used interchangeably. Contrary to popular belief, artificial intelligence is needed to meet the constraints. You can solve various problems. Chart colouring, queens, and schedules are examples of problems that can be treated as constraints. Constraint programming is used to solve these problems. Both constraint and Boolean satisfiability have emerged as new areas of computer science, but they are treated in completely different ways. Any propositional expression can be considered as a constraint satisfaction problem (CSP), so SAT can be considered as a specific example of CSP.

Some studies have shown that there is no universally ideal way to deal with combination problems such as configuration problems. Heuristics are currently used to search for heuristics and try to find the best solution for each problem instance to solve these problems. The algorithm selection approach works well in many situations, and related approaches for predicting solver performance have become the subject of more general research. Machine learning algorithms that can learn domain-specific search heuristics can help reduce search effort and improve predictive quality. In addition, machine learning can be used to predict whether a particular problem can be solved and reduce the overall search effort.

**miniSAT Solver**

MiniSat simplifies the process of solving SAT problems by using two literal monitoring techniques for fast BCP and phrase learning based on competitive analysis. It acted as a backend and standalone solver for the SatElite pre processor. MiniSat has undergone some minor improvements over the original Chaff. Incremental SAT interface and user-defined Boolean constraints are supported. None of these advances apply to SAT contests that use only non-incremental SAT questions in CNF. However, these are essential if the MiniSat is part of a larger solution.

The order is flexible. MiniSat uses a modified version of VSIDS that reduces the activity that fluctuates with each conflict by 5%. The original VSIDS variable is exhausted by about 50 to 1000 conflicts. The VSIDS extension responds quickly to changes in branch variables and does not branch on older benchmark-validated variables. The heuristic's main goal is to increase the activity of all variables that appear in the clause used in the conflict analysis, not just the variables that appear in the last conflict clause (as in the first version of MiniSat). This version of the solver uses the heap to always organize variables per activity.

True or false sentence. By storing the propagated literals, the binary clause supports direct propagation in the watcher list. Our test wasn't definitive, but this approach was better than storing all the binary clauses individually. All binary clauses must be passed this way before or after BCP passes a larger clause. This distorts the implication graph and displays the conflicting phrases more closely.

The miniSAT installation process included downloading the miniSAT tar folder from the free source miniSAT page. After which the application was extracted from the tar folder and then installed on the virtual machine of ubuntu. On running the SAT instances through the miniSAT it provided an output which was machine readable. The structures were in a unstructured binary format. This made it difficult to understand and processes the data further for modelling. To overcome this issue, we tweaked the miniSAT code in such a way that the output that we got became human readable. Basically we changed the miniSAT code and then processed the instances through it. The new outputs that were generated had all the detailing of each task performed which consisted of “Event”, “Time”, “AdjustCount”, “Restarts”, “Conflicts”, “Decisions”, “LEARNT\_Limit”, “LEARNT\_Clauses”, “LEARNT\_Lit/Cl”, “Progress”. It was easily modifiable and thus more convenient to process while modelling.

1. Event:- It defined the name of the event which is being performed. In our case there are 2 events Restart and LearnMore.
2. Time:- Defines the time taken by that instances to run.
3. AdjustCount:- Total number of counts taken by the machine to adjust itself so as to proceed to the next instances.
4. Restarts:- Number of time the machine restarts in order to process the whole instance.
5. Conflicts:- Total number of problems faced by the machine while running the particular instance.
6. Decisions:- Total count required to proceed to the machine to make any conclusion.

## 

## The Constraint satisfaction problems (CSPs)

The number of parameters that can be allocated to a collection of variables is limited when a restriction is defined on that subset (APT, 2003). The constraint satisfaction problem (CSP) is a problem in which a combination of variables and values ​​is specified by meaningful variables (Apt, 2003). The assignment must instantiate each variable to meet all the criteria in question. The CSP is also responsible for financial resolution recommendations (57), communication switch configuration updates (57), and system reconfigurations (57). The following is the definition of the constraint satisfaction problem (CSP) as the basis for the next topic (58).

The CSP and its solution are defined. V, D, and C are all variables, domain definitions, and constraints, and are all defined as triples (V, D, C). Resolve the CSP by assigning the variable v1 to valv1, vn to valvn, the variable value from vj to valvj, and to the domain (vj) and consistency (C \* A). Preliminary inspection (analyzing how changing the value of one variable affects the value of another variable) and several different ways to check the integrity properties (all single phases defined in one variable) Constraints are mutually exclusive) Backtrack search and (needs consistency) and circular consistency (for example, if a binary link can connect two variables, for example x1 and x2, the corresponding domain values ​​are domains (for example) to each other. Located in x1). Must be consistent. Commonly used for solutions used in CSP Apt (2003) is a problem solver to meet constraints.

Constraint satisfaction problem tends to solve those problems those have some particular pattern or similarity in them. It basically analysis that pattern and then tries to fit the new entities in that pattern to make it satisfiable. In our modelling ‘time’ being the target entity, it can be compared with the runtime that we get from the training data and thus CSP will make it easy to predict the runtime of the solver.

**The Boolean satisfiability problems (SAT)**

Boolean Satisfiability (SAT) In CSP, Boolean expressions are represented as Boolean expressions and constraint expressions are represented as Boolean expressions (58). CSP is closely related to SAT issues and can be rewritten in SAT format (59). In SAT problems, the variables are the same as in CSP problems, and each variable represents a domain value. In SAT format, the domain (x1) is V = [x1x2].

The SAT problem has been researched for decades and plays a significant part in the history of computational complexity. When classifying the efficiency of algorithms, computer scientists created the NP class for complex decision problems3,4. Intractable issues are those that are "hard" in the sense that no existing algorithm can determine whether a solution exists in the worst-case situation in polynomial time. The SAT issue was the first to be demonstrated to belong to the category of NP-complete problems3, suggesting that any NP-complete decision problem may be reduced to a polynomial-time SAT problem. A specific example of the problem structure is an efficient exponential time algorithm, but there is no known polynomial time strategy for solving NP-complete problems. There is a "broad opinion" 4 that the development of polynomial time algorithms is not possible, but this concept does not preclude the development of polynomial time continuous physics systems.

From scientific research (graph theory, algebra and numerical theory, arithmetic programming) to industrial applicability (graph theory, algebra and numerical theory, mathematical programming) (network design, data storage and retrieval, program optimization) Up to, there are hundreds of other NPcomplete issues. If the polynomial algorithm can solve all NPcomplete class problems, it can effectively solve all NPcomplete problems. 3SAT is an NP-complete problem that is a subset of SAT4. Randomly generated 3SAT examples are known to be difficult for many solution strategies due to the lack of a recognizable problem structure.

As the results of the SAT Contest show, the performance of the SAT solver varies dramatically between the SAT categories (random, custom, industry) (random, craftsman, industry). One theory is that the SAT solver uses the basic structure of a SAT instance. The structure of the SAT has been attempted to be defined using structural measurements such phase transition, backbone, backdoor, tiny world, scale-free, tree width, centrality, community, self-similarity, and entropy. However, empirical evidence of SAT structural measurements is provided only for the selected SAT category. Moreover, the evidence is only theoretical. In addition, the effect of structural measures on the behaviour of the SAT solver has not been fully investigated. Thus, various researches and experiments have been conducted on the Boolean satisfiability problem from time to time, but a proper solution is yet to be discovered.

**Machine learning and constraint solving**

Machine learning focuses on using training data (past events) to create accurate models that can be generalized (hopefully) to specific data. Machine learning serves two basic tasks. Test cases are categorized based on their potential to fall into one of several categories. When used in combination with constraint-based configuration scenarios, estimating the user's response to various individual items (or new services) can be considered a classification issue. Prediction looks at a specific value and makes a prediction. Predicting user individual pricing limitations (for example, predicting an individual`s top price limit) could be classified as a prediction problem.

The efficiency and effectiveness of the machine learning solution are defined by the type and quality of data as well as the effectiveness of such learning algorithms. To effectively design data-driven systems, machine learning algorithms like classification analysis, regression, data clustering, feature generation, dimension reduction, association rule learning, as well as reinforcement learning exist. Deep learning is also part of a larger collection of machine learning approaches that originated from artificial neural networks that may be used to successfully evaluate data. As a result, selecting the best learning algorithm for a given domain's goal application is difficult.

Here, the machine learning concept is important as we have trained our model in such a way that it can perform some kind of a human task. The machine has been trained once based on the train data and it has analysed the pattern of the same. So whenever we send a new data(test data) to the machine it can identify the pattern based on the past experiences and predict the expected output. The time taken by the miniSAT to run those 100 different sized SAT instances individually has been recorded in our model. When the new SAT instance is run through the miniSAT our model can thus predict the expected time the miniSAT will take to generate the output by calculating the all the different aspects of miniSAT(literals, clauses, time, conflicts, restarts, etc)

The main way to integrate machine learning and constraint resolution is supervised machine learning. Researcher’s contributions are divided into four areas based on the machine learning methods used (decision trees, neural networks, etc.), application methods for constraint resolution (algorithm selection, heuristic learning, etc.), and evaluation methods (performance, etc.) can be divided.

**Data Cleaning and Processing**

We have used python code in Jupyter notebook to process our data. Jupyter Notebook is like an open source app that allows users, scientists, scholars, or analysts to create and share documents called notebooks that include live programs, documents, graphs, plots, and visualizations. Jupyter Notebook supports over 40 programming languages, including the most popular languages ​​such as Python, R and Julia. Users can save the notebook as a PDF, HTML, Python, Markdown, or .ipynb file.

So to start with, we run each file of SAT instances through our miniSAT solver. The output that we got was stored in a single folder named DATA. We sorted the output files based on their extension(.csv) and called them on the jupyter notebook. Then we read each file on the notebook one by one through the DATA folder and separated them using the delimiter. This helped us to access each column within each file and thus we were able to access the ’time’ column which is our target column in the modelling process.

The most time consuming part of the project was data processing as it took hours and days to run each file to miniSAT and then store it onto the machine. After which those outputs of 100 files were stored in a single file and separated by the delimiter. There were many instances when the whole virtual machine crashed as it was difficult for the machine to process such a large file of thousand lines through the miniSAT. To overcome this, we had split the data into 2 files named df1 and df2 and processed them separately. After which we merged them using the panda framework.

Our first objective was to generate miniSAT output. This information is created, and part of it is displayed here.

['{"Event":"Restart","Time":0.005,"AdjustCount":99,"Restarts":1,"Conflicts":1,"Decisions":19,"Propagations":32,"Vars":194,"Clauses":1725,"Literals":6876,"LEARNT\_Limit":575,"LEARNT\_Clauses":1,"LEARNT\_Lit/Cl":3,"Progress":3.003}',

'{"Event":"LearnMore","Time":0.006,"AdjustCount":150,"Restarts":1,"Conflicts":100,"Decisions":981,"Propagations":2089,"Vars":194,"Clauses":1725,"Literals":6876,"LEARNT\_Limit":632,"LEARNT\_Clauses":100,"LEARNT\_Lit/Cl":5,"Progress":3.003}',

'{"Event":"Restart","Time":0.006,"AdjustCount":149,"Restarts":2,"Conflicts":101,"Decisions":1018,"Propagations":2168,"Vars":194,"Clauses":1725,"Literals":6876,"LEARNT\_Limit":632,"LEARNT\_Clauses":101,"LEARNT\_Lit/Cl":5,"Progress":3.008}',

'{"Event":"Restart","Time":0.007,"AdjustCount":49,"Restarts":3,"Conflicts":201,"Decisions":1677,"Propagations":3826,"Vars":194,"Clauses":1725,"Literals":6876,"LEARNT\_Limit":632,"LEARNT\_Clauses":201,"LEARNT\_Lit/Cl":5,"Progress":3.003}',

'{"Event":"LearnMore","Time":0.008,"AdjustCount":225,"Restarts":3,"Conflicts":250,"Decisions":2277,"Propagations":5217,"Vars":194,"Clauses":1725,"Literals":6876,"LEARNT\_Limit":695,"LEARNT\_Clauses":250,"LEARNT\_Lit/Cl":5,"Progress":3.003}'

The next stage is to separate and organise all of the CSVs, which is done using machine learning methods.

[{"Event":"Restart","Time":0.002,"AdjustCount":99,"Restarts":1,"Conflicts":1,"Decisions":38,"Propagations":67,"Vars":194,"Clauses":1725,"Literals":6878,"LEARNT\_Limit":575,"LEARNT\_Clauses":1,"LEARNT\_Lit/Cl":5,"Progress":3.003},

166129

| **Event** | **Time** | **AdjustCount** | **Restarts** | **Conflicts** | **Decisions** | **Propagations** | **Vars** | **Clauses** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Restart | 8.502 | 5093 | 73 | 20652 | 75009 | 38723637 | 30171 |
| **1** | Restart | 0.002 | 99 | 1 | 1 | 38 | 67 | 194 |
| **2** | LearnMore | 0.003 | 150 | 1 | 100 | 581 | 1487 | 194 |
| **3** | Restart | 0.003 | 149 | 2 | 101 | 633 | 1591 | 194 |
| **4** | Restart | 0.004 | 49 | 3 | 201 | 1709 | 4042 | 194 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **166119** | Restart | 8.073 | 5898 | 69 | 19847 | 73620 | 36899595 | 30183 |
| **166120** | Restart | 8.165 | 5696 | 70 | 20049 | 74117 | 37272596 | 30171 |
| **166121** | Restart | 8.402 | 5293 | 71 | 20452 | 74665 | 38272343 | 30171 |
| **166122** | Restart | 8.448 | 5193 | 72 | 20552 | 74821 | 38471335 | 30171 |
| **166123** | Restart | 8.502 | 5093 | 73 | 20652 | 75009 | 38723637 | 30171 |

With the data, we created a csv file. The data segregates the entire output from MiniSAT into usable data, which is then worked on and pre processed for use in models and predictions.

|  | **Event** | **Time** | **AdjustCount** | **Restarts** | **Conflicts** | **Decisions** | **Propagations** | **Vars** | **Clauses** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Restart | 0.005 | 99 | 1 | 1 | 19 | 32 | 194 | 1725 |
| **1** | LearnMore | 0.006 | 150 | 1 | 100 | 981 | 2089 | 194 | 1725 |
| **2** | Restart | 0.006 | 149 | 2 | 101 | 1018 | 2168 | 194 | 1725 |
| **3** | Restart | 0.007 | 49 | 3 | 201 | 1677 | 3826 | 194 | 1725 |
| **4** | LearnMore | 0.008 | 225 | 3 | 250 | 2277 | 5217 | 194 | 1725 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2061** | Restart | 25.311 | 71388 | 2042 | 925980 | 1467142 | 27204593 | 193 | 1709 |
| **2062** | Restart | 25.421 | 68188 | 2043 | 929180 | 1471370 | 27311810 | 193 | 1709 |
| **2063** | Restart | 25.627 | 61788 | 2044 | 935580 | 1479694 | 27522689 | 193 | 1709 |
| **2064** | Restart | 26.017 | 48987 | 2045 | 948381 | 1497292 | 27929286 | 193 | 1709 |
| **2065** | Restart | 26.804 | 23387 | 2046 | 973981 | 1532298 | 28748380 | 191 | 1674 |

In this process the data was processed and shuffled which was essential for training the model.

The model will be trained next during the prediction process, and we're doing the same before generating the model.

| **Time** | **AdjustCount** | **AdjustCount\_std** | **Restarts** | **Restarts\_std** | **Conflicts** | **Conflicts\_std** | **Decisions** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 193.801 | 2.903800e+06 | 2209849.0 | 16372.012228 | 9464.523048 | 1.065717e+07 | 6.482452e+06 |

After training the model we used the pyCaret(machine learning library) to compare the models and choose the best model for modelling.

**PyCaret**

PyCaret is an open source Python machine learning toolkit that simplifies the general activities of machine learning projects. This is a Python variant of the R program Caret, known for its ability to evaluate, compare, and optimize models with very few lines of code for a particular dataset. With a single function call, the PyCaret package allows Python machine learning experts to sample a set of typical ml algorithms on regression and classification datasets.

In our modelling process, we called the PyCaret package and passed the folder through it in which we had stored the output of the data which was generated in the form of output from the miniSAT. The target was set to CPU time as that is the entity which is to be predicted for new instances.This toolkit has helped us in comparing the models for the particular instances and predict which regression model performs better considering our modelling.

The comparison of the models that was generated by using the PyCaret package:

|  | **Model** | **MAE** | **MSE** | **RMSE** | **R2** | **RMSLE** | **MAPE** | **TT (Sec)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **omp** | Orthogonal Matching Pursuit | 0.6823 | 1.9020 | 1.1443 | 0.9981 | 0.0604 | 2.9359 | 0.0110 |
| **lasso** | Lasso Regression | 0.8899 | 4.4269 | 1.4037 | 0.9978 | 0.0757 | 1.6964 | 0.3750 |
| **en** | Elastic Net | 0.8930 | 4.4607 | 1.4080 | 0.9978 | 0.0756 | 1.6881 | 0.0110 |
| **br** | Bayesian Ridge | 0.9812 | 5.3579 | 1.5805 | 0.9972 | 0.0860 | 2.1159 | 0.0050 |
| **ridge** | Ridge Regression | 0.9864 | 5.3887 | 1.5929 | 0.9971 | 0.0868 | 2.0020 | 0.0080 |
| **lr** | Linear Regression | 1.5781 | 12.8927 | 2.3742 | 0.9840 | 0.1863 | 6.0785 | 0.8910 |
| **gbr** | Gradient Boosting Regressor | 3.3788 | 136.0956 | 6.8969 | 0.9636 | 0.0863 | 0.4053 | 0.0130 |
| **rf** | Random Forest Regressor | 3.8301 | 225.4265 | 7.6094 | 0.9537 | 0.1317 | 1.2551 | 0.0880 |
| **llar** | Lasso Least Angle Regression | 4.9972 | 67.6430 | 7.0878 | 0.9243 | 0.7906 | 80.4020 | 0.0140 |
| **dt** | Decision Tree Regressor | 4.9824 | 212.8911 | 9.2000 | 0.9099 | 0.2062 | 0.3836 | 0.0110 |
| **ada** | AdaBoost Regressor | 5.0696 | 170.4243 | 8.4463 | 0.9094 | 0.6478 | 69.0119 | 0.0210 |
| **et** | Extra Trees Regressor | 3.8117 | 234.4693 | 8.2368 | 0.8108 | 0.1128 | 0.2705 | 0.0600 |
| **huber** | Huber Regressor | 12.1141 | 1272.4823 | 24.2921 | 0.0496 | 0.5813 | 23.5902 | 0.0110 |
| **knn** | K Neighbors Regressor | 18.2651 | 1481.2856 | 31.8068 | -0.1186 | 0.6727 | 33.1075 | 0.0100 |
| **lightgbm** | Light Gradient Boosting Machine | 21.3227 | 1185.2692 | 31.8460 | -0.4206 | 1.1381 | 209.3361 | 0.0110 |
| **dummy** | Dummy Regressor | 35.4549 | 2920.5086 | 49.3418 | -3.3644 | 2.0024 | 598.9427 | 0.0070 |
| **par** | Passive Aggressive Regressor | 35.8720 | 7824.0164 | 66.2662 | -4.3910 | 1.0362 | 184.3619 | 0.0060 |
| **lar** | Least Angle Regression | 124.7349 | 152315.1668 | 204.9995 | -817.0070 | 1.4953 | 1510.6991 | 0.0120 |

**Statistical Measures in Modelling:**

* + 1. MAE(Mean Absolute Error)

There are many ways to determine the correctness of a model. Mean Absolute Error or MAE is one of many indicators used to summarize and evaluate the quality of machine learning algorithms.

What does the word "error" mean in this metric? Subtract the predicted value from the actual value, as shown below.

An error occurred while predicting the actual value of the predicted value.

Actual Value - Predicted Value ->Prediction Error

This predicted error is calculated for each record, and then all errors are converted to positive. This is accomplished by calculating the absolute value of each inaccuracy as shown below:

Absolute Error → |Prediction Error|

Finally, we compute the average of all absolute errors observed (Average sum of all absolute errors).

MAE stands for "Mean of All Absolute Errors."

* + 1. MSE(Mean Squared Error)

The level of inaccuracies in a statistical model is measured by mean squared error (MSE). The average of the squares of the difference between the observed and expected values ​​is calculated. If there are no errors in the model, MSE is zero. Its value increases as the model becomes more inaccurate. Root-mean squared deviation is another name for squared error (MSD).

* + 1. RMSE(Root Mean Squared Error)

A statistic that informs us the square root of the mean squared difference between a dataset's expected and actual values. The RMSE indicates how well a model matches a dataset. The better a model matches a dataset, the lower the RMSE.

It's produced as follows:

RMSE = √Σ(ŷi – yi)2 / n

where:

Σ means “sum”

ŷi means predicted value of the ith observation

yi means observed value of the ith observation

n means sample size

* + 1. R2

R-squared is used to calculate the spread of data sets around the regression line. The determination coefficient or coefficient of multiple determination is another name for it in multiple regression. For the same data set, higher R-squared values mean fewer differences between observed and fitted values. R2 basically defines how much of the observed variation can be explained by a particular model’s input.

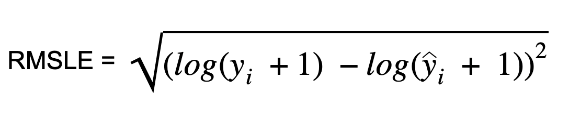
A linear model explains a fraction of the variation in the dependent variable with R-squared.

{\displaystyle R^2 = \frac {\text{Variance explained by the model}}{\text{Total variance}}}

* + 1. RMSLE(Root Mean Squared Logarithmic Error)

The root mean square logarithmic error is determined by applying the log and then calculating the difference between the actual and expected values. RMSLE is tolerant of outliers and treats small and large errors equally.

The penalty for the model will be higher if the value achieved is lower than the actual value, and it will be smaller if the calculated value is greater than the actual value. There is no penalty for high errors due to the protocol. As a result, the model penalizes underestimation rather than overestimation. This is useful if you don't mind overestimating but don't want to underestimate.



* + 1. MAPE(Mean Absolute Percentage Error)

One of the most commonly studied statistics to measure a model's prediction accuracy is MAPE statistics, which stands for mean absolute percentage error.

MAPE is calculated using the following formula:

MAPE = (1/n) \* Σ(|actual – forecast| / |actual|) \* 100

where:

Σ means “sum”

n means sample size

actual refers to the actual data value

forecast refers to the forecasted data value

MAPE is popular because it is simple to understand and apply. A MAPE of 8%, for example, indicates an average difference of 8% between the projected and actual value.

From the above table of models and their accuracy of predictions we only focused on 4 important machine learning models considering its simplicity, interpretability, scientific acceptance, and widespread availability. Rest of the models have some or the other drawbacks which would make it difficult to create good model.

For example, Orthogonal Matching Pursuit(OMP) are complex and time consuming in nature, Lasso Regression(lasso) are automatic in nature making it irresponsible in behaviour, Elastic Net(en) have high computation cost, Bayesian Ridge(br) doesn’t have a good approach when there is a huge amount of data, etc.

**Linear Regression**

Because a weighted sum of the attribute inputs, a linear regression model estimates the goal. The learned relationship's linearity makes interpretation simple. Statisticians, computer scientists, and others who work with numbers have long relied on linear regression models.

Linear regression is appealing because of its simplicity of depiction. The linear equation (y) represents the set of input values ​​(x) and the expected result (y) of those values. Therefore, it does not matter whether the input value (x) or the output value (y) is a number.

Each data entry or segment of a linear equation is described by a scaling factor, also known as a factor. Each input value or column is represented in Greek uppercase beta (B). Increase the degree of freedom of the line by adding coefficients, also known as intersection or bias coefficients (such as peaks and valleys in a 2D plot).

The following is an example of a simple regression model (x and y).

B0 + B1 \* x is the equation for y.

In higher dimensions, lines with many entrances (x) are called planes or hyperplanes. Then the form of the equation and the values ​​of the coefficients are displayed (B0 and B1 in the example above). The complexity of regression models, such as linear regression, is often discussed. The number of coefficients is the total number of coefficients utilised in the model. If the variable has a coefficient of zero, the input variable has no effect on the model and is ignored in the prediction (0 \* x = 0). The regularization approach minimizes the complexity of the regression model by modifying the learning strategy, thereby reducing some coefficients to zero.

In straight line regression we can calculate the parameters of simple linear regression with statistics if we just have one input. The mean, standard deviation, correlation, and covariance of said data must all be calculated. All data must be available. Over here we have calculated all of these things for each column and stored it in a folder.

If we had multiple inputs, the usual least squares method is to estimate the coefficients. The usual least squares is intended to minimize the sum of square roots of the residuals. In this scenario, the regression line is calculated by adding the squared error and squared the distance between each dataset and the regression line. Therefore, the simplest least squares method aims to reduce the root-mean squares error. The ideal coefficient values ​​for the data represented as a matrix are calculated using linear algebra techniques. That is, we need enough disk space to access all the data, store the data, and perform matrix calculations. This kind of analysis is known as least square analysis.

In descent by gradient there are given one or more inputs, the coefficient values ​​can be optimized by repeatedly minimizing errors in the training data model. Each coefficient of the steepest descent method is initially given a random value. The root-mean-squared error for each pair of input and output values ​​is determined. To reduce inaccuracies, the learning rate is used as the scaling factor and the factor is changed. No further improvement is possible unless the minimum error sum of squares is achieved. We need to select the learning rate (alpha) option for this approach. It defines how much progress we can make with each iteration of the procedure.

Standardization is nothing but regularization method of training linear models. In both cases, we strive to lower the model's complexity while minimizing the squared error on the training data (like the number of coefficients or the absolute size of their sum).

The mean absolute error that we got from linear regression model for this particular instance is 1.5781 which is acceptable.

The mean square error of the linear regression model is 12.8927 which is also quite less making it less inaccurate.

The linear regression model's root mean squared error appears to be 2.3742, indicating that the gap between the dataset's expected and actual value is less.

The linear regression model's higher R-squared value of 0.9840 suggests that there are less differences among actual and estimated parameters for the given set of data.

Considering all these parameters, linear regression seems to be a good model for prediction of the runtime of the miniSAT.

**Random Forest**

Random forests are used in supervised learning. They are a robust nonparametric statistical technique for considering regression, two-class, and multi-class classification issues in an uniform and adaptable framework. In this method, a set of decision trees is formed as a "forest" and is usually trained using the bagging method. The bagging method improves the end result by shuffling a large number of learning models. Random forest predictions are more accurate and reliable because they combine multiple decision trees. Random forests have the advantage of being able to address classification and regression issues compared to other machine learning systems. Classification is usually considered the basic foundation of machine learning, so here we considered a random forest method for classification.

Decision trees and bagging classifiers have hyperparameters that are similar to those of random forests. A random forest classifier class could be used instead of a decision tree or a bagging classifier if we didn’t wanted to use both. Regression problems are solved using the algorithm's regressor. When we add a random forest to the model, the trees become more surprising. Rather than the most essential trait, nodes are split based on the best among a series of randomly selected attributes. As a result, the model benefits from variation.

As a result, when splitting into a random forest, only a random subset of the cluster's attributes are considered. Using a random threshold for each attribute, like a traditional decision tree, allows the tree to be much more random than searching for the highest possible threshold.

Going through the output table of the comparing the models for the particular instance, the random forest has the mean absolute error of 3.8301 which is average.

The mean squared error is 225.4265 which seems high and thus makes this model incompatible for our modelling.

**Decision Tree Regressor**

It is a convenient tool that can be used in various situations. Regression problems can be solved in the same way. Displays predictions in a tree-like flowchart based on a series of feature-based splits. Leaf selection is made from the root node. Now that we know the amount of entropy and how to calculate it, we can understand how to calculate entropy. Node contamination is measured by its entropy. This is unpredictable due to the apparent randomness or contamination of the data. With a pure sub split, we get both "yes" and "no" answers.

Because a problem usually contains a lot of features, there are a lot of splits, leading in a big tree. Due to which these trees, overfitting may arise. When is the best time to come to a halt? Set minimum amount of training data inputs to be used on each leaf is one way to accomplish this. We could draw a judgement based on at least 10 instances, and therefore any leaf with fewer than 10 would be dismissed. We might additionally specify the depth of our model. The critical path among a root and a leaf is referred to as maximum depth.

In the above generated output data, decision tree regressor performs average as the mean squared error of the method is 212.8911. This is high when compared to other models making it less accurate in modelling.

**K-Nearest Neighbor (k-NN)**

The k-Nearest Neighbor algorithm is a simple but powerful machine learning algorithm (kNN). It can be used to classify and forecast. It is, however, most often used to forecast classifications. Resemblances to previously trained data, kNN organises newly entered records into logical clusters and subsets. The input is sent to the subclass with the nearest neighbours. Despite its effectiveness, kNN has a lot of limitations. The kNN algorithm including modified variations of the algorithm discovered in prior works are investigated in this paper. These versions of kNN give a more effective approach to kNN by removing its shortcomings.

KNN uses a case-based learning process that requires it to keep every training information in order to categorize it. Because it is a slow training method, interactive web extraction for a large dataset is not appropriate. Inductive learning, which comprises creating an inductive learning technique from a training sample and using its presentation (representatives) for classification, is one strategy to increase its performance. Many techniques, such as decision trees as well as neural networks, are now particularly developed to build such models. The efficacy of several algorithms is evaluated. As kNN is indeed a simple, yet efficient technique to classifying time, we created a model to boost its efficiency while maintaining its classification accuracy.

Having a look at the output generated on the training data by k-NN method, the R2 is -0.1186. This shows that the k-NN model can’t explain any of the observed variation in the model’s input. Thus this is not the right model to predict the runtime of miniSAT.

This prediction was done roughly based just by observing the certain statistical measures which were generated from the PyCaret. Now to prove our prediction related to the best model amongst the 4 models that we choose, we trained these models on the 20% of our data. For this 28 csv files were selected containing a bunch of data and the CPU runtime.

| **MAE** | **MSE** | **RMSE** | **R2** | **RMSLE** | **MAPE** |
| --- | --- | --- | --- | --- | --- |
| **0** | 17.2180 | 632.9472 | 25.1584 | 0.4954 | 0.7095 | 129.8370 |
| **1** | 15.8403 | 943.7001 | 30.7197 | 0.7916 | 0.5618 | 35.8779 |
| **2** | 9.2179 | 290.2399 | 17.0364 | 0.1268 | 0.4791 | 1.3744 |
| **3** | 37.9619 | 3757.9490 | 61.3021 | -0.2847 | 1.0815 | 19.0889 |
| **4** | 40.3723 | 6587.2046 | 81.1616 | 0.1590 | 0.9351 | 1.5550 |
| **5** | 14.1371 | 519.1176 | 22.7842 | 0.0824 | 0.5712 | 0.8550 |
| **6** | 3.0265 | 28.3351 | 5.3231 | 0.7970 | 0.5914 | 137.2687 |
| **7** | 6.3741 | 180.0311 | 13.4176 | -4.7601 | 0.5737 | 3.3717 |
| **8** | 15.1463 | 1008.1074 | 31.7507 | 0.8071 | 0.3719 | 0.4563 |
| **9** | 23.3569 | 865.2239 | 29.4147 | 0.5993 | 0.8518 | 1.3901 |
| **Mean** | 18.2651 | 1481.2856 | 31.8068 | -0.1186 | 0.6727 | 33.1075 |
| **SD** | 11.7976 | 1975.2062 | 21.6705 | 1.5864 | 0.2089 | 51.3901 |

Whether it's mean square error (MSE), mean absolute error (MAE), or root mean square error, linear regression provides a higher accuracy (R2 score) and lower errors (RMSE). Random forest regressor will be the second rank. Then the regressor Decision Tree. The KNN regressor is in last place.

/\* Real time of csv file is "27.3807" and the predicted time model (knn\_model) is "24.385000228881836"

Real-time of csv file is "27.3807" and the predicted time model (decisiontree\_model) is "28.631"

Real-time of csv file is "27.3807" and the predicted time from (randomForest\_model) is "28.178"

Real time of csv file is "27.3807" and the predicted time from (linearReg\_model) is "27.072999954223633" \*/

From this data it is clear that our prediction of linear regression model being the best amongst all the 4 models is correct as it predicts the closet possible output to the actual output generated from the miniSAT. Also, as linear regression model is simple to train data on, it has lower time complexity, has easy mathematical equations it becomes a great model for our predictions.

Overfitting occurs whenever a machine learning model tightly fits a dataset and hence catches the noisy data. This diminishes the model's efficiency upon that test set and has a negative impact on its performance.Regularization is a simple technique that effectively reduces a function's complexity, lowering the danger of overfitting which can be easily implemented on the linear regression model.

# Conclusion

This investigation challenges the basic assumptions about the run-time behavior of the complete search solver. Most modern solutions involve some sort of randomness. As a result, their execution times vary significantly from instance to instance, often orders of magnitude. We have seen how the results of empirical comparisons such as SAT can be misleading. We impose statistical limits on such deviations, as competition can fluctuate just by repeating the test. It also applies the vulnerabilities of current maturity prediction models to these maturity distributions, demonstrating that current practice is not sufficient to take just one example of maturity. Solver ratings, portfolios, automated installations, and run-time predictions are just a few of the areas where such insights provide widespread results for empirical comparisons.

This investigation challenges the basic assumptions about the run-time behavior of the complete search solver. To counter this problem, there is actually a context-sensitive strategy rather than a single answer. The empirical assessment should compare the MiniSAT solvers based on their run-time distribution. The Kolmogorov Smirnov test and the Chisquared test are useful. Of course, this increases processing costs, but has the advantage of making the conclusions more reliable. A related question is how many run-time samples need to be taken to get representative results.

We've looked into picking fundamental elements like a basic task of algorithms, parameterization, search heuristic selection, performance prediction, and satisfiability prediction. This architecture predicts the runtime of miniSAT with a prediction accuracy of 28 and a time complexity of 28 deep NNs. Deep learning is employed for the first time on random CSPs with no influence on satisfiability due to constraint tightnesses or constraint densities. CSPs' NP-hardness frequently precludes deep learning from properly using training data. Massive volumes of labelled data were generated using a machine learning method to address this issue.

As a result, we can infer that no realistic model for predicting the runtime of a SAT solver exists because the Boolean Satisfiability issue is dynamic, and the output of each instance varies depending on a variety of parameters. Though the total research yields a model that predicts runtime that is near to the accurate runtime. As a result, with the data provided, the linear regression model performs best and forecasts the miniSAT's runtime with near-perfect accuracy when unsatisfactory examples are processed through it.

The objectives of our project were achieved as we were able to demonstrate the runtime variations that occurred in the comparisons of the outcomes and build the best model for the given datasets in the given period of time. Though the SAT instances were huge and took many huge time to run, we were successfully able to process the data which helped us in building the model. The statistical analysis was provided that helped to compare the performance of solvers from a runtime distribution perpective.

There are many unanswered questions in the areas of runtime prediction and solver portfolio, including automatic configurators. It is not enough to get a single sample of the solver's execution time as a ground truth. Alternatively, you can try to estimate run-time distribution statistics or parameters. Depending on your application, you can try to maximize the probability of resolving your instance within a specific amount of time. You may prefer a solver that behaves more consistently, or you may prefer a solver with many variations in the hope of luck. This had a knock-on effect even in a parallel environment. There are many possibilities for future research in these areas.

## Future Work

Though we were able to create a model which predicts the runtime of the miniSAT still there is a bit difference between the predicted and actual output.

The modelling would have been done better and more accurate if we have used more SAT instances. But due to the time constraints, we were only able to build our model on 100 instances.

There are many researchers who have used new generation solver that are easy and fast to understand. Those solvers would have made the modelling easy and more accurate.

# References

[1] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve Circuit-SAT: An

unsupervised differentiable approach. In International Conference on Learning Representations.

2019.

[2] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of SAT

formulas. In Theory and Applications of Satisfiability Testing – SAT 2012, pages 410–423,

2012.

[3] Adrian Balint and Uwe Schöning. Choosing probability distributions for stochastic local search

and the role of make versus break. In Theory and Applications of Satisfiability Testing – SAT

2012, pages 16–29, 2012.

[4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. Science,

286(5439):509–512, October 1999.

[5] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner,

Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani,

Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra,

Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational

inductive biases, deep learning, and graph networks. arXiv:1806.01261, October 2018.

[6] Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve

real-world SAT instances. In Proceedings of the Fourteenth National Conference on Artificial

Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence, pages

203–208, 1997.

[7] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940, January 2017.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning.

In Proceedings of the 26th Annual International Conference on Machine Learning, pages 41–48,

2009.

[9] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial

optimization: A methodological tour d’horizon. arXiv:1811.06128, November 2018.

[10] Armin Biere. Yet another local search solver and Lingeling and friends entering the SAT

competition 2014. In Proceedings of SAT Competition 2014, pages 39–40. 2014.

[11] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh. Handbook of Satisfiability.

IOS Press, 2009.

[12] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling,

Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton.

A survey of Monte Carlo tree search methods. IEEE Transactions on Computational Intelligence

and AI in Games, 4(1):1–43, March 2012.

[13] Stephen A. Cook. The complexity of theorem-proving procedures. In Proceedings of the Third

Annual ACM Symposium on Theory of Computing, pages 151–158, 1971.

[14] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random

3-SAT. Artificial Intelligence, 81(1):31–57, March 1996.

[15] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Theory and Applications of

Satisfiability Testing – SAT 2004, pages 502–518, 2004.

[16] Paul Erdos and Alfréd Rényi. On random graphs. I. ˝ Publicationes Mathematicae, 6:290–297,

1959.

[17] Andreas Fröhlich, Armin Biere, Christoph Wintersteiger, and Youssef Hamadi. Stochastic local

search for satisfiability modulo theories. In Proceedings of the Twenty-Ninth AAAI Conference

on Artificial Intelligence, pages 1136–1143, 2015.

[18] Alex S. Fukunaga. Evolving local search heuristics for SAT using genetic programming. In

Genetic and Evolutionary Computation – GECCO 2004, pages 483–494, 2004.

[19] Alex S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing.

Evolutionary Computation, 16(1):31–61, March 2008.

10

[20] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl.

Neural message passing for quantum chemistry. In Proceedings of the 34th International

Conference on Machine Learning, pages 1263–1272, 2017.

[21] Michele Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph

domains. In Proceedings of the 2005 IEEE International Joint Conference on Neural Networks,

volume 2, pages 729–734, July 2005.

[22] Marketa Illetskova, Alex R. Bertels, Joshua M. Tuggle, Adam Harter, Samuel Richter, Daniel R.

Tauritz, Samuel Mulder, Denis Bueno, Michelle Leger, and William M. Siever. Improving

performance of CDCL SAT solvers by automated design of variable selection heuristics. In

2017 IEEE Symposium Series on Computational Intelligence (SSCI), pages 1–8, November

2017.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training

by reducing internal covariate shift. In Proceedings of the 32nd International Conference on

International Conference on Machine Learning, pages 448–456, 2015.

[24] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial

optimization algorithms over graphs. In Advances in Neural Information Processing Systems

30, pages 6348–6358. 2017.

[25] Ashiqur R. KhudaBukhsh, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. SATenstein:

Automatically building local search SAT solvers from components. Artificial Intelligence, 232:

20–42, March 2016.

[26] Jeong Han Kim and Van H. Vu. Generating random regular graphs. In Proceedings of the

Thirty-Fifth Annual ACM Symposium on Theory of Computing, pages 213–222, 2003.

[27] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In

International Conference on Learning Representations. 2019.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep

convolutional neural networks. In Advances in Neural Information Processing Systems 25,

pages 1097–1105. 2012.

[29] Michail G. Lagoudakis and Michael L. Littman. Learning to select branching rules in the DPLL

procedure for satisfiability. Electronic Notes in Discrete Mathematics, 9:344–359, June 2001.

[30] Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals. CNFgen: A generator of

crafted benchmarks. In Theory and Applications of Satisfiability Testing – SAT 2017, pages

464–473, 2017.

[31] Gil Lederman, Markus N. Rabe, Edward A. Lee, and Sanjit A. Seshia. Learning heuristics for

automated reasoning through deep reinforcement learning. arXiv:1807.08058, April 2019.

[32] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In Advances in Neural Information Processing Systems

31, pages 539–548. 2018.

[33] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential recency

weighted average branching heuristic for SAT solvers. In Proceedings of the Thirtieth AAAI

Conference on Artificial Intelligence, pages 3434–3440, 2016.

[34] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based

branching heuristic for SAT solvers. In Theory and Applications of Satisfiability Testing – SAT

2016, pages 123–140, 2016.

[35] Jia Hui Liang, Hari Govind V.K., Pascal Poupart, Krzysztof Czarnecki, and Vijay Ganesh. An

empirical study of branching heuristics through the lens of global learning rate. In Theory and

Applications of Satisfiability Testing – SAT 2017, pages 119–135, 2017.

[36] Mathew Penrose. Random Geometric Graphs. Oxford University Press, May 2003.

[37] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall

Press, 3rd edition, 2009.

[38] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini.

The graph neural network model. IEEE Transactions on Neural Networks, 20(1):61–80, January

2009.

11

[39] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing.

In Cliques, Coloring, and Satisfiability, pages 521–532, 1993.

[40] Bart Selman, David G. Mitchell, and Hector J. Levesque. Generating hard satisfiability problems.

Artificial Intelligence, 81(1):17–29, March 1996.

[41] Daniel Selsam and Nikolaj Bjørner. Guiding high-performance SAT solvers with unsat-core

predictions. In Theory and Applications of Satisfiability Testing – SAT 2019, pages 336–353,

2019.

[42] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L.

Dill. Learning a SAT solver from single-bit supervision. In International Conference on

Learning Representations. 2019.

[43] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur

Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap,

Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters

chess, shogi, and Go through self-play. Science, 362(6419):1140–1144, December 2018.

[44] Richard S. Sutton and Andrew G. Barto. Introduction to Reinforcement Learning. MIT Press,

1st edition, 1998.

[45] Tijmen Tieleman and Geoffrey E. Hinton. RMSProp: Divide the gradient by a running average

of its recent magnitude. Slides of lecture “Neural Networks for Machine Learning”, 2012.

[46] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of ‘small-world’ networks. Nature,

393(6684):440–442, June 1998.

[47] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3):229–256, May 1992.

[48] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based

algorithm selection for SAT. Journal of Artificial Intelligence Research, 32(1):565–606, June

2008.

[49]Kotthoff et al., 2012; Hutter et al., 2014, Automated Algorithm Selection:Survey and Perspectives.

[50]Xu et al., 2011, Automated Algorithm Selection: Survey and Perspectives

[51]Ertekin et al., 2007, Basic Applied Reservoir Simulation

[52]Barry et al., 2018, Vector-based navigation using grid-like representations in artificial agents

[53]Lopez-Ib ´ anez and St ´ utzle, 2014, Automatically Improving the Anytime Behaviour of Optimisation Algorithms

[54]Finkler and Mehlhorn 1997; Smith-Miles and Lopes 2012; Heutter et al.2014, Runtime Prediction of Real Programs on Real Machines

[55] Nudelman et alinitial .`s, Understanding Random SAT: Beyond the Clauses-to-Variables Ratio.

[56]Xu, Hoos, and Leyton Brown (2012), Algorithm Runtime Prediction: Methods & Evaluation

2014, Hutter et al., An Efficient Approach for Assessing Hyperparameter Importance

[57] Felfernig et al., 2018, A Graphical Framework for Supporting Mass Customization

[58] Rossi et al., 2006; Tsang, 1993, New trends in constraint satisfaction, planning, and scheduling: A survey

[59] Gu et al., 1996, The Multi-SAT algorithm

.

Installation guide

Installation guide for the VMWare:-

<https://www.vmware.com/pdf/ws7_manual.pdf>

Installation guide for Ubuntu on VMWare:-

<https://linuxhint.com/install_ubuntu_vmware_workstation/>

Installation of miniSAT on Ubuntu:-

1. Download the "[minisat-2.2.0.tar.gz](http://minisat.se/downloads/minisat-2.2.0.tar.gz)". Go to the downloaded directory and open the Linux Terminal there.

2. Run "tar xvf [minisat-2.2.0.tar.gz](http://minisat.se/downloads/minisat-2.2.0.tar.gz)".

3. Run "cd minisat"

4. Run "export MROOT=$(pwd)"

5. Run "cd core".

6.Run "sudo apt-get install libghc-zlib-dev".

7.Run "make"

8. Run the application with "./minisat".