Predicting SAT Solver Performance on Heterogeneous Hardware

Zack Newsham, Vijay Ganesh, and Sebastian Fischmeister

University of Waterloo, Canada

Abstract. In recent years a lot of effort has been expended in determining if SAT solver performance is predictable. However, the work in this area invariably focuses on individual benchmarking machines, and often on individual solvers. It is unclear whether predictions made on a specific solver and benchmarking machine are accurate when translated to other solvers and hardware. In this work we consider five state-of-the-art solvers, 28 benchmarking machines and 144 feature instances selected from the 2011 — 2014 SAT competitions. Using combinations of solvers, machines and instances we present four results. First, we show that UNSAT instances are more predictable than corresponding SAT instances. Second, we show that the number of cores in a machine has more impact on performance than L2 cache size. Third, we show that instances with fewer reused clauses are more CPU bound than those where clause reuse is high. Finally, we make accurate predictions of solution time for each of the instances considered across a diverse set of benchmarking machines.

1 Introduction

Despite the focus in the SAT community on determining predictability of SAT instances, there is very little work concerning the effectiveness of predictions made on one machine when executing instances on a second machine. This has led to two problems. First, any tools which make predictions of solver performance can only make predictions based on instances that have been ran on the same machine the predictions are to be valid on. Secondarily, it is virtually impossible to compare results of past publications in this field without first repeating the experiments on the same hardware. Cross machine models will enable practitioners to make predictions of solver performance on new hardware without first having to run known benchmarks, reducing overall effort. Additionally, if it is possible to determine an "absolute" solution time for an instance — regardless of hardware used — it will be possible to combine benchmarking data from multiple sources. This will allow larger, more comprehensive studies of SAT solver performance to be undertaken without duplication of effort.

2 Experimental setup

The experiments included here were run on the Datamill [1] platform. Datamill is designed to run sets of experiments on a heterogeneous set of hardware, and optionally varies system level properties on execution. All machines on the Datamill

platform run Gentoo Linux with kernel version 3.3.8, and GCC 4.5.3. A full listing of the machines used is available in Appendix A.

Due to memory constraints within the different machines, and timing constraints for data gathering, it was not possible for every machine, instance, solver combination to complete successfully. To mitigate any bias this may cause in our analysis, we limit ourselves to the set of machines, instances and solvers such that every machine solved every instance in the set on at least one solver. This reduced the data available for analysis from 15 588 to 13 648, the number of machines reduced to 21 while the number of instances remained unchanged.

For each machine based parameter p_x and instance i we used a standard linear regression with Equation 1 to calculate the adjusted R^2 value r_{ip_x} for all trials t using the instance i. As such, the value r_{ip_x} expresses the amount of variability in solution time of instance i accounted for by factor p_x. The adjusted R^2 metric ranges from zero to one, with one being a perfect model and zero indicating that the provided model does not explain the response variable (time). For brevity the remainder of this document will use the notation $time \sim p$ in place of Equation: 1.

$$time_{it} \sim \beta_0 + \beta_1 p_{xt} + \epsilon_t \tag{1}$$

We then consider different instance based characteristics c_y to determine whether classes of instance (identified as having similar c_y values for specified values of y) are more or less reliant on different machine parameters.

In total we considered 34 instance characteristics and 7 machine parameters. While not all of them were found to be significant, those that were are discussed in this work. The machine parameters considered were: CPU speed, CPU architecture, CPU manufacturer, CPU cores, FSB speed, RAM size and L2 cache size. In the future we intend to increase this set of features, particularly concerning cache sizes and RAM speeds, however this information is not available for the test machines at this time. Modern CPU's express their FSB speed in GT/i whereas older CPUs utilise MHz, since GT/i is considered a more accurate measurement, which describes not only the clock speed of the bus but the data width, we converted all measurements of FSB speed to GT/i. We included CPU Model to determine if CPUs from different models/manufacturers with similar specifications perform differently, a reason for this could be cache replacement policies and implementation specific timing characteristics, such as the proportion of integer vs floating point cores within the CPU. The data in this paper is available online [2].The instance characteristics considered are listed in Appendix B.

The five solvers considered were MiniSAT 2.0 [3], Glucose 3.0 [4], Lingeling [5], Plingeling [6] and SWDiA5BY 2.3. These were the silver and gold medal winners of the 2013/2014 SAT competitions for the application category. While PeneLoPe was also a medal winner, it was excluded as it did not compile on the target environment.

The instances used in this work were randomly selected from the 2011 — 2014 SAT competitions. They were selected using a stratified random sampling

technique to ensure that a diverse set of instances was included in the sample. The stratified sample considered values of Q, $|Co|$, $|V|$, $|Cl|$, CVR and solution time. At most three instances were selected for each range of the measured property.

Each pair of instance and solver was then run on each of the 26 machines. Due to constraints of the DataMill platform, each trial was allowed a maximum of two hours for completion and trials were ran in batches. Each batch contained no more than 18 trials to keep the batch time (including setup, solving and collection) under two days, which is a requirement of DataMill. 40 batches were ran per machine, the order of trials was randomized prior to batching, as such each batch for each machine contained the same instances, additionally batches were ran in a random order.

From the outset we expected that memory size, CPU clock speed and cache size were going to be significant contributing factors, however as shown in [7] memory layout as well as other factors can have significant impact on any execution times, including that of a SAT solver. It was for this reason we decided to utilise DataMill [1].

Due to the heterogeneous nature of the hardware used, it was not possible to run all solvers on all machines. We found that the ARMV7 machines (machines 22-25 in Appendix A) were only able to run Lingeling, all the other solvers (including pLingeling) failed to compile due to a floating point library not being present in the ARM version of the operating system. Each machine was dedicated to running only the experiment given, and is not virtualised or shared in any way.

To remove the possibility that these timing differences were the result of randomness in the solver, we pre-simplified the instances then turned off simplification on the solvers, this is to resolve the known issue that clause (and variable) ordering has an effect on solution time [8]. In addition to this we set a fixed random seed for all the solvers on all machines (with the exception of pLingeling which did not support this option), such that a single instance/solver pair should have identical performance on an identically specified machine.

The complete set of all combinations of solvers, workers and instances would have created approximately 18 500 trials. Unfortunately even with the timeouts and batching we imposed, DataMill was unable to complete all of these trials within the required time frame. As such we are limited to analysing those results which were gathered, approximately 15 500 of them. Within these results approximately 1 600 were unable to run due to a lack of support in the ARM kernel, as mentioned previously. Leaving a total of 13 825 instances available for analysis. The majority of our analysis looks at aggregate results, either across the instances, solvers or workers. To ensure that no bias has been introduced by certain combinations of workers/instances not being included we only analyse the complete set of workers and instances, by which we mean that every worker in our final dataset solved every instance at least once. To accomplish this we used an implementation of maximal biclique enumeration algorithm from Alexe et al[9] where the workers are one half of the graph and the instances are the

other. The resulting clique included 13 648 trials, unless otherwise stated this is the set used in all results presented.

3   Results

The following sections detail the results mentioned above. In each of the four sections we discuss a single result providing evidence and levels of confidence of it, and discuss the consequences of that result.

3.1   UNSAT instance performance is more predictable than SAT

When analysing the set of all trials, regardless of solver, a clear trend can be observed regarding the predictability of instances. For all machine parameters considered, UNSAT instances were consistently more predictable than SAT instances. For example, using the model $time \sim CPUSpeed$ the maximum R^2 for UNSAT instances is 0.65, whereas the maximum R^2 for SAT instances is 0.64. While in this case the maximum R^2 do not differ significantly, 17.4% of the 69 UNSAT instances had an $R^2 > 0.5$ and 55.0% had an $R^2 > 0.3$. This is compared with SAT instances, where only 4.0% of the 74 SAT instances had an $R^2 > 0.5$ and 24.3% had an $R^2 > 0.3$.

The same trend, of UNSAT instances having a higher R^2 score, can be observed for FSB speed, RAM size, L2 size and number of CPU cores. Though it should be mentioned that SAT instances achieved a marginally higher maximum R^2 for L2 size (0.14) compared to UNSAT instances (0.13). However, fewer SAT instances achieved an $R^2 > 0.05$ than UNSAT.

Further results in Section 3.4 support the result that UNSAT instance performance is more predictable than SAT instance performance. To demonstrate this, we test the following hypothesis, where S is the set of solvers described in Section 2:

$$H1_0 : p(UNSAT_s) \leq p(SAT_s), \quad \forall s \in S$$
$$H1_a : p(UNSAT_s) > p(SAT_s), \quad \forall s \in S$$

The function p returns the probability that the predictions for the set of instances provided will be accurate to with 20% of the solution time of each instance. We then utilise a binomial probability test to determine whether the probability of an accurate prediction is higher for UNSAT instances than SAT, for each solver.

We performed the hypothesis test with a significance level of 0.05, as such we accept the null hypothesis in the cases where the p-value > 0.05. Table 1 shows the results of this hypothesis test. Each row in the table presents the result of the hypothesis test for a single solver, along with the probabilities of accurate predictions for the SAT and UNSAT instances on that solver, and finally whether we accept or reject the null hypothesis. For MiniSAT, Lingeling and pLingeling, UNSAT instances are more predictable than SAT instances, with a p-value low

enough that we reject the null hypothesis. However, In the cases of Glucose and SWDiA5BY, the results suggest that SAT instances are more predictive than UNSAT. It is unclear why Glucose and SWDiA5BY should show a different result than the other solvers and requires further experimentation at a later date.

Solver	$p(UNSAT)$	$p(SAT)$	p-value	Reject NULL
Minisat	47.3%	41.0%	<0.001	REJECT
Glucose	49.6%	50.5%	0.674	ACCEPT
Lingeling	56.4%	52.6%	0.022	REJECT
pLingeling	46.3%	41.7%	0.011	REJECT
SWDiA5BY	47.1%	50.6%	0.95	ACCEPT

Table 1: Shows the probability that a prediction for an instance will be less than 20% of that instances runtime. The result column shows the significance level of the test.

3.2 Number of cores is more important than L2 cache size

The second result from this experiment was that we found the number of CPU cores in a machine is more significant when determining performance than L2 cache size, for both SAT and UNSAT instances. This was an unexpected result, as the majority of solvers tested were sequential, as such their performance should not be impacted by multi core architectures.

The maximum R^2 for the model $time \sim L2$ was 0.14, compared with the maximum R^2 for the model $time \sim CPUCores$ which was 0.57.

To confirm the result that number of cores is more important in determining solver performance than L2 cache size, we test the following hypothesis, where S is the set of solvers described in Section 2 and the function p returns the probability that the provided machine parameters R^2 will exceed 0.1:

$$H2_0 : p(L2_s > 0.1) \leq p(CPUCores_s > 0.1), \quad \forall s \in S$$
$$H2_a : p(L2_s > 0.1) > p(CPUCores_s > 0.1), \quad \forall s \in S$$

Table 2 shows, for each solver, the probability that the value of R^2 will be exceed 0.1 for the model $time \sim L2$, the probability that the value of R^2 will exceed 0.1 for the model $time \sim CPUCores$, the p-value for the hypothesis test, and whether we reject or accept the null hypothesis. In every case the solver's performance is impacted significantly more by the number of CPU cores than L2 cache size.

One possible explanation for this result is the overall trend between both CPU speed and number of cores, and RAM size and number of cores. However,

Solver	$p(L2 > 0.1)$	$p(CPUCores > 0.1)$	p-value	Reject NULL
MiniSAT	12.6%	94.4%	<0.001	REJECT
Glucose	12.6%	96.5%	<0.001	REJECT
Lingeling	3.5%	35.0%	<0.001	REJECT
pLingeling	65.0%	95.0%	<0.001	REJECT
SWDiA5BY	18.9%	98.6%	<0.001	REJECT

Table 2: The results of the hypothesis test H2, showing the probability that the R^2 for the models $time \sim L2 > 0.1$ and $time \sim CPUCores$, as well as the p-value for the hypothesis test that $CPUCores > L2$

there are several machines that do not follow this trend (E.g., machines number 3,7 and 20). Another possible explanation for this result is that L2 cache size is not representative of the overall impact of cache size on solver performance. The L1 cache and lowest level cache (which in some cases will be L2) are potentially more significant than L2 cache size alone. The reason for excluding the L1 and "lowest level" cache sizes is that at present we do not have that data for all machines considered. This will be included in future versions of this work. A third possible explanation for the performance increase is that overhead in the operating system takes place on one core, while the solving of the SAT problem takes place, uninterrupted, on another. Table 3 shows the average execution time for instances solved on machines with different numbers of cores. The second, third columns and fourth columns show average execution times for all solvers, sequential and parallel solvers respectively. The small decrease of the average solution time for parallel solvers between four and eight cores would suggest that pLingeling (the only parallel solver considered) is either unable to utilise eight cores to their fullest potential, or some other parameter of the machine with eight cores is reducing performance enough to mask any improvement the additional cores provide.

	Average solution time (s)		
#Cores	All solvers	Sequential solvers	Parallel solvers
1	2528	2521	2557
2	1766	1780	1712
4	1246	1368	579
8	735	773	581

Table 3: Number of cores and the associated average execution time in seconds

3.3 A negative correlation exists between clause reuse and the impact of CPU speed on solver performance

In this section we use the term "clause reuse" to indicate sets of clauses with the same variables with different polarity. For example, in the DIMACS below clause c1 is said to be reused three times (in clauses c2 and c4). Clause c3 is seen only once.

$$
\begin{aligned}
&\text{c1: 1 2 3 0}\\
&\text{c2: -1 2 3 0}\\
&\text{c3: 1 3 -4 0}\\
&\text{c4: 1 -2 3 0}
\end{aligned}
$$

The terms max(reuse) and mean(reuse) refer to the maximum and average times any single clause is reused. Figures 1a and 1b show, for SAT and UNSAT instances respectively, the correlation between the average clause reuse for formula (on the X-axis) and the average R^2 of the model $time \sim CPUSpeed$ for those instances (on the y-axis). Each data point represents a 0.1 range of the average clause reuse. These figure show a negative correlation between the average clause reuse and the impact of CPUSpeed as a predictor of time. This negative correlation indicates that instances where individual clauses contain unique sets of variables are more CPU bound than those which have a small set of variables that are repeatedly assigned different values in clauses.
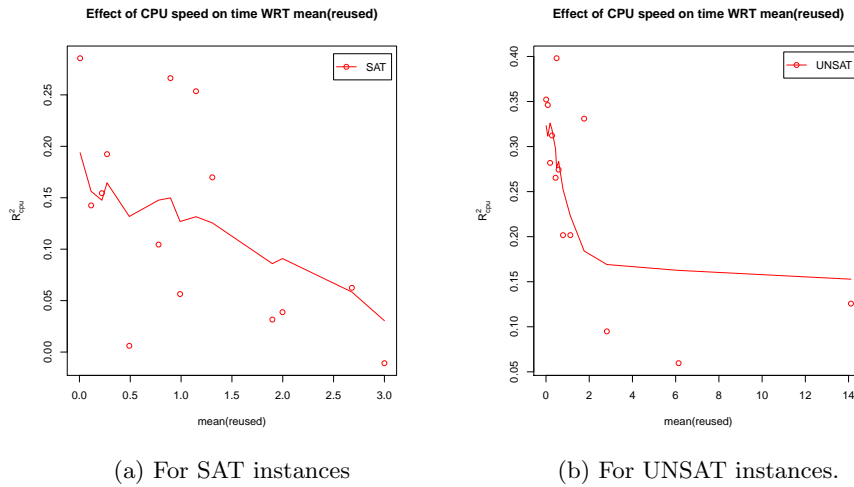

(a) For SAT instances   (b) For UNSAT instances.

Fig. 1: Figures showing the clause reuse against average R^2 for SAT and UNSAT instances, the line represents the moving average over a window of seven data points.

To confirm this result, we test the following two hypothesis, where S is the set of solvers described in Section 2:

$$H3_0 : p(time(i_l, s)) \leq p(time(i_h, s)), \quad \forall s \in S, \quad Mean.Reused(i_l) < Mean.Reused(i_h)$$
$$H3_a : p(time(i_l, s)) > p(time(i_h, s)), \quad \forall s \in S, \quad Mean.Reused(i_l) < Mean.Reused(i_h)$$

Function p returns the predictability of the runtime of a solver on a particular instance across all considered hardware. i_h and i_l refer to randomly selected instances with higher, and lower average clause reuse respectively.

For this experiment, we randomly selected 500 pairs of instances and checked whether the instance with the lower mean(reused) value was more CPU bound than the instance with the higher mean(reused). Table 4 shows the probability that for any randomly selected pair of instances (where the mean(reused) differ) the alternate hypothesis holds true. It also shows the significance level of the test, and whether we accept or reject the null hypothesis. Interestingly, for all solvers with the exception of MiniSAT, we reject the null hypothesis. It is not clear why MiniSAT does not exhibit the same pattern as the other solvers considered.

Solver	% $H3_a$ holds	p-value	Reject NULL
MiniSAT	50.0%	0.954	ACCEPT
Glucose	61.0%	0.001	REJECT
Lingeling	63.4%	0.001	REJECT
pLingeling	63.6%	0.001	REJECT
SWDiA5BY	63.4%	0.001	REJECT

Table 4: Shows the results of the hypothesis test H3. For each solver the % of randomly selected pairs of instances where $H3_a$ holds is given, along with the p-value of the hypothesis test.

3.4   Predicting performance

Using a standard linear regression with Equation 2 we can predict the solution time of individual instances on specific solvers for previously untested hardware, based on their performance on our test machines. This model results in an $R^2 > 0.73$ for 80% of the instances considered, across all solvers. The $\oplus$ notation here denotes that not only were each of the individual factors considered, but all interaction terms of the three factors as well.

$$time_{is} \sim CPUSpeed \oplus CPUCores \oplus RAM \qquad (2)$$

From the preliminary data, Lingeling was the most predictable solver, where 80% of instances had an $R^2 > 0.87$. SWDiA5BY and MiniSAT were the least predictable solvers, where in both cases 80% of instances had an $R^2 > 0.80$. This

indicates that Lingeling's performance is less determined by factors not included in our model, such as RAM speed and cache sizes. Conversely, the performance of MiniSAT and SWDiA5BY may be effected by these absent parameters.

To test these results, we performed a k-fold cross validation with $k = 5$. For each instance and solver combination we randomly selected 80% of the machines as a training set to predict the execution time of the remaining machines. We repeated this ten times for each solver to mitigate the issue of different prediction error depending on which machines were assigned to the training and test sets. Table 5 shows the predictability of the five considered solvers according to their median absolute error, maximum error and 80% confidence level. This data confirms that predictions made for the Lingeling solver are most accurate, and that predictions made by MiniSAT are least accurate. However, SWDiA5BY predictions were relatively accurate, while pLingeling predictions were significantly less accurate.

This is indicated by the increased median error and 80% confidence level. The difference between MiniSAT and pLingeling in this case is very small, seven seconds for the median error and 30 seconds for the 80% confidence level. One possible explanation is that machines were assigned to folds randomly, as such it is possible that the assignment for the MiniSAT instances was less predictive than the one used for pLingeling.

Solver	Median error (s)	Maximum error (s)	80% error (s)
MiniSAT	104	163894	1087
Glucose	53	1939631	665
Lingeling	46	1218864	555
pLingeling	89	2116591	999
SWDiA5BY	50	111689	641

Table 5: The predictability of different solvers, showing median, maximum and the 80% confidence level of predictions, for each of the considered solvers.

Due to the random sampling performed, results vary on depending on which machines are assigned to each fold for the cross validation. Had we set $k = 21$ this would not have been the case. However, one of our goals is to find a small subset of machines that can be used to predict solution times across a wide range of hardware, as such we were interested in minimising the training set. Figures 2a and 2b show the results of the cross validated predictions for Lingeling and MiniSAT respectively. These solvers are highlighted as they were the most and least predictable solvers respecitvely.

While no obvious trend exists between the quality of the predictions, and any of the 34 considered instance characteristics, there does exist a trend between the quality of the predictions, and the average execution time for the instances. Figure 3 shows the trend of decreasing accuracy against average execution time. It also shows that the variability of the predictions increases as execution time
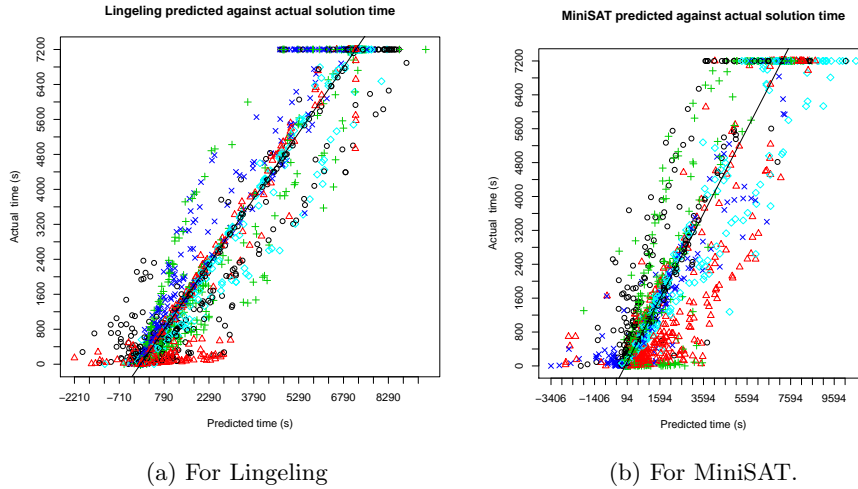
(a) For Lingeling

(b) For MiniSAT.

Fig. 2: Figures showing the predicted solution times against actual solution times for the Lingeling and MiniSAT solvers for a single repetition of the k-fold cross validation. Approximately 5% of the outlier data points are omitted from these plots to improve interpretability. The different colours represent the individual folds. The black line represents perfect predictions.

increases. The most likely explanation for this is the number of timeouts affecting both the average execution time, and the accuracy of predictions. The more timeouts per instance, the less accurate the predictions. Approximately 14% of instances in this dataset timeout, of those instances, 42% were on UNSAT instances and 58% were on SAT instances. Considering the UNSAT instances show more accurate predictions over the increasing range of execution time and have fewer timeouts, when compared to SAT instances, this would support our conjecture.

4 Conclusion

In this work we have explored the relationship between different classes of instance, and their solution times on differently specified benchmarking machines and solvers. We have shown that the solution time for a specific instance varies greatly across different machines, in ways that are not completely predictable when considering characteristics such as CPU speed, RAM size and cache size. We have further shown that the impact of each of these factors on the solution time of an instance depends on the structure of the instance. In some cases this structure is characterised by graph theoretical concepts, and others use SAT specific concepts such as the clause-variable ratio.

We have presented a model that is able to predict the solution time of SAT instances across a diverse set of hardware with relatively high accuracy. We have
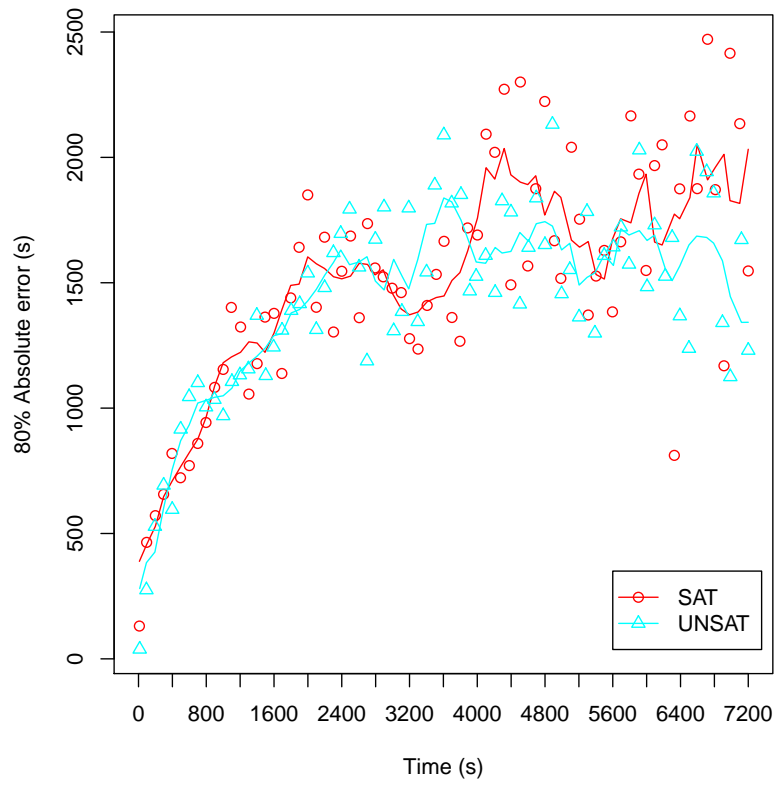
Fig. 3: The correlation between the 80% confidence level for predictions, and the average solution time of the instances. The lines represent the moving average over a window of five data, each data point represents a 10 second time interval.

also identified that, of the observed solvers, Lingeling has the most predictable performance and UNSAT instances are generally more predictable than SAT instances. In addition to the results presented here, we have found strong correlations between the predictability of instance performance and max(reused), max(clause), mean(clause), max(var) as well as others. These results are omitted from this work for the sake of brevity.

While we have not been able to produce a model that completely explains the variability in solution time when varying the machines, we have been able to explain a large amount of it. The remaining variability is likely to be in the factors that were only partially available in our dataset (e.g. cache sizes, RAM speeds, etc), however we also speculate that cache replacement policies will be a determining factor in solution times of specific instances.

5 Future Work

The next steps for this work are to find more details on the machines used, as previously mentioned it was not possible to gather all levels of cache size for all machines, similarly RAM channels and speeds were missing in some cases. We hope this will partially solve the issue of un-defined variation. However, if this does not complete the model we are planning on exploring the relationship between cache replacement policies and performance. We feel this may be what is missing when we compare machines with very different specifications, that perform similarly — for example, the Pentium M and Pentium D machines as described above.

While it was important to perform a fair random assignment for the k-fold cross validation, we also intend to perform a "cherry-picked" version where we select individual machines based on their parameters as the training set. In doing so we hope to maximise the quality of our predictions while minimising our training set.

In this work we chose to perform the analysis with a standard linear regression, and it has provided some strong results. However, it is not the only analysis technique suitable for this area. Random forests [10], non linear regression and Bayesian inference are examples of techniques that have been used with varying levels of success to predict the solution time of sets of instances on fixed hardware. While we are focusing on predicting the solution times on heterogeneous hardware from known solution times, these techniques could also apply.

Finally, we are looking into tuning SAT solvers based on the hardware that is being used to solve an instance. It has already been established that some solvers perform better on different hardware, and different classes of instance. While there has been significant work in selecting/tuning an algorithm based on instance characteristics [11–13], there has been relatively little work in selecting/tuning an algorithm based on the hardware being used.

6 Related Work

The related work falls into two categories. First, there is significant work on predicting SAT solver performance on sets of instances, on fixed hardware. Second, there has been significant work outside of the SAT community in the prediction of program runtime on diverse hardware.

In 2010, Kadioglu et al presented a method for instance specific algorithm configuration (ISAC) [11]. This work, which encompasses algorithm selection and tuning, focused on characteristics of the input instance to tune the selected algorithm and thus increase performance. They utilise g-means to cluster the instances, working on the premise that instances that are clustered together will behave similarly when solving.

In 2012, Malitsky and Sellmann presented the idea of using ISAC for the construction of portfolio based SAT solvers [12]. They compare this performance against that of portfolio based solvers such as SATzilla [13] and show that in many cases ISAC outperforms them.

In 2012, an update on SATzilla was published [13] which presented the utilisation of cost-sensitive classification models. In this version of SATzilla, feature computation is limited to 90 CPU seconds. This is a particularly interesting technique when considering community based algorithms such as CNM [14], as the runtime of these algorithms can be prohibitively high, when compared to a relatively low runtime of a solver on certain types of instance.

In 2004, Marin and Mellor-Crummey presented a work on predicting parallel application performance across different architectures using parametrised models [15]. In this work the authors utilise instrumented code to predict execution times, considering factors such as the execution count of individual sequential sections of code and memory latency on the target architecture. This technique requires measuring specific hardware factors (such as the cost of an L1 cache miss) at runtime to enable accurate predictions.

In 2005, Yang et al presented a work on the prediction of parallel application performance using partial execution [16]. This technique utilises observation based techniques and as such has no need to consider features of the platform, such as CPU speed, memory size, etc. The advantage of this technique is that it can be highly accurate for certain types of program, it is unclear whether SAT solvers fall into this category. However, one drawback is that for each new platform a representative set of sample applications must be ran in order for the predictions to be accurate. This makes the technique more valid for large scale parallel computing clusters that utilise multiple instances of similarly specified hardware.

In 2006, Hoste et al presented a work on predicting application performance based on program simplicity [17]. This work utilises the SPEC CPU2000 [18] results on 36 machines to determine if a correlation exists between the proposed simplicity metrics and the speedup rates published in SPEC CPU 2000. Their results show an improved worst-case and average correlation coefficient when compared to current practice.

In 2006, Lee and Brooks present a work on predicting application performance on the Turandot simulator [19]. They consider 12 architectural parameters including number of general purpose registers, sizes of L1 and L2 cache, and memory and L2 latency. The use of a simulator allowed high levels of control on features such as pipeline depth and memory latency in cycles. They found that application specific models were most predictive of performance prediction.

References

1. Augusto Oliveira, Jean-Christophe Petkovich, Thomas Reidemeister, and Sebastian Fischmeister. Datamill: Rigorous performance evaluation made easy. In *Proc. of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 137–149, Prague, Czech Republic, April 2013.
2. Zack Newsham. Maching parameters experimental data. http://satbench.uwaterloo.ca/download/55/1, 2014. Accessed: 2015-01-10.
3. Niklas Een and Niklas Sörensson. Minisat: A SAT solver with conflict-clause minimization. *SAT*, 5, 2005.
4. Gilles Audemard and Laurent Simon. Glucose: a solver that predicts learnt clauses quality. 2009.
5. A Biere. Lingeling. *SAT Race*, 2010.
6. A Biere. Plingeling: solver description. *SAT Race*, 2010.
7. Augusto Oliveira, Jean-Christophe Petkovich, and Sebastian Fischmeister. How Much Does Memory Layout Impact Performance? A Wide Study. In *Proceedings of the International Workshop on Reproducible Research Methodologies (REPRODUCE)*, page 23–28, Orlando, USA, Febuary 2014.
8. Rodrigo Castaño and José M Castaño. Propositional satisfiability (sat) as a language problem. In *XVII Congreso Argentino de Ciencias de la Computación*, 2011.
9. et al Alexe. Maximal biclique enumeration implementation. http://genome.cs.iastate.edu/supertree/download/biclique/README.html, 2004. Accessed: 2015-01-10.
10. Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.
11. Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. Isac-instance-specific algorithm configuration. In *ECAI*, volume 215, pages 751–756, 2010.
12. Yuri Malitsky and Meinolf Sellmann. Instance-specific algorithm configuration as a method for non-model-based portfolio generation. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 244–259. Springer, 2012.
13. Lin Xu, Frank Hutter, Jonathan Shen, Holger H Hoos, and Kevin Leyton-Brown. Satzilla2012: improved algorithm selection based on cost-sensitive classification models. *Balint et al.(Balint et al., 2012a)*, pages 57–58, 2012.
14. Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.
15. Gabriel Marin and John Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. In *ACM SIGMETRICS Performance Evaluation Review*, volume 32, pages 2–13. ACM, 2004.
16. Leo T Yang, Xiaosong Ma, and Frank Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 40–40. IEEE, 2005.

17. Kenneth Hoste, Aashish Phansalkar, Lieven Eeckhout, Andy Georges, Lizy K John, and Koen De Bosschere. Performance prediction based on inherent program similarity. In *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pages 114–122. ACM, 2006.
18. John L Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33(7):28–35, 2000.
19. Mayan Moudgill, John-David Wellman, and Jaime H Moreno. Environment for powerpc microarchitecture exploration. *Micro, IEEE*, 19(3):15–25, 1999.

Appendix A

A list of all machines used in the various experiments in this paper.

Number	CPU	Cores	CPU speed	Cache (L1 i/d + L2 + L3)	RAM amt	RAM speed
1	Intel Core i7 i686	4	3400	32/32 + 256 + 8192	8266580	0
2	Intel Pentium M i686	1	1695	0/0 + 256 + 0	902287	0
3	Intel Pentium 4 i686	2	2992	0/16 + 2048 + 0	894177	533
4	VIA Nano X2 i686	2	1733	128/128 + 2048 + 0	1814036	1066
5	Intel Pentium 4 i686	2	3200	0/0 + 512 + 0	1000263	0
6	Intel Pentium 4 i686	1	1595	0/0 + 256 + 0	254781	0
7	Intel Pentium 4 i686	2	2998	0/16 + 1024 + 0	893347	0
8	Intel Pentium 4 i686	1	1595	0/0 + 256 + 0	514119	133
9	Intel Pentium 4 i686	2	2992	0/0 + 1024 + 0	894269	0
10	Intel Pentium 4 i686	2	3200	0/0 + 512 + 0	1000540	0
11	Intel Pentium 4 i686	2	2793	64/64 + 2048 + 0	902461	0
12	Intel Pentium 4 i686	2	1614	0/0 + 256 + 0	894269	0
13	Intel Pentium 4 i686	2	1600	0/0 + 256 + 0	242851	0
14	Intel Pentium 4 i686	2	3198	0/0 + 512 + 0	894269	0
15	AMD Athlon XP i686	1	1111	64/64 + 256 + 0	514199	0
16	Intel Pentium D i686	2	2993	0/0 + 1024 + 0	2076180	0
17	Intel Pentium 4 i686	2	3200	0/16 + 512 +0	894269	0
18	Intel Pentium 4 i686	2	3192	0/0 + 512 + 0	505661	0
19	Intel Xeon x86_64	2	3000	0/0 + 4098 + 0	2831155	0
20	Intel Pentium 4 i686	2	3200	0/0 + 512 + 0	2076180	0
21	Intel Core i7 x86_64	8	3401	128/128 + 1024 + 0	8095006	0
22	ARM Rev 10 armv71	4	1988	0/0 + 1024 + 0	896563	0
23	ARM Rev 10 armv71	4	1988	0/0 + 1024 + 0	896563	0
24	ARM Rev 10 armv71	4	1988	0/0 + 1024 + 0	896563	0
25	ARM Rev 10 armv71	4	1988	0/0 + 1024 + 0	896563	0
26	Intel Core i5 x86_64	4	3291	128/128 + 1024 + 0	8095006	0
27	Intel Core i7 64	4	3400	32/32 + 256 + 8192	8388608	0
28	AMD Athlon	1	757	64/64 + 256 + 0	773079	0

List of machine specifications used for the varied parameters trial. Machines
2-28 were ran through the DataMill platform (0's indicate we were unable to
determine the value of this property for a specific machine)

Appendix B

The full description of all features considered in this paper.

Variable Name	Definition
vars	The number of variables in the formula
weight	The difference in the number of true/false literals.
CO	The set of communities
Q	The quality (Q) of the community structure
$max(\text{var})$	The maximum number of times a variable appears
$mean(\text{var})$	The average number of times a variable appears
$min(\text{com})$	The size of the smallest community
$mean(\text{com})$	The average community size
$max(\text{com})$	The size of the largest community
$sd(\text{com})$	The standard deviation of the community sizes
$min(\text{inter})$	The minimum number of inter-community edges from a single community
$max(\text{inter})$	The maximum number of inter-community edges from a single community
$mean(\text{inter})$	The average number of inter-community edges from a single community
$sd(\text{inter})$	The standard deviation of the number of inter-community edges from a single community
$min(\text{intra})$	The minimum number of intra-community edges in a single community
$max(\text{intra})$	The maximum number of intra-community edges in a single community
$mean(\text{intra})$	The average number of intra-community edges in a single community
$sd(\text{intra})$	The standard deviation number of intra-community edges in a single community
$edgeratio$	The overall ratio of inter/intra community edges
$max(\text{edgeratio})$	The maximum ratio of inter/intra community edges for a single community
$min(\text{edgeratio})$	The minimum ratio of inter/intra community edges for a single community
$mean(\text{edgeratio})$	The average ratio of inter/intra community edges for a single community
$sd(\text{edgeratio})$	The standard deviation ratio of inter/intra community edges for a single community
UE	The set of unique edges in the graph
TE	The set of all edges (counting degrees) in the graph
CL	The set of all clauses
VUC	The set of clauses using distinct variables
$max(\text{clause})$	The length of the longest clause
$mean(\text{clause})$	The average clause length
$max(\text{reused})$	The maximum times a clause with the same variables is reused
$min(\text{reused})$	The minimum times a clause with the same variables is reused
$mean(\text{reused})$	The average times a clause with the same variables is reused
CVR	The ratio of clauses to variables
TVR	The ratio of total clauses to clauses using unique sets of variables

Table 6: Description of all instance characteristics used

Appendix C

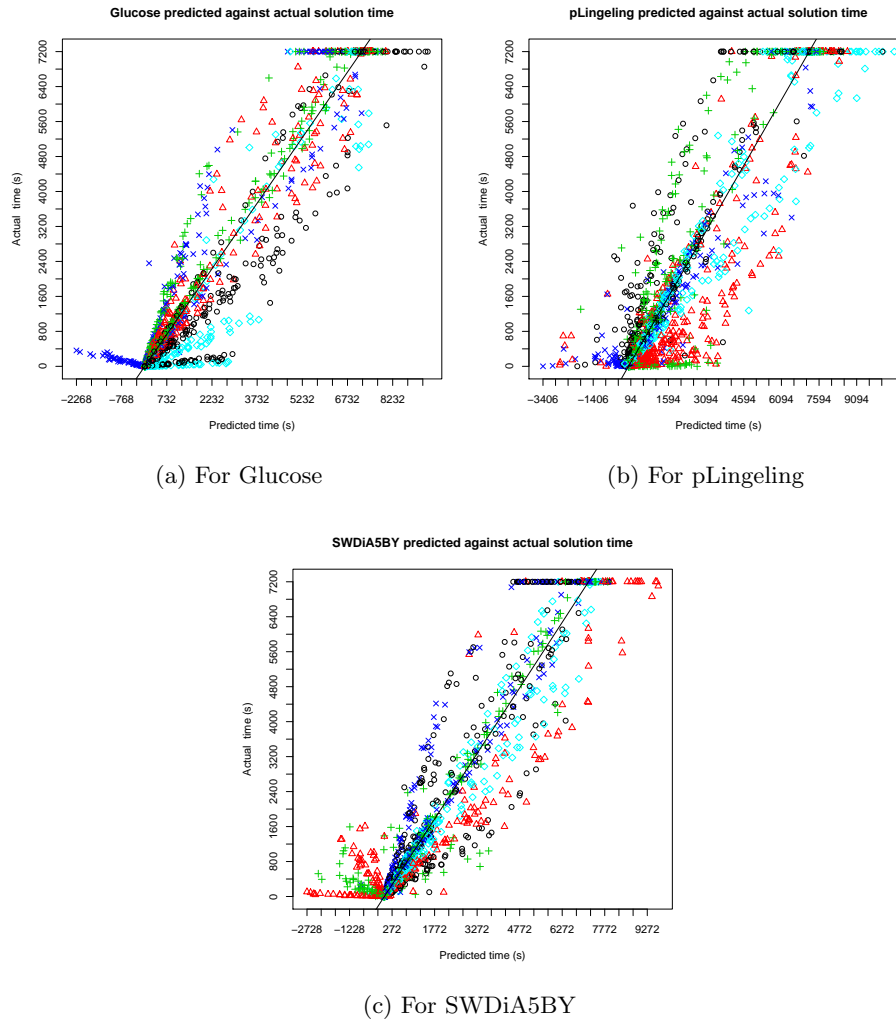(a) For Glucose

(b) For pLingeling

(c) For SWDiA5BY

Fig. 4: Figures showing the predicted solution times against actual solution times for the Glucose, pLingeling and SWDiA5BY solvers for a single repetition of the k-fold cross validation. Approximately 5% of the outlier data points are omitted from these plots to improve interpretability. The different colours represent the individual folds. The black line represents perfect predictions.