Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Statistical Regimes and Runtime Prediction

Barry Hurley and **Barry O'Sullivan**
Insight Centre for Data Analytics, University College Cork, Ireland
{barry.hurley,barry.osullivan}@insight-centre.org

Abstract

The last decade has seen a growing interest in solver portfolios, automated solver configuration, and runtime prediction methods. At their core, these methods rely on a deterministic, consistent behaviour from the underlying algorithms and solvers. However, modern state-of-the-art solvers have elements of stochasticity built in such as randomised variable and value selection, tie-breaking, and randomised restarting. Such features can elicit dramatic variations in the overall performance between repeated runs of the solver, often by several orders of magnitude. Despite the success of the aforementioned fields, such performance variations in the underlying solvers have largely been ignored. Supported by a large-scale empirical study employing many years of industrial SAT Competition instances including repeated runs, we present statistical and empirical evidence that such a performance variation phenomenon necessitates a change in the evaluation of portfolio, runtime prediction, and automated configuration methods. In addition, we demonstrate that this phenomenon can have a significant impact on empirical solver competitions. Specifically, we show that the top three solvers from the 2014 SAT Competition could have been ranked in any permutation. These findings demonstrate the need for more statistically well-founded regimes in empirical evaluations.

1 Introduction

Modern combinatorial search solvers contain many elements that are stochastic in nature, such as randomised variable and value selection, tie breaking in heuristics, and random restarting. These elements add a degree of robustness to the solver. Nevertheless, this stochasticity results in variations in runtime between repeated runs of the solver, a fact which is often not accounted for by users and practitioners in related fields. This paper illustrates the consequences that such assumptions can have when comparing solver performance.

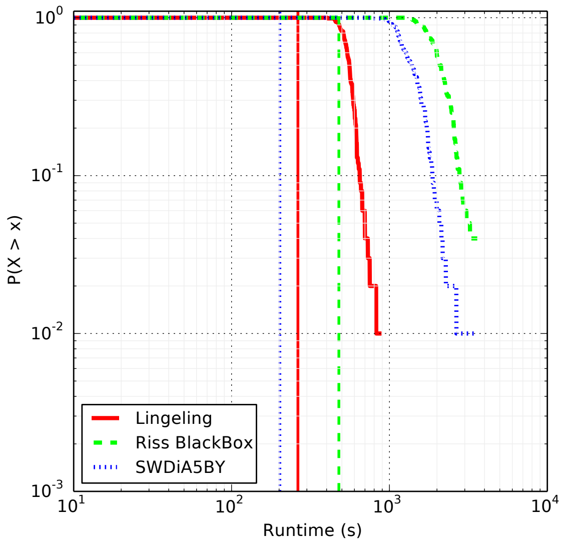
One such field is that of solver portfolios [Gomes and Selman, 2001; Kotthoff, 2014], which have consolidated and advanced the state-of-the-art across a broad range of problem domains such as Constraint Programming (CP) [O'Mahony *et al.*, 2008; Hurley *et al.*, 2014], Satisfiability (SAT) [Xu *et al.*, 2008; Kadioglu *et al.*, 2010], MaxSAT [Ansótegui *et al.*, 2014], Planning [Seipp *et al.*, 2015], and many more. Their effectiveness relies on the ability to identify the best solver for each individual instance. At its core, training a portfolio involves collecting the runtime performance of each solver on a collection of benchmarks. From this, machine learning techniques are used to choose amongst the solvers based on instance features. Existing work in this area has only considered a single sample of the solver runtime for each instance, and makes the assumption that the behaviour is consistent between repeated runs of the solver. However, as we will see, the performance of a solver can vary by many orders of magnitude between repeated runs. This behaviour is related to heavy-tailed distributions discovered by [Gomes *et al.*, 2000].

In a related field, automated solver configuration tools [Hutter *et al.*, 2011; Ansótegui *et al.*, 2009; Fitzgerald *et al.*, 2014] have successfully been able to tune solvers to surpass their default configuration. Their effectiveness relies on obtaining consistent, representative performance data for a given parameterisation on a sample of input instances. Crucially, these methods will not typically make allowances for any stochastic behaviour in the underlying solvers.

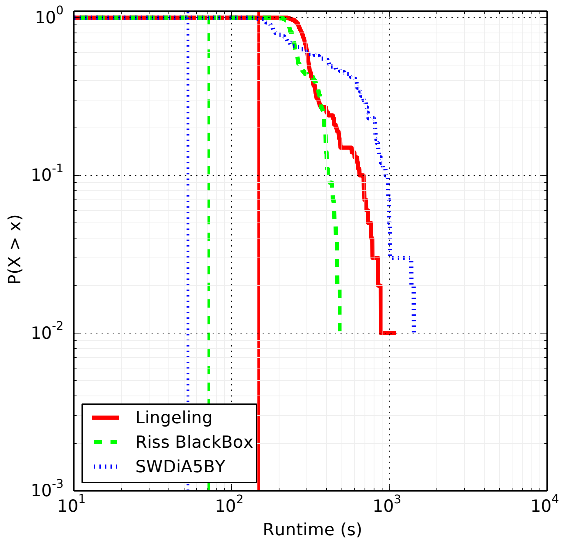
Related to the above is the task of solver runtime prediction [Hutter *et al.*, 2014]. This focuses on building an empirical hardness model to predict the performance of a solver based on instance specific features. This paper will demonstrate that state-of-the-art runtime prediction methods do not account for the stochastic aspect of solver runtime and as such can have a substantial effect on their accuracy.

Thus, the contributions of this paper are as follows:

- we demonstrate that dramatic runtime variations occur in practice and can alter the outcome of empirical comparisons, such as the annual SAT competitions,
- we show that runtime variation needs to be accounted for in fields such as solver runtime prediction, portfolios, and automated configuration,
- we conduct a large-scale empirical evaluation on over 10 years of industrial SAT instances consuming over 600 weeks of CPU-time, and
- we provide a statistical analysis that can be applied to compare the performance of solvers from a runtime distribution perspective.



(a) SAT Competition 2014 instance UCG-15-10p0.cnf



(b) SAT Competition 2014 instance q_query_3_L90_coli.sat.cnf

Figure 1: Log-log plots showing the empirical runtime distribution of two instances from the application category at the 2014 SAT Competition. The three solvers are the top-ranked solvers from the competition and the vertical lines mark their observed runtime during the competition.

2 Runtime Variations in Practice

This section illustrates a deficiency of empirical comparisons based on a single run. Figure 1 shows the runtime distribution of the top three solvers [Biere, 2014; Alfonso and Manthey, 2014; Oh, 2014] on two industrial instances from the 2014 SAT Competition. Vertical lines mark the solver's observed runtime during the competition, and the other lines plot the survival function of the empirical runtime distribution over 100 runs. In Figure 1a, the solver SWDiA5BY solved the instance faster than Lingeling during the competition, however we can clearly deduce from the runtime distribution that over repeated runs there is a larger probability of SWDiA5BY having a longer runtime than Lingeling. In this instance, the runtime distribution shows a clear ranking of the solvers: Lingeling $\prec$ SWDiA5BY $\prec$ Riss BlackBox. In Figure 1b the distinction between solvers is not as pronounced. Interestingly, in the competition, SWDiA5BY is ranked as the fastest of the three solvers on this instance, however when you consider its runtime distribution, it is most likely to take the longest.

This exemplifies a critical flaw in this type of empirical evaluation that is often performed when comparing solvers, only a single sample of the solver's runtime is taken on each benchmark instance. The current evaluation metric used in the SAT Competition purely considers the number of instances solved within a specified timeout, and only uses runtimes in the case of ties. However, the runtime distribution can often span over the timeout value, thus affecting the number of instances solved. Section 4 will demonstrate the ramifications of this, showing that these three solvers could have been ranked in any permutation. However, first, Section 3 will present a more holistic, statistically-founded methodology which is lacking in current comparisons.

3 Expected Performance

We can model the performance of a solver on a set of benchmark instances as a random variable, doing so will allow us to define bounds on the possible outcomes. Let p_i be the probability that instance i is solved within a given timeout. p_i will be simply 0.0 or 1.0 if a single sample runtime is taken, or a continuous value if a solver is run multiple times on the same instance. Naturally, the accuracy of p_i is proportional to the number of repeated runs performed.

Next, let X_i be a discrete random variable which takes the value 1 if instance i is solved within the timeout, 0 otherwise. The expected value of X_i is p_i:

$$E[X_i] = \Pr(X_i = 0) \times 0 + \Pr(X_i = 1) \times 1 \equiv p_i \tag{1}$$

We can define a bound on the variation of X_i by taking the critical value for $p_i = 0.5$; thus the standard deviation of X_i can be bounded by $\frac{1}{2}$:

$$\begin{aligned} \sigma^2(X_i) &= E[X_i^2] - (E[X_i])^2 \\ &= E[X_i] - p_i^2 \\ &= p_i - p_i^2 \\ &= p_i(1-p_i) \\ &\leq \tfrac{1}{4} \quad \text{for any } p_i \end{aligned} \tag{2}$$

$$\sigma(X_i) = \sqrt{p_i(1-p_i)} \leq \frac{1}{2} \tag{3}$$

If the benchmark set contains k instances, then the success rate S, or ratio of instances solved within the timeout, is simply the mean of the X_i variables over the k instances. This allows the definition of the expected value of S as being the mean over the p_i values:

$$E[S] = \frac{\sum_{i=1}^{k} E[X_i]}{k} = \frac{\sum_{i=1}^{k} p_i}{k} \tag{4}$$

Finally, bounds on the variation and standard deviation of S can be defined:

$$\begin{aligned}\sigma^2(S) = &\ \sigma^2\left(\frac{\sum_{i=1}^k X_i}{k}\right) = \frac{1}{k^2}\left(\sum_{i=1}^k \sigma^2(X_i)\right) \\ = &\ \frac{1}{k^2}\left(\sum_{i=1}^k p_i(1-p_i)\right) \le \frac{k}{4k^2} = \frac{1}{4k}\end{aligned} \tag{5}$$

$$\sigma(S) \le \frac{1}{2\sqrt{k}} \tag{6}$$

Importantly, this means that even over repeated runs, the variation in the total number of instances solved is bounded by a function of the number of instances. This fact should be of great consequence to the designers of empirical comparisons.

4 SAT Competition 2014

This section will project the analysis of Section 3 onto one annual empirical solver comparison. Consider the top three solvers from the application category at 2014 SAT Competition, namely, *Lingeling* [Biere, 2014], *SWDiA5BY5by* [Oh, 2014], and *Riss BlackBox* [Alfonso and Manthey, 2014] which solved 231, 228, and 226 of the 300 instances in this category, respectively. Note that, like many other empirical competitions, only a single sample of the solver's runtime is taken during the evaluation and the scoring metric is purely based on the number of instances solved within the specified timeout. Using the equations of Section 3, we can postulate on possible outcomes and give upper-bounds on the variation that could occur in such empirical evaluations, this is presented in Table 1.

Given that the performance difference amongst the top-ranked solvers is so narrow, it is not unreasonable to suspect that the outcome could have been different if the competition was re-run. To assess the potential outcomes, each of the three solvers above were re-run 100 times on each of the 300 instances from the industrial/application category at the SAT Competition 2014. Since Lingeling is the only one of these solvers to support the ability to pass a random seed as input, we emulate the functionality across all solvers simply by shuffling the input instance: randomly renaming the variables, shuffling their order in each clause, and reordering the clauses. This preserves the same structure and semantics of the original instance but may result in the solver taking different paths in the solving process across runs. Note that it is common in the SAT Competition to re-use instances from previous years by performing this type of shuffling [Le Berre and Simon, 2004].

Table 1: Number of instances solved out of 300 by the top-3 solvers in the application category of the 2014 SAT Competition along with the upper-bound on the deviation from these values if the competition was rerun.

Solver	#Solved	S	$\overline{\delta(S)}$	95% bounds
Lingeling	231	0.770	0.0289	214..248
SWDiA5BY5by	228	0.760	0.0289	211..245
Riss BlackBox	226	0.753	0.0289	209..243

Performance data was collected on a cluster of Intel Xeon E5430 2.66Ghz processors running CentOS 6. The solvers were limited to 1 hour and 2GB of RAM per instance. All times are reported in CPU-time. Note that these experiments are run on different hardware to that used in the SAT Competition and used a shorter timeout due to the increased computational complexity. In total, 322 CPU-weeks were used to record the data.

Considering the observed runtime distributions across 100 runs, together with the analysis from Section 3, we can quantify more statistically well-founded outcomes to the SAT Competition; these are presented in Table 2. Additionally, we can derive 95% confidence bounds on the expected success rate if we assume it to be normally distributed; which we will see to be reasonable in Figure 2. Significantly, the 95% confidence bounds of the solvers overlap, and as such, any permutation of the ordering could be possible.

Table 2: Expected success rates over multiple runs on 300 application instances from the SAT Competition 2014.

Solver	$E[S]$	$\delta(S)$	95% bounds
Lingeling	0.553	3.6	159..173
Riss BlackBox	0.530	3.3	152..165
SWDiA5BY5by	0.571	2.6	166..176

Figure 2 augments these results by visualising the distribution of the total number of instances solved by each solver across the runs. Their frequencies appear normally distributed but additional samples would be needed to verify this. Ordering by probability mass would rank the solvers SWDiA5BY $\prec$ Lingeling $\prec$ Riss BlackBox, consistent with the ranking from Table 2. Interestingly, this is a different ranking to that of the official competition.

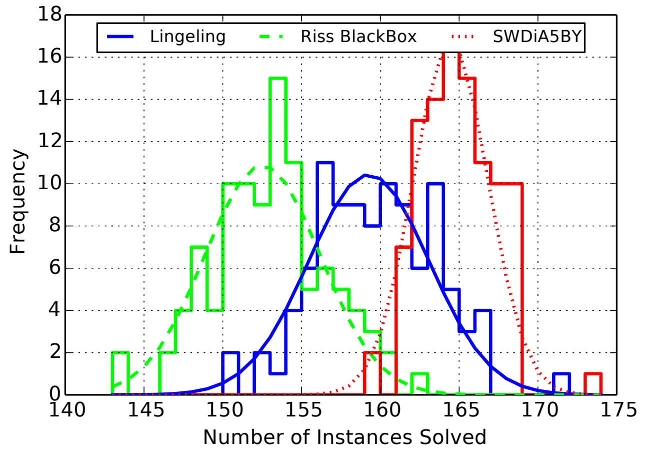


Figure 2: A histogram of the number of instances solved across 100 runs on 300 instances from the application category of the SAT Competition 2014. The three solvers are the top-ranked solvers from the competition.

Table 3 presents some more detailed statistics about the

Table 3: Summary of the runtime variations of three solvers on a set of sample instances from the 2014 SAT Competition. The 'Comp.' column contains the runtime observed in the competition, AUC is the area under the survival function for the empirical distribution. Bold entries mark instances where there is at least one order of magnitude between the best and worst observed runtime for the solver.

	Instance Name	Solver	#runs	Median	Mean	Std.dev	Min	Max	Comp.	AUC
15	**006-23-80.cnf**	**Lingeling**	**100**	**3600.0**	**3509.6**	**438.3**	**63.5**	**3600.0**	**5000.0**	**3509.6**
15	006-23-80.cnf	Riss BlackBox	100	3600.0	3600.0	0.0	3600.0	3600.0	5000.0	3600.0
15	006-23-80.cnf	SWDiA5BY	100	3600.0	3600.0	0.0	3600.0	3600.0	5000.0	3600.0
21	**008-80-4.cnf**	**Lingeling**	**100**	**1274.4**	**1752.1**	**1264.6**	**142.0**	**3600.0**	**621.0**	**1752.5**
21	**008-80-4.cnf**	**Riss BlackBox**	**100**	**808.8**	**818.9**	**382.3**	**85.3**	**3600.0**	**794.0**	**819.4**
21	**008-80-4.cnf**	**SWDiA5BY**	**100**	**209.7**	**202.7**	**78.2**	**10.5**	**359.6**	**85.4**	**203.2**
53	**6s168-opt.cnf**	**Lingeling**	**100**	**260.5**	**473.4**	**622.6**	**26.6**	**3600.0**	**370.8**	**473.9**
53	6s168-opt.cnf	Riss BlackBox	100	13.1	14.2	4.5	7.4	36.8	17.5	14.6
53	6s168-opt.cnf	SWDiA5BY	100	83.9	87.0	21.8	51.9	162.5	56.7	87.4
117	UCG-15-10p0.cnf	Lingeling	100	556.7	562.9	75.0	404.6	863.9	263.2	563.4
117	UCG-15-10p0.cnf	Riss BlackBox	100	2117.1	2165.9	510.9	1290.1	3600.0	478.4	2166.5
117	UCG-15-10p0.cnf	SWDiA5BY	100	1385.2	1445.4	402.9	757.4	3600.0	202.4	1445.9
258	q_query_3_... li.sat.cnf	Lingeling	100	307.8	379.6	165.1	226.7	1071.9	148.6	380.0
258	q_query_3_... li.sat.cnf	Riss BlackBox	100	268.7	308.5	78.5	198.2	515.1	71.5	308.9
258	**q_query_3_... li.sat.cnf**	**SWDiA5BY**	**100**	**422.3**	**507.9**	**320.6**	**141.3**	**1487.7**	**52.9**	**508.4**
265	rpoc_xits_15_SAT.cnf	Lingeling	100	7.3	5.5	2.9	1.6	11.9	1.9	6.0
265	**rpoc_xits_15_SAT.cnf**	**Riss BlackBox**	**100**	**12.1**	**59.7**	**359.1**	**4.9**	**3600.0**	**3.6**	**60.2**
265	rpoc_xits_15_SAT.cnf	SWDiA5BY	100	5.3	5.8	2.0	2.1	11.7	1.1	6.3
271	**stable-400... 140011.cnf**	**Lingeling**	**100**	**3600.0**	**3458.7**	**568.3**	**125.6**	**3600.0**	**5000.0**	**3458.7**
271	**stable-400... 140011.cnf**	**Riss BlackBox**	**100**	**3600.0**	**3482.0**	**532.4**	**133.6**	**3600.0**	**5000.0**	**3482.0**
271	**stable-400... 140011.cnf**	**SWDiA5BY**	**100**	**3600.0**	**2577.2**	**1346.1**	**78.3**	**3600.0**	**5000.0**	**2577.4**
287	**vmpc_32.re... 5-1919.cnf**	**Lingeling**	**100**	**3600.0**	**2929.7**	**1113.6**	**19.3**	**3600.0**	**5000.0**	**2929.8**
287	**vmpc_32.re... 5-1919.cnf**	**Riss BlackBox**	**100**	**1445.9**	**1831.9**	**1365.0**	**17.2**	**3600.0**	**1931.7**	**1832.2**
287	**vmpc_32.re... 5-1919.cnf**	**SWDiA5BY**	**100**	**2533.8**	**2187.1**	**1488.7**	**3.1**	**3600.0**	**149.5**	**2187.4**

performance of the three solvers, only a sample of the 300 instances is presented due to space constraints. We list the runtime which was observed in the SAT Competition under the 'Comp.' column. Notice that for numerous instances where a solver recorded a timeout in the SAT Competition we also observe many timeouts across the runs, but critically, the solver may also solve the instance very quickly.

We observe dramatic variations in runtime, often more than three orders of magnitude between a solver's best and worse time on an instance. In numerous cases, on a single instance, a solver will timeout on a large proportion of the runs but conversely may solve the instance very quickly in other cases. This could be the critical difference between solved or unsolved in the competition setting. Given that the difference amongst the top ranked solvers in the SAT Competitions is typically only a handful of instances, this constitutes a significant observation.

5 State-of-the-art Runtime Prediction

The ability to predict the runtime of a solver is interesting from a number of perspectives. Solver designers can make use of it as an informative analytical tool and it frequently serves as a central component in solver portfolios and configuration tools. Often, these tools base their decisions on underlying runtime prediction models. The prediction target is the runtime of the solver on an instance, which has conventionally been collected by recording a single runtime. However this approach is flawed, since in reality we are dealing with solvers that exhibit a highly variable runtime distribution, a single value is simply not representative. This section highlights the detrimental effects that this fundamental flaw can have on state-of-the-art runtime prediction methods.

5.1 Log Transformation

The typical practice when predicting solver runtime is to log-transform the runtime in seconds [Hutter *et al.*, 2014]. Error calculations on this, like root-mean-squared error and mean absolute error, will over-penalise runtimes of less than one second. Differences between runtimes of 0.01 seconds and 0.1 seconds are valued to the same extent as the difference between 10 and 100 seconds. These are all an order of magnitude different, but we argue that the first order of magnitude for runtimes should range from $[0..9)$ seconds, the second from $[9..99)$, and so on. Thus, the runtimes should be transformed as $\log(1 + \text{runtime})$. This additionally eliminates the need to censor runtimes of zero seconds which would have been needed when performing the log transformation. For the predictions in this section we will use the adjusted log-transformed runtime, $\log_{10}(1 + \text{runtime})$.

Table 4: Samples of the runtime variations of MiniSat on a set of sample industrial category instances from the SAT Competitions, Races, and Challenges between 2002 and 2011. The AIJ'14 column contains the runtime observed in [Hutter *et al.*, 2014], AUC is the area under the survival function for the empirical distribution.

Instance Name	#runs	Median	Mean	Std.dev	Min	Max	AIJ'14	AUC
dated-10-13-s.cnf	100	82.0	337.4	675.3	7.5	3600.0	12.6	337.9
fvp-unsat.shuffled.cnf	100	94.4	1294.5	1642.6	8.0	3600.0	3600.0	1294.8
grieu-vmpc-s05-34.cnf	100	3600.0	2963.2	1189.9	9.6	3600.0	3600.0	2963.3
gss-23-s100.cnf	100	3600.0	3542.7	310.8	1243.9	3600.0	3600.0	3542.7
k2fix_gr_rcs_w8.shuffled.cnf	100	3600.0	3394.9	818.5	5.6	3600.0	3600.0	3394.9
mizh-md5-47-4.cnf	100	200.3	283.3	254.0	18.8	1458.7	595.2	283.8
q_query_3_L100_coli.sat.cnf	100	609.4	763.3	522.6	129.6	2236.2	139.0	763.8
rbcl_xits_14_SAT.cnf	100	3.7	50.6	373.1	1.1	3600.0	1.9	51.1
shuffling-1-s11...s.sat04-461.cnf	100	6.2	1455.6	1642.4	3.3	3600.0	6.1	1456.0
uts-l06-ipc5-h33-unknown.cnf	100	234.7	276.4	173.0	109.0	1546.4	228.9	276.9
velev-fvp-sat-3.0-b18.cnf	100	34.9	65.3	69.7	16.4	372.3	6.2	65.8
velev-live-sat-1.0-03.cnf	100	127.4	153.6	92.6	52.9	628.4	71.7	154.1
vmpc_31.cnf	100	2179.5	2080.3	1395.9	2.9	3600.0	3600.0	2080.7
vmpc_33.cnf	100	2591.9	2244.0	1311.2	15.6	3600.0	305.8	2244.3

5.2 Experimental Setup

The benchmark set comprises 1676 industrial instances from 9 years of the SAT Competitions, Races, and Challenges between 2002 and 2011. We use MiniSat 2.0 [Eén and Sörensson, 2004] as the solver with a timeout of 1 hour and a limit of 2GB RAM. Performance data was collected on a cluster of Intel Xeon E5430 Processors (2.66GHz) running CentOS 6. A total of 315 weeks of CPU-time was consumed to accumulate this performance data. Table 4 presents a sample summary of the runtimes. The dataset is available at `http://ucc.insight-centre.org/bhurley/`

5.3 Predictive Fragility

To perform the regression we use a random forest with the same parameters as [Hutter *et al.*, 2014], that is using 10 regression trees, using half of the variables as split variables at each node ($perc = 0.5$), a maximum of 5 data-points in leaf nodes ($n_{min} = 5$). This configuration was shown to be the most accurate for runtime prediction in comparison to ridge regression, neural networks, Gaussian processes, and individual regression trees. The random forest may choose from a set of 138 SAT features [Lin Xu and Leyton-Brown, 2012] which have been used for runtime prediction and many award winning portfolios. As per [Hutter *et al.*, 2014] and other work on runtime prediction, we do not treat runs that timeout with any speciality, they are considered to just have taken the timeout value. We use a standard randomised 10-fold cross-validation, where the dataset is split in 10 folds. Each of the folds takes a turn as the test set, with a model being built from the remaining 9 folds of training data. We use a standard error metric for regression, the root-mean-squared error (RMSE) from the predicted value to the true test value.

Some simple statistical values are extracted from the runtime distribution on each instance in order to model a range of scenarios which may arise using the current methodology, where it is possible for the models to be trained on completely different runtime samples to that considered in testing. Table 5 presents a set of these scenarios, highlighting the fragility in current state-of-the-art runtime prediction methods. Columns vary the target metric and rows vary the training sample. Within each column, values in italics highlight the best mean RMSE, bold values are those which produced predictions statistically significantly different (95% confidence) from this best.

The closest setting to current methodology is shown in the *sample* entries, this simply uses a random sample of the observed runtimes. The predictions produced by these models are reasonable but are not the most accurate. Unsurprisingly, the most accurate predictions typically arise when we are using the same sample statistic for both training and testing. For example, when aiming to predict the median runtime it is best to have used the median runtime of the training set instances, likewise for predicting the minimum and maximum runtimes.

Conversely, another setting, albeit the most extreme, is if the training data consists of the minimum value from each distribution and we are attempting to predict the maximum runtime. In this case, the runtime prediction values are off by one order of magnitude on average. Likewise for predicting the minimum runtime where the maximum is used as training.

It is clear, and should seem natural, that the origin of the training sample has a considerable impact on the outcome of the models, but surprisingly, this is not something which is considered in existing methodology.

6 Related Work

A large body of work in the SAT community has studied runtime variations when running deterministic backtracking algorithms on distributions of random instances, but also by repeated runs of randomised backtracking algorithms on individual instances. The Handbook of Satisfiability dedicates a chapter [Gomes and Sabharwal, 2009] to runtime variations in complete solvers, modelling them as complex physical phenomena. The chapter focuses on an analysis and prevention of extreme variations using the framework of fat- and heavy-tailed distributions.

Table 5: Mean RMSE on $\log(1 + \text{runtime})$ predictions across 10-fold cross-validation, lower is better. Comparison of training and testing on different samples of each instance's observed runtimes. 'sample' is simply a single random sample from the observed runtimes for an instance. For each column, values in italics represent the best mean RMSE, bold values are statistically significantly different from this best.

		Test				
		sample	min	median	mean	max
Train	sample	**0.701**	**0.797**	0.669	**0.658**	**0.739**
	min	**0.823**	*0.631*	**0.807**	**0.833**	**1.006**
	median	**0.692**	**0.823**	*0.638*	*0.636*	**0.722**
	mean	*0.684*	**0.844**	**0.653**	**0.661**	**0.702**
	max	**0.768**	**1.016**	**0.755**	**0.712**	*0.669*

[Gomes *et al.*, 2000] first presented the heavy- and fat-tailed phenomenon in solver runtime. That paper was written during a time before randomised restarting was common and the fastest complete-search solvers for SAT were based on the DPLL method; modern SAT solvers have come a long way since. The paper also modified the algorithms to add an element of randomisation and a new heuristic equivalence parameter to increase its frequency. Our work has not modified the solvers in any way, we use the current state-of-the-art solvers. Nevertheless, it is evident that modern solvers have an element of stochasticity built in, resulting in runtime distributions with significant variance.

[Nikolić, 2010] formed a methodology for statistically comparing two solvers given observations from their runtime distributions. Our paper provides substantial additional evidence to support the need for such comparisons. [Van Gelder, 2011] proposes a new *careful ranking* method for competitions which addresses a number of issues with existing ranking methods, but the system is still based on individual runtime samples.

[Hutter *et al.*, 2014] presents the state-of-the-art for solver runtime prediction, evaluating the effectiveness of a number of machine learning methods for runtime prediction across a wide range of SAT, MIP, and TSP benchmarks. Our comparison in Section 5 does not compare different models on their accuracy, but does question a fundamental assumption being made about the underlying runtime behaviour.

[Fawcett *et al.*, 2014] tackles runtime predictions for state-of-the-art planners where they observe substantial runtime variations of planners across different domains. The authors propose new feature sets which improve the accuracy of predictions however the work does not consider runtime variations on individual instances, only a single sample runtime is taken.

[Leyton-Brown *et al.*, 2002] identified features of combinatorial auction problems to predict the runtime of CPLEX on different distributions of instances, but only one sample runtime is taken despite elements of randomness in the solver.

7 Conclusions and Discussion

This paper questions a fundamental assumption being made about the runtime behaviour of complete search solvers. Modern solvers usually have some form of in-built randomness. As a consequence their runtime exhibit significant variation, sometimes by orders of magnitudes on individual instances. We have shown that the outcome of empirical comparisons such as the SAT Competitions can fluctuate simply by re-running the experiments, and we provide statistical bounds on such variations. We also project the fragility of state-of-the-art runtime prediction methods to these runtime distributions, showing that it is insufficient to take a single sample of the runtime in the current practice. Such observations have broad reaching implications for empirical comparisons in a number of fields, solver analysis, portfolios, automated configuration, and runtime prediction.

There may not be a single solution that should be applied to overcome this phenomenon, but rather a context dependant approach should be taken. For empirical evaluations, solvers should be compared based on their runtime distributions. Statistical tests such as the Kolmogorov-Smirnov or chi-squared tests would be applicable. Of course, this increases the computational cost but gives the advantage of more authoritative conclusions. One related question concerns the appropriate number of runtime samples required to obtain representative results.

For the fields of runtime prediction, solver portfolios, and automated configurators, there remains a great number of open questions. It is simply insufficient to take a single sample of a solver's runtime as ground-truths. Alternatively we should consider predicting statistics or parameters of the runtime distribution. Depending on the application we may attempt to maximise the probability of solving an instance within a certain time, we may prefer solvers which give more consistent behaviour, or conversely a solver which varies dramatically in the hope that we may get lucky. Additionally, in a parallel setting this would have knock-on effects. There are many avenues for future study in these areas.

Acknowledgments

This work is supported by Science Foundation Ireland (SFI) Grant 10/IN.1/I3032 and FP7 FET-Open Grant 284715. The Insight Centre for Data Analytics is supported by SFI Grant SFI/12/RC/2289. We would like to thank Nic Wilson for his valuable contributions to Section 3.

References

[Alfonso and Manthey, 2014] Enrique Matos Alfonso and Norbert Manthey. Riss 4.27 BlackBox. SAT Competition, 2014.

[Ansótegui *et al.*, 2009] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A Gender-based Genetic Algorithm for the Automatic Configuration of Algorithms. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP 2009)*, pages 142–157, 2009.

[Ansótegui *et al.*, 2014] Carlos Ansótegui, Yuri Malitsky, and Meinolf Sellmann. MaxSAT by Improved Instance-Specific Algorithm Configuration. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 2594–2600, 2014.

[Biere, 2014] Armin Biere. Yet another Local Search Solver and Lingeling and Friends Entering the SAT Competition 2014. SAT Competition, 2014.

[Eén and Sörensson, 2004] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518. Springer-Verlag, 2004.

[Fawcett *et al.*, 2014] Chris Fawcett, Mauro Vallati, Frank Hutter, Jörg Hoffmann, Holger H Hoos, and Kevin Leyton-Brown. Improved Features for Runtime Prediction of Domain-Independent Planners. In *Proceedings of the 24th International Conference on Autonomous Planning and Scheduling (ICAPS 2014)*, 2014.

[Fitzgerald *et al.*, 2014] Tadhg Fitzgerald, Yuri Malitsky, Barry O'Sullivan, and Kevin Tierney. ReACT: Real-Time Algorithm Configuration through Tournaments. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014.*, 2014.

[Gomes and Sabharwal, 2009] Carla P. Gomes and Ashish Sabharwal. Exploiting Runtime Variation in Complete Solvers. In *Handbook of Satisfiability*, pages 271–288. 2009.

[Gomes and Selman, 2001] Carla P. Gomes and Bart Selman. Algorithm Portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[Gomes *et al.*, 2000] Carla P Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning*, 24(1-2):67–100, 2000.

[Hurley *et al.*, 2014] Barry Hurley, Lars Kotthoff, Yuri Malitsky, and Barry O'Sullivan. Proteus: A Hierarchical Portfolio of Solvers and Transformations. In *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014*, pages 301–317, 2014.

[Hutter *et al.*, 2011] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, pages 507–523, 2011.

[Hutter *et al.*, 2014] F Hutter, L Xu, H H Hoos, and K Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[Kadioglu *et al.*, 2010] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC – Instance-Specific Algorithm Configuration. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 751–756, 2010.

[Kotthoff, 2014] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[Le Berre and Simon, 2004] Daniel Le Berre and Laurent Simon. The Essentials of the SAT 2003 Competition. In *Theory and Applications of Satisfiability Testing*. Springer-Verlag, 2004.

[Leyton-Brown *et al.*, 2002] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the Empirical Hardness of Optimization Problems: The Case of Combinatorial Auctions. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, pages 556–572. Springer Berlin Heidelberg, January 2002.

[Lin Xu and Leyton-Brown, 2012] Holger Hoos Lin Xu, Frank Hutter and Kevin Leyton-Brown. Features for SAT, 2012.

[Nikolić, 2010] Mladen Nikolić. Statistical methodology for comparison of SAT solvers. In *Theory and Applications of Satisfiability Testing*, 2010.

[Oh, 2014] Chanseok Oh. MiniSat_HACK_999ED, MiniSat_HACK_1430ED and SWDiA5BY. SAT Competition, 2014.

[O'Mahony *et al.*, 2008] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving. *Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[Seipp *et al.*, 2015] J. Seipp, S. Sievers, M. Helmert, and F. Hutter. Automatic Configuration of Sequential Planning Portfolios. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, January 2015.

[Van Gelder, 2011] Allen Van Gelder. Careful Ranking of Multiple Solvers with Timeouts and Ties. In *Theory and Applications of Satisfiability Testing*. Springer-Verlag, 2011.

[Xu *et al.*, 2008] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. In *Journal Of Artificial Intelligence Research*, pages 565–606, 2008.