Revisiting SATZilla Features in 2024

Hadar Shavit
Chair for AI Methodology, RWTH Aachen University, Germany

Holger H. Hoos
Chair for AI Methodology, RWTH Aachen University, Germany
Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
Department of Computer Science, University of British Columbia, Vancouver, Canada

Abstract

Boolean satisfiability (SAT) is an $\mathcal{NP}$-complete problem with important applications, notably in hardware and software verification. Characterising a SAT instance by a set of features has shown great potential for various tasks, ranging from algorithm selection to benchmark generation. In this work, we revisit the widely used SATZilla features and introduce a new version of the tool used to compute them. In particular, we utilise a new preprocessor and SAT solvers, adjust the code to accommodate larger formulas, and determine better settings of the feature extraction time limits. We evaluate the extracted features on three downstream tasks: satisfiability prediction, running time prediction, and algorithm selection. We observe that our new tool is able to extract features from a broader range of instances than before. We show that the new version of the feature extractor produces features that achieve up to 26% lower RMSE for running time prediction, up to 3% higher accuracy for satisfiability prediction, and up to 15 times higher closed gap for algorithm selection on benchmarks from recent SAT competitions.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Satisfiability, feature extraction, running time prediction, satisfiability prediction

Digital Object Identifier 10.4230/LIPIcs.SAT.2024.27

Supplementary Material *Software (Source Code)*: `https://github.com/hadarshavit/revisiting_satzilla` [16], archived at `swh:1:dir:a06a5e5ad5473cd07b6a423a1dc5491b54af7c61`

Funding This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation program under GA No. 952215. HH gratefully acknowledges support through an Alexander-von-Humboldt Professorship in Artificial Intelligence.

Acknowledgements We would like to thank Matthias König and Anja Jankovic for the discussions. We would also like to thank the reviewers who gave a valuable feedback.

1 Introduction

The Boolean satisfiability problem (SAT) is an important problem in computer science from both theoretical and practical viewpoints. Common usages of SAT include hardware and software verification, cryptography, and more. However, as SAT is $\mathcal{NP}$-complete, it takes substantial time and computing power to solve it, which becomes prohibitively expensive as formulas become larger. In order to deepen our understanding of SAT itself and develop better SAT solvers, it is of crucial importance to be able to describe SAT instances via an informative set of features.

Some of the most widely adopted such features are the SATZilla features [13, 7]. These are a fixed set of features that are calculated from the DIMACS CNF representation of a SAT instance. The SATZilla features consist of multiple groups, ranging from basic syntactic features describing the formula, such as its number of variables and clauses, to more complicated features, such as probing features derived from short runs of SAT solvers. Another type of features are based on statistics of graph representations of a given formula.

The SATZilla features have been successfully used in various domains, such as empirical performance models (EPMs; also known as performance or running time prediction) and algorithm selection [17, 18], algorithm configuration [6], and benchmark generation [8]. The features are also used for caching in CDCL-based model counting solvers [14] and in a variety of SAT solvers that incorporate machine learning techniques [4] However, the SATZilla features in their latest version date back to 2012. Since then, the SAT community has undergone various changes. Most notably, SAT instances that we typically encounter today have a larger number of variables and clauses, thus taking significantly more time and memory to preprocess. Currently, the existing SATZilla feature extraction tool is unable to compute many of these features because of time and memory limitations.

In this paper, we revisit the SATZilla features and introduce a new version of the feature extraction tool. First, we replace the underlying solvers and the preprocessor with their most up-to-date versions. We then fix compilation errors and other memory errors related to dealing with larger formulas. Finally, we allow the user to set the time limits for feature computation. We compare the performance of our new tool with the old one in two SAT competitions. We measure the running times and the number of extracted features to check for performance gains of our new tool. We then evaluate the extracted features on three downstream tasks: satisfiability prediction, performance prediction and algorithm selection. We show that our new tool yields an important advantage in performance compared to the old tool across all three tasks.

The rest of the paper is organised as follows. We give a historical overview of the development and applications of the SATZilla features in Section 2. We then introduce the technical definitions, as well as the standard methodological pipeline in Section 3, wrapping it up with the contributions of this study. The results are presented in Section 4, and conclusions drawn in Section 5.

2 Related work

The SATZilla features were first introduced by Nudelman et al. [13] to construct EPMs, *i.e.*, machine learning models that predict the running time of various SAT solvers given the features representing SAT instances. The authors identified key features that contribute the most towards having a good EPM prediction. Consequently, they used the EPMs as a basis for algorithm selection, in which the algorithm that is selected corresponds to the one with the lowest predicted running time. They leveraged the *performance complementarity* phenomenon of SAT solvers, where no SAT solver dominates all others over all instances. Therefore, selecting the best solver for each instance results in substantially better performance compared to choosing any standalone solver for all instances. The SATZilla features were also successfully used for the satisfiability prediction task for SAT instances from various distributions [2].

Further developments in algorithm selection led to the 2007 version of the SATZilla algorithm selector [17], which combined running time and satisfiability prediction with other improvements. It won multiple medals in the SAT competition. This shows that extracting features from SAT instances can also speed up the solving process, and not only help understanding SAT. A newer version of the SATZilla algorithm selector was introduced in 2012 [18], winning multiple awards as well. It used a random forest as a predictor, and introduced an ensemble of pairwise classifiers to establish a ranking of the solvers, instead of predicting the running times directly. Additional features were introduced thereafter, revealing further important information about SAT instances.

Another common usage of the SATZilla features is (model-based) algorithm configuration. To this end, the hyperparameters of SAT solvers are optimised such that their performance is as good as possible on all instances. As running SAT solvers is computationally expensive, a surrogate model is employed; it takes as an input a configuration of hyperparameters and instance features and predicts the performance of the SAT solver using a given configuration on a given instance. The instance features boost the accuracy of the surrogate model. An example of that is SMAC [6], which successfully used the SATZilla features to optimise the performance of a wide range of SAT solvers on various benchmarks.

Last but not least, feature-based EPMs are proven to be useful for benchmark generation [8]. In this context, given a new instance, we compute the features, which is usually cheaper than running a SAT solver, and use the EPM to predict whether the instance is hard (or not) for the solvers at hand.

3 SATZilla features

The SATZilla features describe the SAT formula using various representations and statistics. We briefly introduce three graph representations of a SAT formula, as undirected graphs are a meaningful representation of SAT, maintaining the permutation invariance: a) variable graph: nodes are variables, an edge exists if variables appear in the same clause; b) clause graph: nodes are clauses, an edge exists when two clauses share a negated literal; and c) variable-clause graph: nodes are variables and clauses, an edge exists between a variable node and a clause node if the variable appears in the clause.

Feature computation starts with the *preprocessing* of the formula. This step, performed before solving an instance, renders the formula more accessible for SAT solvers. This means that the features are also computed on the version of the formula that is close to the one seen by the solvers. We believe that all these aspects can be boosted by using a modern preprocessor. Features are classically computed using the SATELITE preprocessor. We instead use the SBVA preprocessor, suggested by the winning solver from the 2023 SAT Competition. SBVA is also able to terminate after a set cutoff time, allowing for partial preprocessing, while SATELITE does not include this functionality. The preprocessed formula can then be directly used by a SAT solver without additional preprocessing within the solver, which improves the performance of algorithm selection.

Following the preprocessing, the *feature extraction* begins. There are ten feature groups that can be extracted. We note that we describe the feature groups according to their implementation in the SATZilla feature extraction tool, not according to their definitions from the corresponding paper [7]. We point out that all feature groups include the time required to compute the features in the group.
Preliminary features include the number of variables and clauses before/after preprocessing. The running time of this feature group includes the preprocessing time and the time required to read the formula. This group contains 7 features.
Basic features are cheap features that provide a basic description of the formula. They consist of the variable-clause ratio, the ratio between positive and negative literals in each clause, the number of unary, binary and ternary clauses, as well as statistics on clause nodes in the variable-clause graph. This group contains 15 features.
KLB are expensive features that include the node degree statistics of the variable nodes in the variable-clause graph, and the ratio of positive to negative occurrences of each variable. They also include measures for the proximity to Horn formula, such as the fraction of Horn clauses and statistics on the number of times each variable appears in a Horn clause. This group contains 21 features.

Clause graph (CG) features are expensive features that contain statistics on the degree of the nodes in the clause graph, as well as the clustering coefficient. This group contains 11 features.

Diameter features contain information on the diameter of the variable graph, which is the shortest path between each pair of nodes in the graph. This group contains 6 features.

DPLL probing (or unit propagation) features are computed by running the DPLL algorithm for various depths and measuring the number of unit propagations at each depth. This group contains 6 features.

Lobjois features are an estimation of the size of the search space. They are computed by running the DPLL algorithm multiple times until a contradiction is found. Then, the average depth of the contradictions is the log-estimation of the search space. This group contains 3 features, which are all based on the work of Lobjois and Lemaître [11].

Survey propagation features are based on computing statistics on the following probabilities returned by the VARSAT [5] solver: a probability of each variable to be assigned to True, to False, and to be unconstrained. This group contains 19 features.

Clause learning (CL) features are based on running a CDCL solver (ZChaff rand [12] in the 2012 version, CadiCal [1] in our new version) for two seconds. We measure the number and length of learned clauses for every 1000 decisions, and compute the statistics of those values. This group contains 19 features.

Local search (LS) features are obtained by running two local search solvers many times, each time up to 10000 steps, and computing statistics on those runs. In the 2012 version, the local search solvers are GSAT and SAPS. We instead use GSAT and Sparrow 2011 in our new version. This group contains 24 features.

Linear programming (LP) features are based on solving a relaxed version of the SAT formula, where $C_1 \wedge C_2 \wedge \dots \wedge C_N$ is a Boolean formula with N CNF clauses $C_1 \dots C_N$ over Boolean variables x_j. We now consider linear programming variables x_j and solve the following linear programming problem: Maximise $\sum_{i=1}^{N} \sum_{l \in C_i} v(l)$, where the value $v(l)$ of literal l is defined as $v(x_j) = x_j$, $v(\neg x_j) = 1 - x_j$, while keeping $\sum_{l \in C_i} v(l) \geq 0$ for each C_i and $0 \leq x_j \leq 1$ for each x_j. This means that every variable has a value between 0 and 1, each clause has a value which is the sum of values of all literals, and the value of the formula is the sum of values of all clauses. The goal is to maximise the value of the formula, while keeping the value of every clause positive. Finally, statistics on the linear programming solution are extracted. In the new version, we upgrade the linear programming solver package, lp solve. This group contains 7 features.

We note that the basic, KLB and clause graph features are computed sequentially, with each feature group being dependent on the successful computation of the previous groups; *e.g.*, if the computation of the basic features fails, the KLB and clause graph features will not be computed at all.

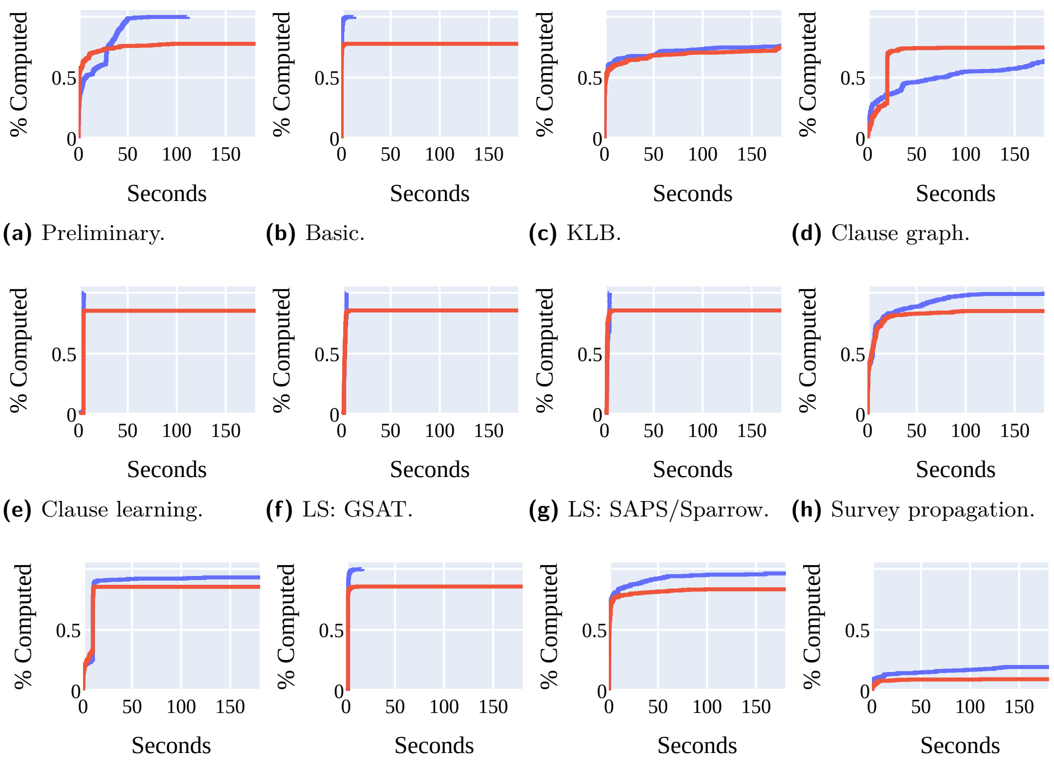
Another change we applied to the feature extraction tool is allowing a more precise timeout setting. In the 2012 version, the time limits were hard-coded and set to high values (for example, 1200 seconds for preprocessing). This can cause many feature computations to simply terminate without computing any features at all. To this end, we adjust the code in the new version to allow the user to set the time limits through a command line argument.

4 Experiments

We evaluate our new SATZilla feature extraction tool on the formulas stemming from two latest (2022 and 2023) SAT Competitions. We first look at the feature extraction times and then evaluate the features on three downstream tasks: satisfiability prediction, running time prediction, and algorithm selection. To assess the advantage of using the new version of the extraction tool, we extract the features using both our new version and the 2012 version of the tool, with a time limit of 180 seconds per feature group.

We use a cluster of 18 nodes, each equipped with 2 AMD EPYC 7543 32-core CPUs with 256 MB L3 cache. Each node also has 1TB of memory. The cluster is running on a Rocky Linux 9.4 operating system. We measure running times using the runsolver tool [15].

4.1 Feature computation time



(a) Preliminary. **(b)** Basic. **(c)** KLB. **(d)** Clause graph.

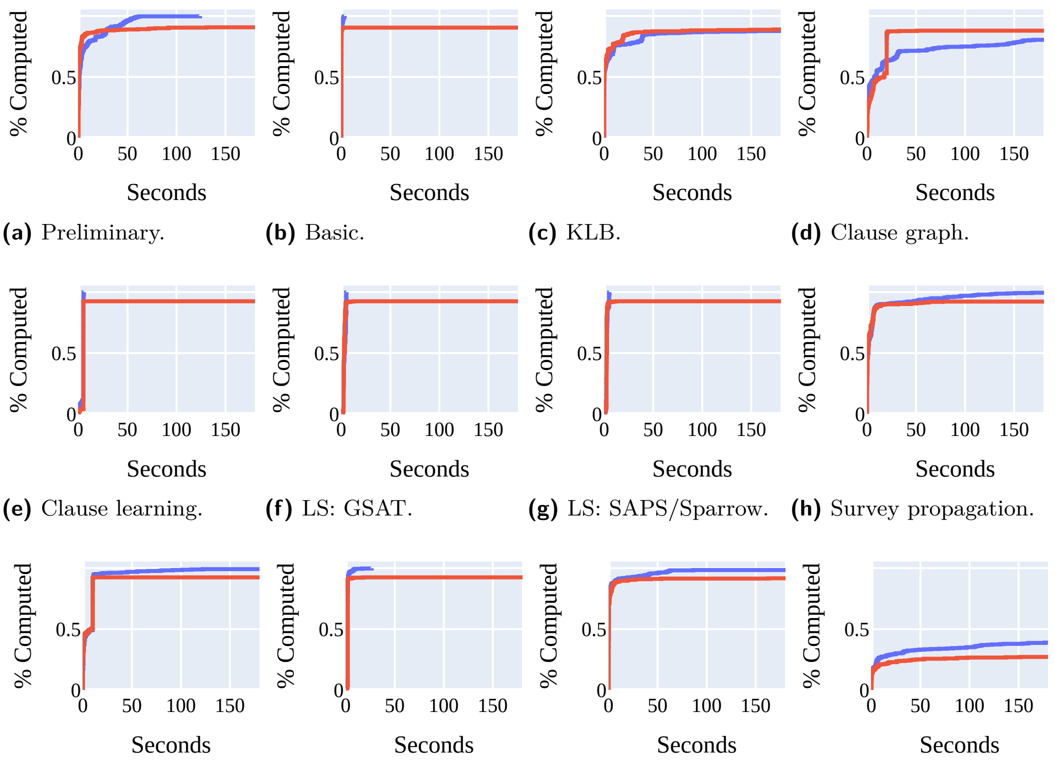
(e) Clause learning. **(f)** LS: GSAT. **(g)** LS: SAPS/Sparrow. **(h)** Survey propagation.

(i) Diameter. **(j)** Lobjois. **(k)** Unit propagation. **(l)** Linear programming.

Figure 1 Percentage of features computed by the old tool (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue) over the available time budget for each feature group on the 2022 SAT Competition. For most feature groups, the new tool extracts features from more instances than the old one.

Figure 1 and Figure 2 show the percentage of features computed over the available time budget using the old (depicted in red) and new (depicted in blue) SATZilla tool on the 2022 and 2023 SAT Competition data, respectively. For simplicity, due to the similarity of the overall results between the two competitions, we draw more detailed insights based on plots from the 2023 edition (Figure 2).

We first observe that the new tool is able to extract more features than the old one for most feature groups. In particular, we highlight the performance gains on the preliminary feature group (Figure 2a), for which the new tool can extract the features for all formulas, compared to less than 80% of the formulas when using the the old tool. We note that for some feature groups, like graph learning (Figure 2d), the old tool is able to extract more features compared to the preliminary feature group. This is due to the fact that the old tool extracts the preliminary, basic, KLB and CG feature groups together. Therefore, in case computing one of those groups takes a long time, the whole feature extraction fails.



(a) Preliminary. **(b)** Basic. **(c)** KLB. **(d)** Clause graph. **(e)** Clause learning. **(f)** LS: GSAT. **(g)** LS: SAPS/Sparrow. **(h)** Survey propagation. **(i)** Diameter. **(j)** Lobjois. **(k)** Unit propagation. **(l)** Linear programming.

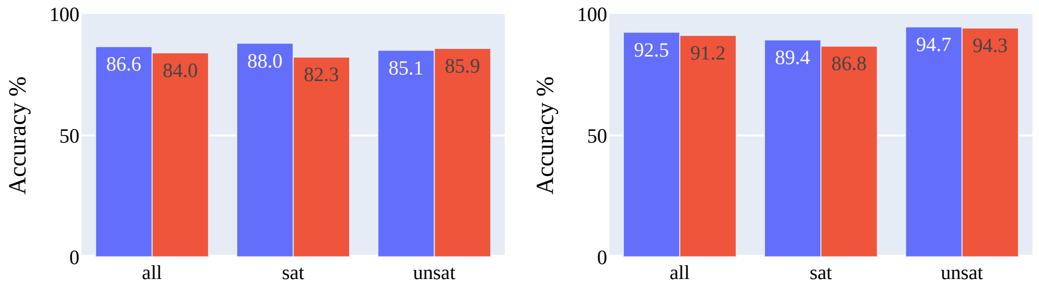
Figure 2 Percentage of features computed by the old tool (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue) over the available time budget for each feature group on the 2023 SAT Competition. For most feature groups, the new tool extracts features from more instances than the old one.

We also note that for KLB and CG features there is an advantage for the old version, due to the new SBVA preprocessing method yielding larger formulas than its predecessor SATElite (as in many cases smaller formulas are not always easier to solve). Similarly, the expensive graph-based features (*e.g.*, Figure 2d and Figure 2i) require more time to extract than smaller formulas. This is more apparent in the 2022 SAT Competition, where larger instances were used than in the 2023 SAT Competition.

4.2 Satisfiability prediction

The first downstream task is satisfiability prediction, for which we measure the performance when using features extracted by our tool. We use the random forest (RF) classifier from the scikit-learn package to learn the mapping between features (representing SAT instances) and outputs (satisfiable or unsatisfiable). We optimise the hyperparameters of the RF for one hour using SMAC3 [9] on 10-fold cross-validation of the training data. We consider instances from the 2022 and 2023 SAT Competitions for which we know the satisfiability result (put differently, we omit instances for which the solution is unknown). On each competition, we evaluate the performance of the RF model using 10-fold cross-validation. This results in having outer cross-validation (for evaluation) and inner cross-validation (for training). Such techniques have been previously used by AutoFolio [10].

We present the satisfiability prediction accuracy scores in Figure 3. We see that, by using features extracted via the new tool, we achieve better performance across all instances on both SAT competitions. Furthermore, we notice that the new tool leads to a higher accuracy gain for satisfiable instances than for unsatisfiable instances. For the latter, the accuracy remains very similar to the one achieved by using the old tool in the 2023 SAT Competition and slightly dropped for the 2022 SAT Competition. This might be due to the fact that the unsatisfiable instances are larger on average, thus being more prone to timeouts even when using our new tool, which goes along with the worse performance in the 2022 SAT Competition, where the unsatisfiable instances were larger than in the 2023 SAT Competition. We point out that such high accuracy was already achieved before for industrial SAT instances [2].



(a) 2022 SAT Competition. **(b)** 2023 SAT Competition.

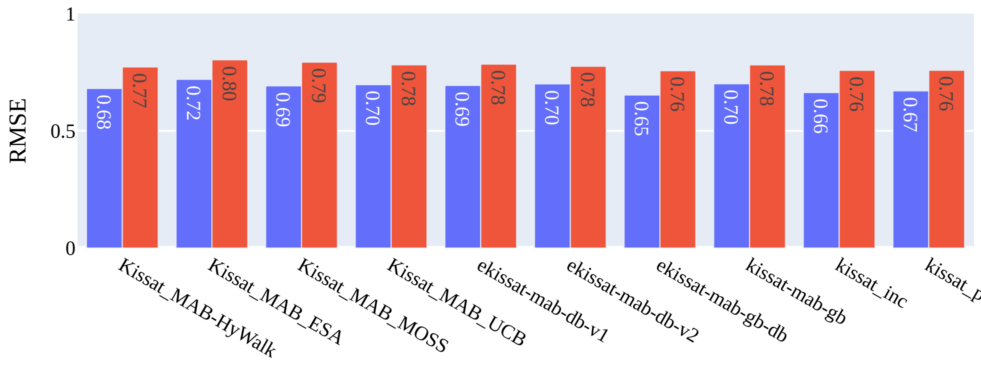
Figure 3 Accuracy of the satisfiability prediction task using a random forest with features extracted by the old (SATZilla 2012) and the new tool (SATZilla 2024). We see an overall higher accuracy for the new tool, which results from higher accuracy on satisfiable instances. On unsatisfiable ones, the accuracy remains the same.

4.3 Performance prediction

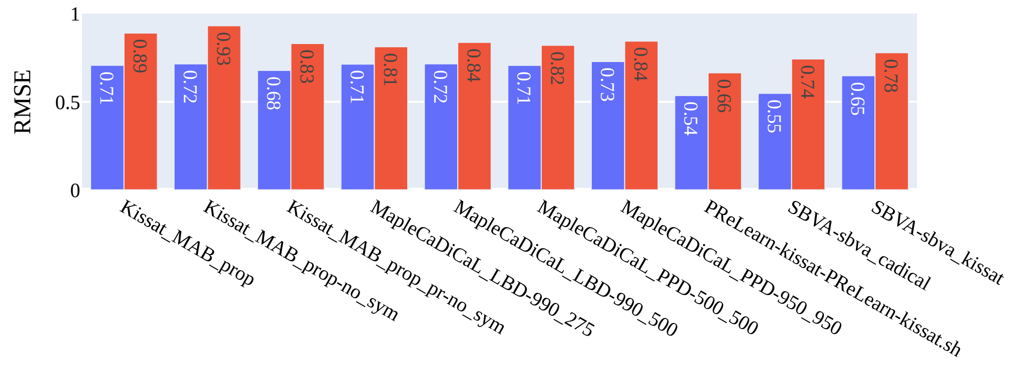
The second downstream task we investigate is performance prediction, which has important applications in algorithm selection, configuration and benchmark generation. We refer to the methodology described in [7] and use a RF regressor as the EPM. It is important to note that for an accurate running time prediction we need to perform a log_{10} transformation of running times prior to training the model, as done by Hutter *et al.* [7]. This transformation allows to capture the order of magnitude rather than small variations in the running time. The EPM then maps instance features to the log-transformed running times, and we aim to minimise the root mean squared error (RMSE) of the model, which is defined as $\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$ for n predicted running times $\hat{y}_i$ and ground truth running times y_i. Lower RMSE scores are better and 0 is the optimal value. In line with the previous task, we perform inner and outer cross-validation and optimise the RF's hyperparameters for one hour using SMAC3.

We look into the performance of the EPM for running time prediction on all solvers from the 2022 and 2023 SAT Competitions. Here, we do not actually run solvers on instances to record their running times, but rather use the running times reported by the competitions. We display the results for the 10 best solvers from each competition in Figure 4 (the results for all solvers can be found in the supplementary material). We see that using the features extracted by the new tool leads to the lower RMSE for all solvers, compared to using those extracted by the old tool. For some solvers, we observe a significantly lower RMSE, like

Kissat_MAB_prop-no_sym, where using the new tool decreases the RMSE from 0.84 to 0.69. Figure 5 shows the histogram of the error percentage for all solvers in the 2022 and 2023 SAT Competitions. We see that for the 2022 competition, the new tool has more instances with error rate lower than 10% compared to the old version. For the 2023 competition, using the features extracted with the new version, more instances are predicted with less than 1% error. Histograms per solver are available in Appendix B.



(a) 2022 SAT Competition.



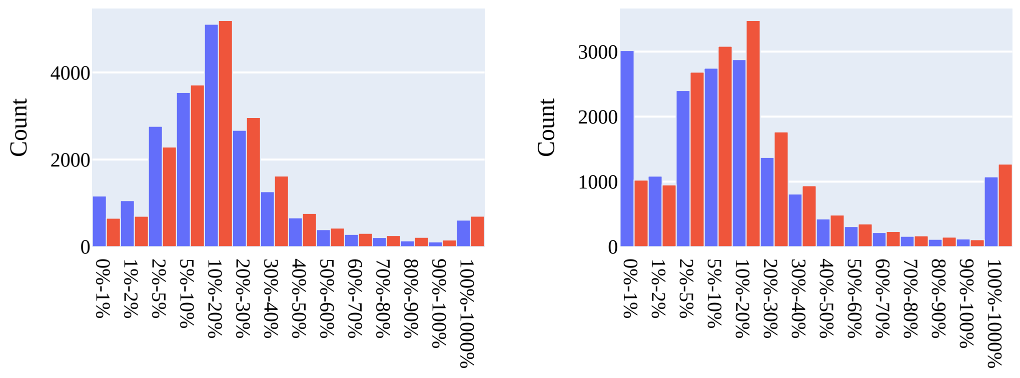
(b) 2023 SAT Competition.

Figure 4 Root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. The new tool achieves lower RMSE for all solvers.

4.4 Algorithm selection

The third and final downstream task we consider is algorithm selection. In algorithm selection, given a set of instances I, a set of solvers (*i.e.*, algorithm portfolio) $\mathbf{A}$ and a performance metric $m : \mathbf{A} \times I \to \mathbb{R}$, we build an algorithm selector $S : I \to \mathbf{A}$ such that its performance is optimal on the instance set I according to the metric m. We compare the performance of algorithm selection to two standard baselines, the single best solver (SBS; *i.e*, the solver with the lowest overall running time) and the virtual best solver (VBS; *i.e.*, the theoretical oracle which for each instance selects the actual best solver on it).

We measure the performance of algorithm selection using *closed gap*, which is computed as $\frac{m_{SBS}-m_S}{m_{SBS}-m_{VBS}}$, (*i.e.*, the closed gap stands for how much of the gap between the SBS and the VBS is closed by using the algorithm selector). We then use AutoFolio [10] as an algorithm



(a) 2022 SAT Competition. **(b)** 2023 SAT Competition.

Figure 5 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. The new tool achieves lower error percentages.

selector, which consists of multiple algorithm selection approaches, from which the best one is suggested by using algorithm configuration. As algorithm portfolio for the selection, we use the 10 best solvers from each SAT competition. We train the selector for 8 hours.

Figure 6 shows the closed gap results on the 2022 and 2023 SAT Competitions. Positive closed gap values on both scenarios using both the old and the new tool indicate that, in general, SATZilla features are useful for the algorithm selection task. Importantly, features extracted with the new tool lead to better closed gap values on both scenarios. In Figure 7, we provide ECDF plots showing the fraction of instances solved over time. In the 2022 SAT Competition scenario, the old version of the tool performs worse than the SBS until a budget of approximately 1000 seconds, while the new version of the tool performs similarly to the SBS. After 1000 seconds, both versions of the SATZilla features perform similarly. In the 2023 SAT Competition scenario, the new tool performs better than the old one for budgets between 100 and 1000 seconds. With a budget of less than 10 seconds, the old tool solves more instances. Overall, the ECDF plots reflect well what is shown in Figure 6, where we see that the new tool exhibits a few percents higher closed gap value.

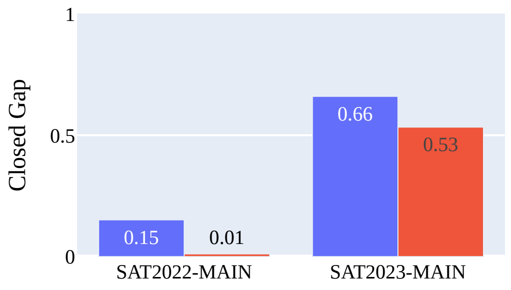
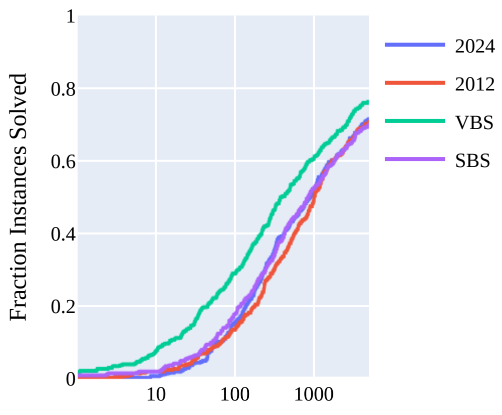
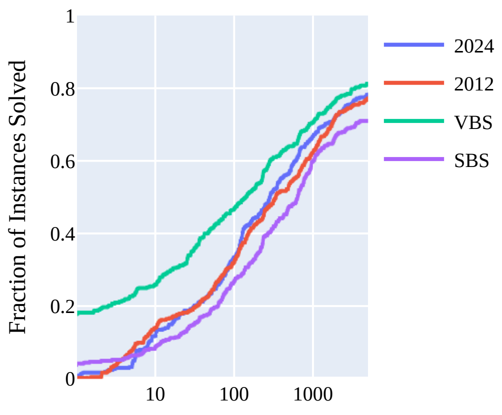


Figure 6 Closed gap values for the algorithm selection task using the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue) on the 2022 and 2023 SAT Competitions; higher is better.



(a) 2022 SAT Competition.



(b) 2023 SAT Competition.

Figure 7 ECDF plots for the algorithm selection task using the old (SATZilla 2012) and the new tool (SATZilla 2024) on the 2022 and 2023 SAT Competitions.

5 Conclusions

In this paper, we introduced an improved version of the well-known SATZilla feature extraction tool, motivated by the need to facilitate the feature extraction process by incorporating better user infrastructure and bug fixing. Our new version uses most up-to-date preprocessing techniques and SAT solvers, which allow for better representation of SAT formulas. Our experiments showed that, by using the new tool, we achieve a more accurate satisfiability prediction, a lower error for running time prediction, and a better closed gap for algorithm selection.

Our new SATZilla 2024 extraction tool aims to facilitate and promote the usage of SAT features even beyond their current scope. It is easily extensible and thus encourages building atop, *e.g.*, by looking into features based on the recent developments in the explainability of SAT solvers [3], or other advancements in SAT.

References

1 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions*, pages 51–53, 2020. URL: `https://researchportal.helsinki.fi/en/publications/proceedings-of-sat-competition-2020-solver-and-benchmark-descript`.

2 David Devlin and Barry O'Sullivan. Satisfiability as a classification problem. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*, pages 1–10, 2008. URL: `http://www.cs.ucc.ie/~osullb/pubs/classification#:~:text=Sat%20can%20be%20seen%20as,the%20problem%20of%20deciding%20Sat`.

3 Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical CDCL insights from theoretical SAT benchmarks. In Jérôme Lang, editor, *Proceedings of the 37th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1300–1308, 2018. `doi:10.24963/ijcai.2018/181`.

4 Wenxuan Guo, Hui-Ling Zhen, Xijun Li, Wanqian Luo, Mingxuan Yuan, Yaohui Jin, and Junchi Yan. Machine learning methods in solving the boolean satisfiability problem. *Machine Intelligence Research*, 20(5):640–655, 2023. `doi:10.1007/s11633-022-1396-2`.

5 Eric I. Hsu and Sheila A. McIlraith. VARSAT: integrating novel probabilistic inference techniques with DPLL search. In Oliver Kullmann, editor, *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 377–390, 2009. doi:10.1007/978-3-642-02777-2_35.

6 Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 507–523, 2011. doi:10.1007/978-3-642-25566-3_40.

7 Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014. doi:10.1016/j.artint.2013.10.003.

8 Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM*, 56(4):22:1–22:52, 2009. doi:10.1145/1538902.1538906.

9 Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23:54:1–54:9, 2022. URL: http://jmlr.org/papers/v23/21-0888.html.

10 Marius Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015. doi:10.1613/jair.4726.

11 Lionel Lobjois and Michel Lemaître. Branch and bound algorithm selection by performance prediction. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 353–358, 1998. URL: http://www.aaai.org/Library/AAAI/1998/aaai98-050.php.

12 Yogesh S. Mahajan, Zhaohui Fu, and Sharad Malik. Zchaff2004: An efficient SAT solver. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, volume 3542, pages 360–375, 2004. doi:10.1007/11527695_27.

13 Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. Understanding random sat: Beyond the clauses-to-variables ratio. In *Proceedings of the 10TH International Conference on Principles and Practice of Constraint Programming*, pages 438–452, 2004. doi:10.1007/978-3-540-30201-8_33.

14 Jeroen Rook, Anna Latour, Holger Hoos, and Siegfried Nijssen. Caching in model counters: A journey through space and time. *Proceedings of the Workshop on Counting and Sampling*, 2021. URL: https://ada.liacs.nl/papers/RooEtAl21.pdf.

15 Olivier Roussel. Controlling a solver execution with the runsolver tool. *Journal Satisfiability Boolean Modelling Computation*, 7(4):139–144, 2011. doi:10.3233/sat190083.

16 Hadar Shavit and Holger Hoos. hadarshavit/revisiting_satzilla. Software, swhId: swh:1:dir:a06a5e5ad5473cd07b6a423a1dc5491b54af7c61 (visited on 2024-07-04). URL: https://github.com/hadarshavit/revisiting_satzilla.

17 Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008. doi:10.1613/jair.2490.

18 Lin Xu, Frank Hutter, Jonathan Shen, Holger H Hoos, and Kevin Leyton-Brown. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge 2012*, pages 57–58, 2012. URL: https://www.cs.ubc.ca/~kevinlb/papers/2012-SATzilla2012-Solver-Description.pdf.

A Running time prediction results

In this appendix, we present the full results of running time prediction for all solvers from the 2022 and 2023 SAT Competitions. In Table 1 and Table 2 we show the results for the 2022 SAT Competition, where we see that our new tool constantly outperforms the old one. Similarly, in Table 3 we show the results for the 2023 SAT Competition, where using the features extracted by our new tool leads to lower RMSE than with the old one.

Table 1 RMSE of random forest for predicting log-transformed running times of SAT solvers from the 2022 SAT Competition using the old and new SATZilla features.

Solver	SATZilla 2012	SATZilla 2024
CaDiCaL-watchsat-lto	0.70	**0.63**
CaDiCaL_DVDL_V1	0.73	**0.65**
CaDiCaL_DVDL_V2	0.75	**0.65**
CadicalReorder	0.75	**0.66**
Cadical_ESA	0.75	**0.66**
IsaSAT	0.73	**0.64**
Kissat-MAB-rephasing	0.74	**0.67**
Kissat_MAB-HyWalk	0.77	**0.68**
Kissat_MAB_ESA	0.80	**0.72**
Kissat_MAB_MOSS	0.79	**0.69**
Kissat_MAB_UCB	0.78	**0.70**
Kissat_adaptive_restart	0.72	**0.68**
Kissat_cfexp	0.79	**0.71**
LSTech_CaDiCaL	0.77	**0.63**
LSTech_Maple	0.74	**0.66**
LSTech_kissat	0.76	**0.67**
LStech-Maple-BandSAT	0.67	**0.62**
LStech-Maple-FPS	0.74	**0.65**
LStech-Maple-HyWalk	0.72	**0.64**
MapleLCMDistChrBt-DL-v3	0.61	**0.58**
MergeSat 4.0-rc-rc3	0.65	**0.61**
SLIME SC-2022	0.70	**0.65**
SLIME SC-2022-alpha	0.70	**0.66**
SLIME SC-2022-beta	0.71	**0.66**
SLIME SC-2022-gamma	0.73	**0.69**
SeqFROST-ERE-All	0.76	**0.68**
SeqFROST-NoExtend	0.77	**0.67**
cadical-hack-gb	0.73	**0.62**
cadical_rel_Scavel	0.71	**0.61**

Table 2 RMSE of random forest for predicting log-transformed running times of SAT solvers from the 2022 SAT Competition using the old and new SATZilla features (contd.).

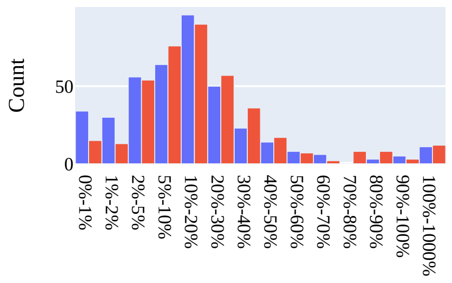
Solver	SATZilla 2012	SATZilla 2024
ekissat-mab-db-v1	0.78	**0.69**
ekissat-mab-db-v2	0.78	**0.70**
ekissat-mab-gb-db	0.76	**0.65**
glucose-reboot	0.78	**0.68**
hCaD_V1-psids	0.69	**0.63**
hCaD_V2	0.68	**0.64**
hKis-psids	0.69	**0.59**
hKis-sat	0.79	**0.69**
hKis-unsat	0.74	**0.66**
kissat-els-v1	0.76	**0.67**
kissat-els-v2	0.74	**0.67**
kissat-els-v3	0.78	**0.67**
kissat-els-v4	0.77	**0.66**
kissat-mab-gb	0.78	**0.70**
kissat-sc2022-bulky	0.76	**0.69**
kissat-sc2022-hyper	0.76	**0.70**
kissat-sc2022-light	0.75	**0.69**
kissat-watchsat-lto	0.73	**0.67**
kissat_inc	0.76	**0.66**
kissat_pre	0.76	**0.67**
kissat_relaxed	0.72	**0.67**

Table 3 RMSE of random forest for predicting log-transformed running times of SAT solvers from the 2023 SAT Competition using the old and new SATZilla features.

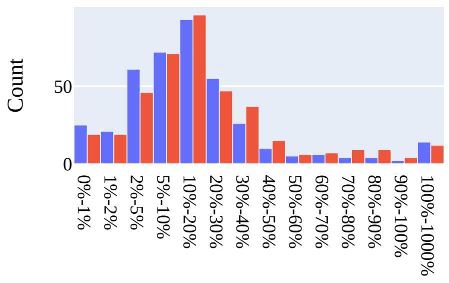
Solver	SATZilla 2012	SATZilla 2024
AMSAT_	0.80	**0.77**
BreakID-kissat-low.sh	0.92	**0.83**
CaDiCaL_vivinst	0.84	**0.73**
Cadical_ESA	0.85	**0.74**
Cadical_rel_1.5.3.Scavel	0.80	**0.74**
IsaSAT	0.90	**0.82**
Kissat_Inc_ESA	0.88	**0.73**
Kissat_MAB_Binary	0.89	**0.72**
Kissat_MAB_Conflict	0.87	**0.72**
Kissat_MAB_Conflict+	0.88	**0.75**
Kissat_MAB_DeepWalk+	0.87	**0.73**
Kissat_MAB_ESA	0.91	**0.74**
Kissat_MAB_Rephases	0.84	**0.71**
Kissat_MAB_prop	0.89	**0.71**
Kissat_MAB_prop-no_sym	0.93	**0.72**
Kissat_MAB_prop_pr-no_sym	0.83	**0.68**
MapleCaDiCaL_LBD-990_275	0.81	**0.71**
MapleCaDiCaL_LBD-990_500	0.84	**0.72**
MapleCaDiCaL_PPD-500_500	0.82	**0.71**
MapleCaDiCaL_PPD-950_950	0.84	**0.73**
MergeSat-bve_gates	0.75	**0.71**
MergeSat-bve_semgates	0.74	**0.73**
MergeSat-thread1	0.68	**0.67**
MiniSat+XorEngine	0.79	**0.77**
PReLearn-kissat-PReLearn-kissat.sh	0.66	**0.54**
PReLearn-kissat-PReLearn-tern-kissat.sh	0.55	**0.46**
ReEncode-kissat-ReEncode-pair-kissat.sh	0.75	**0.68**
SBVA-sbva_cadical	0.74	**0.55**
SBVA-sbva_kissat	0.78	**0.65**
SeqFROST	0.81	**0.73**
SeqFROST-ERE-All	0.81	**0.71**
SeqFROST-NoExtend	0.79	**0.74**
hKis-psids	0.81	**0.72**
hKis-sat_psids	0.80	**0.79**
hKis-unsat	0.86	**0.69**
hKissatInc-unsat	0.85	**0.77**
kissat-3.1.0	0.88	**0.76**
kissat-hywalk-exp	0.85	**0.70**
kissat-hywalk-exp-gb	0.89	**0.74**
kissat-hywalk-gb	0.88	**0.72**
kissat_incsp	0.95	**0.80**
tabularasat-1.0.0	0.88	**0.76**

B Running time prediction histograms

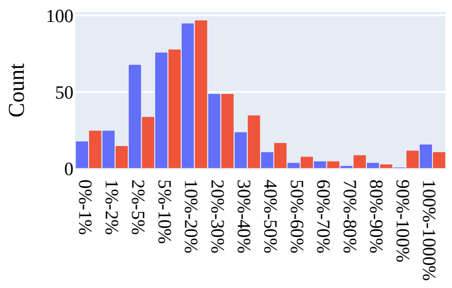
We present the histograms for the RMSE values per solver for running time prediction in Figures 8–19.



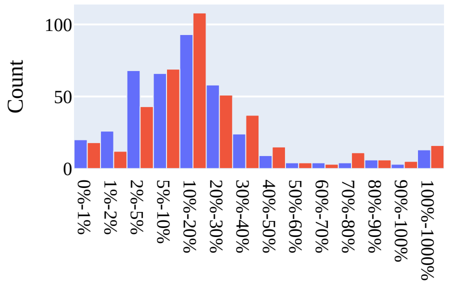
(a) 2022 Comp. CaDiCaL-watchsat-lto.



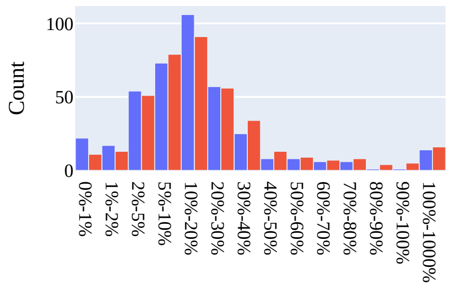
(b) 2022 Comp. CaDiCaL_DVDL_V1.



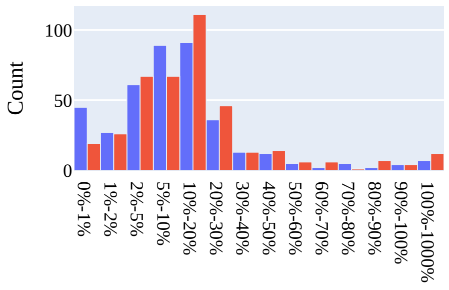
(c) 2022 Comp. CaDiCaL_DVDL_V2.



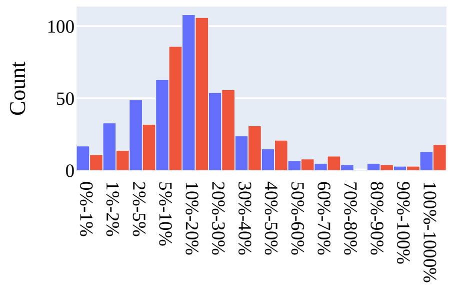
(d) 2022 Comp. CadicalReorder.



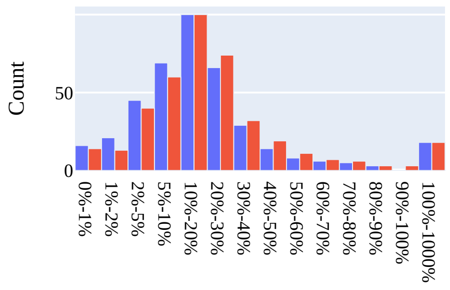
(e) 2022 Comp. Cadical_ESA.



(f) 2022 Comp. IsaSAT.

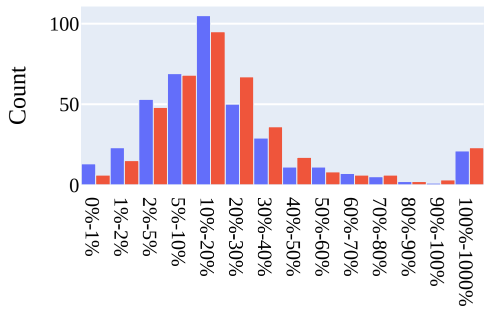


(g) 2022 Comp. Kissat-MAB-rephasing.

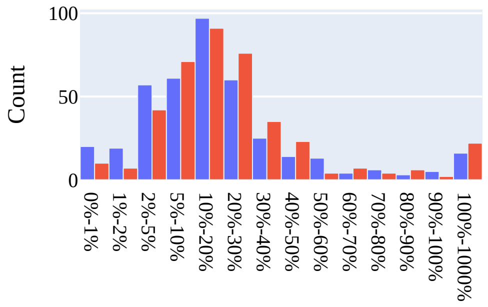


(h) 2022 Comp. Kissat_MAB-HyWalk.

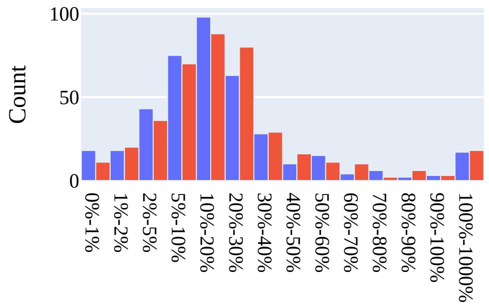
Figure 8 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver.



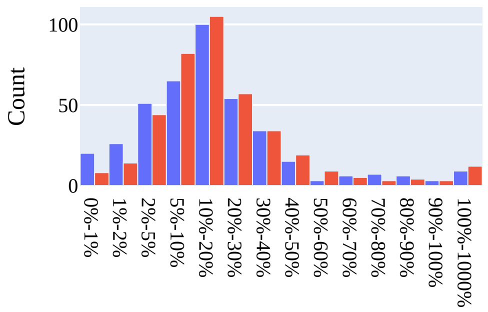
(a) 2022 Comp. Kissat_MAB_ESA.



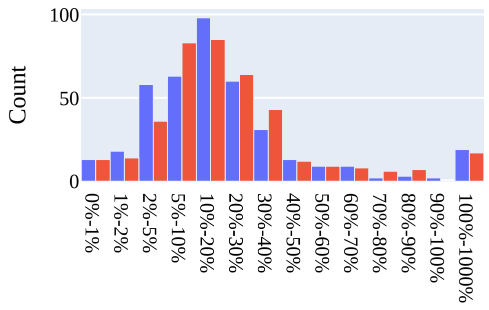
(b) 2022 Comp. Kissat_MAB_MOSS.



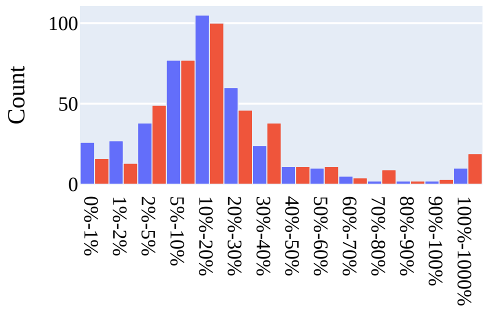
(c) 2022 Comp. Kissat_MAB_UCB.



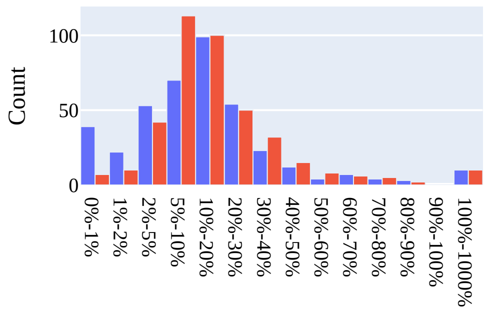
(d) 2022 Comp. Kissat_adaptive_restart.



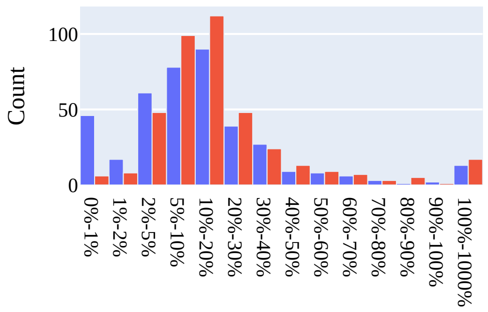
(e) 2022 Comp. Kissat_cfexp.



(f) 2022 Comp. LSTech_CaDiCaL.

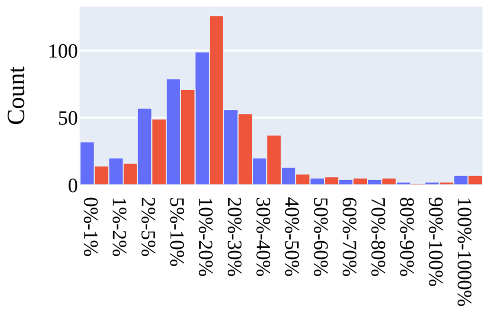


(g) 2022 Comp. LSTech_Maple.

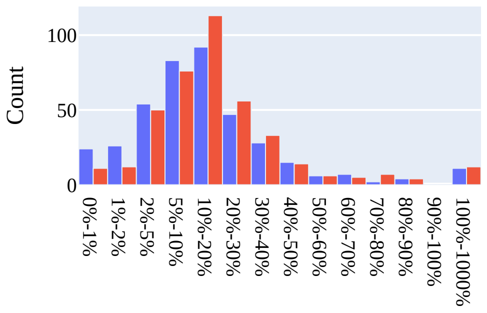


(h) 2022 Comp. LSTech_kissat.

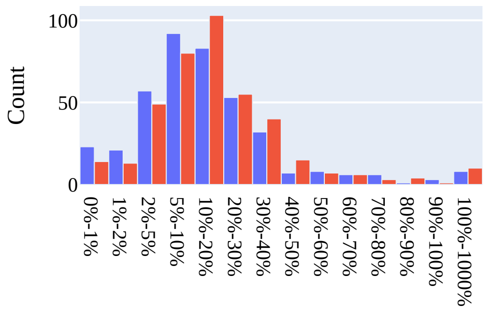
Figure 9 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



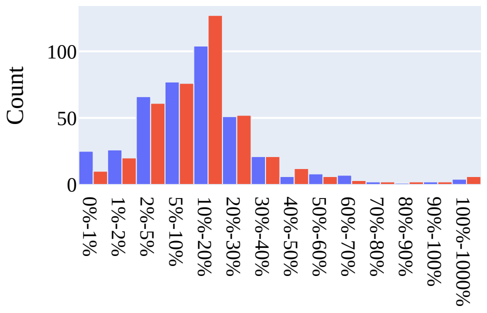
(a) 2022 Comp. LStech-Maple-BandSAT.



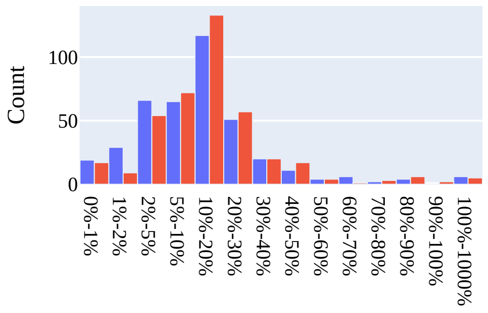
(b) 2022 Comp. LStech-Maple-FPS.



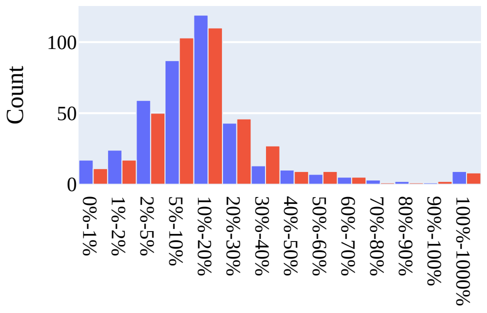
(c) 2022 Comp. LStech-Maple-HyWalk.



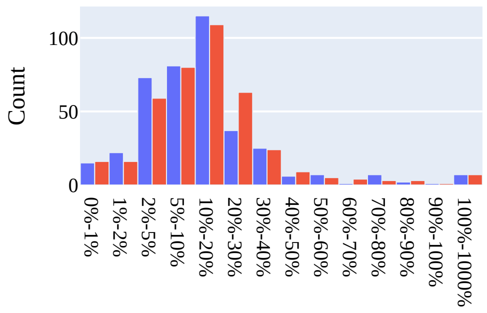
(d) 2022 Comp. MapleLCMDistChrBt-DL-v3.



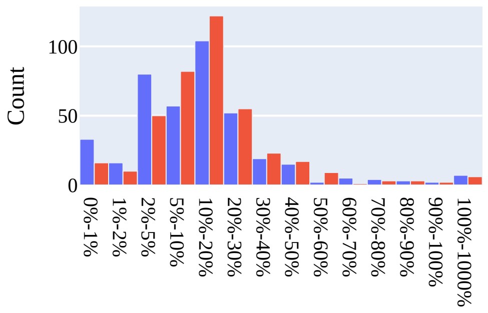
(e) 2022 Comp. MergeSat 4.0-rc-rc3.



(f) 2022 Comp. SLIME SC-2022.

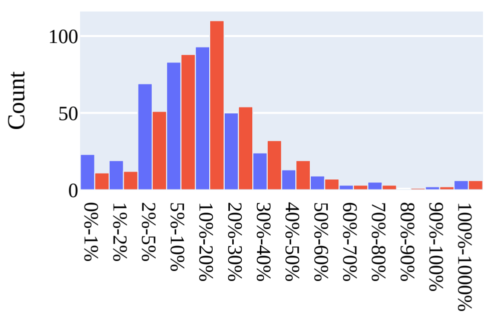


(g) 2022 Comp. SLIME SC-2022-alpha.

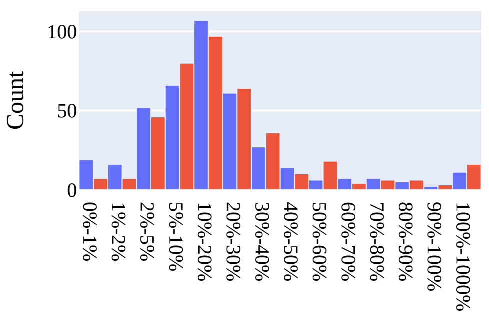


(h) 2022 Comp. SLIME SC-2022-beta.

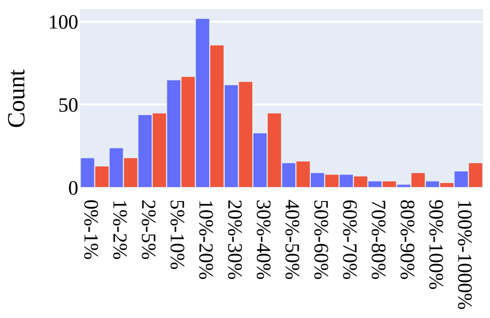
Figure 10 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



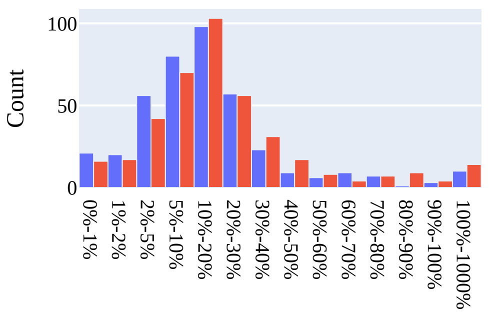
(a) 2022 Comp. SLIME SC-2022-gamma.



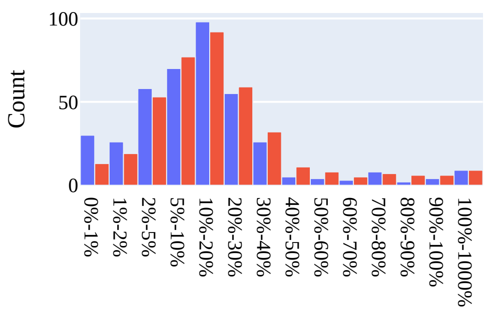
(b) 2022 Comp. SeqFROST-ERE-All.



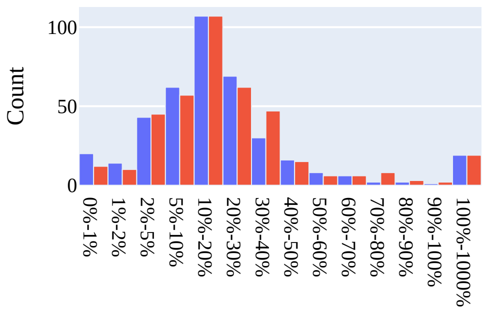
(c) 2022 Comp. SeqFROST-NoExtend.



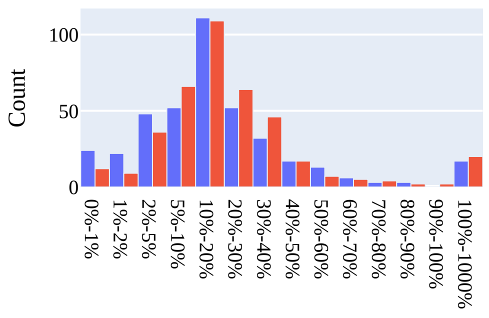
(d) 2022 Comp. cadical-hack-gb.



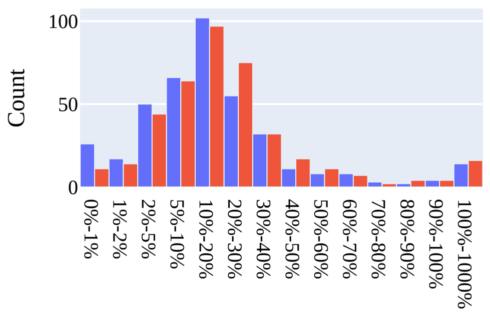
(e) 2022 Comp. cadical_rel_Scavel.



(f) 2022 Comp. ekissat-mab-db-v1.

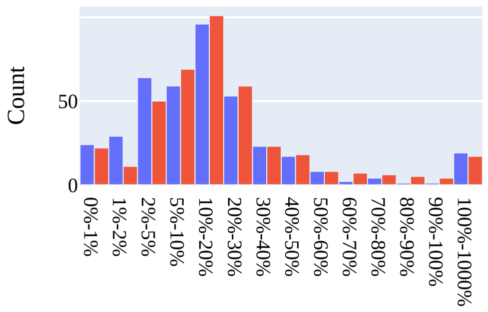


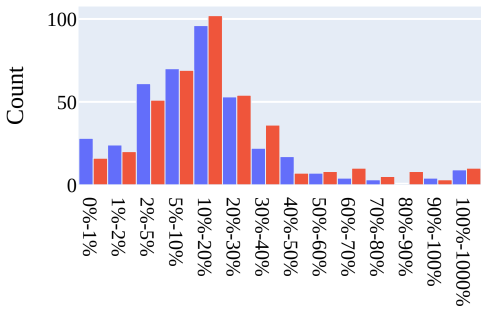
(g) 2022 Comp. ekissat-mab-db-v2.

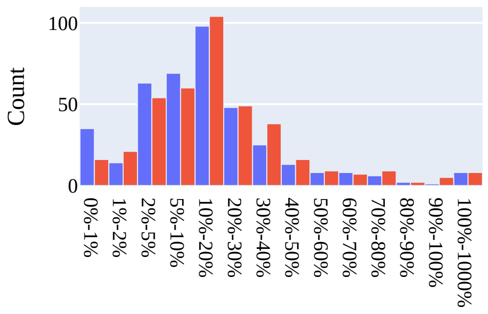


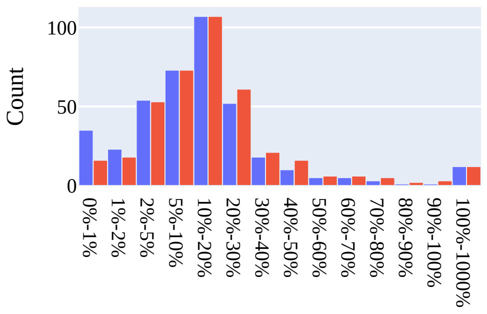
(h) 2022 Comp. ekissat-mab-gb-db.

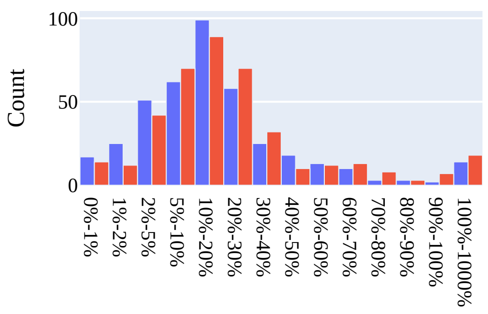
Figure 11 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).

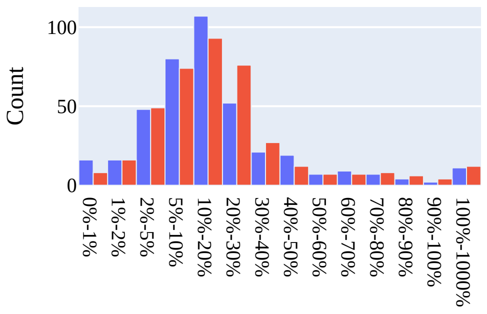

(a) 2022 Comp. glucose-reboot.

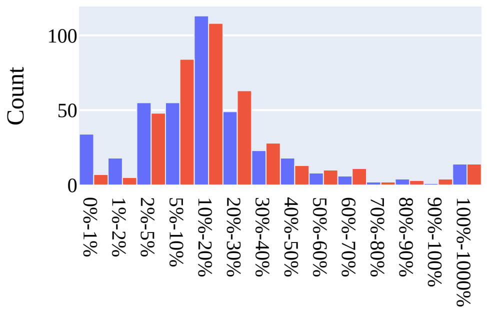

(b) 2022 Comp. hCaD_V1-psids.


(c) 2022 Comp. hCaD_V2.


(d) 2022 Comp. hKis-psids.


(e) 2022 Comp. hKis-sat.


(f) 2022 Comp. hKis-unsat.


(g) 2022 Comp. kissat-els-v1.

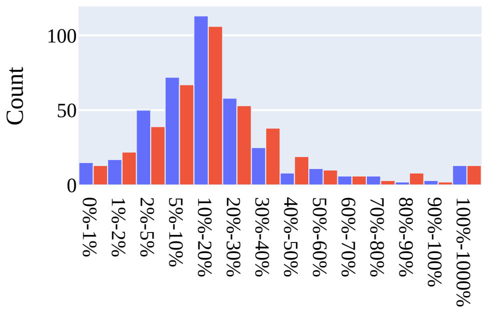
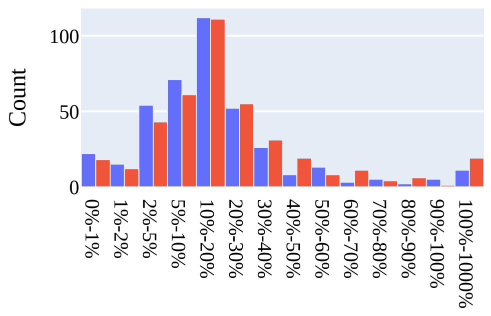
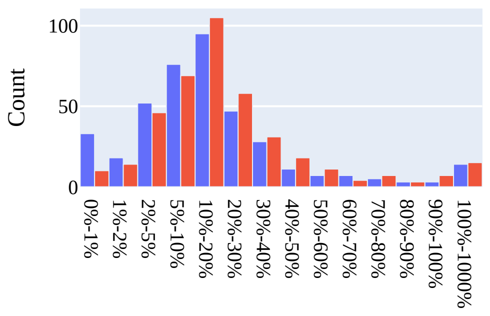

(h) 2022 Comp. kissat-els-v2.

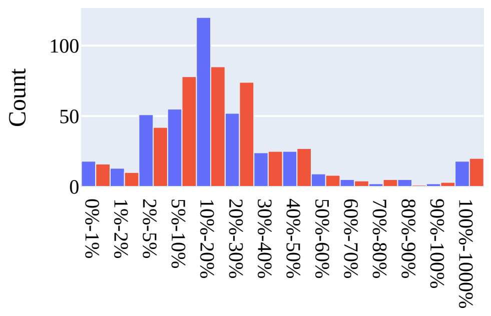
Figure 12 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



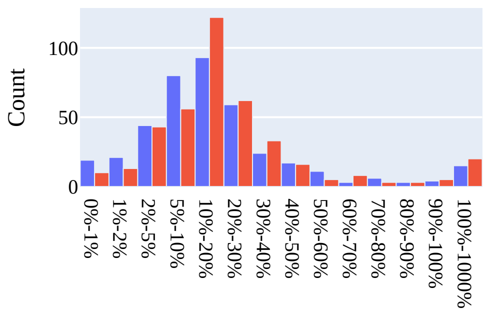
(a) 2022 Comp. kissat-els-v3.



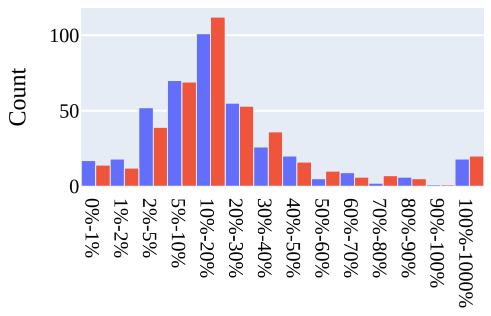
(b) 2022 Comp. kissat-els-v4.



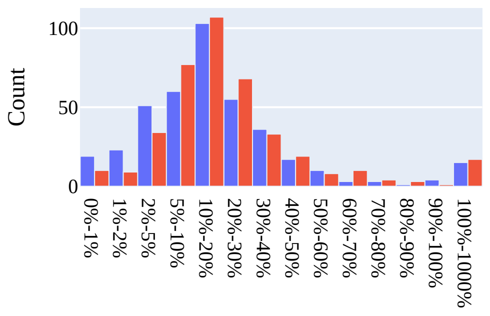
(c) 2022 Comp. kissat-mab-gb.



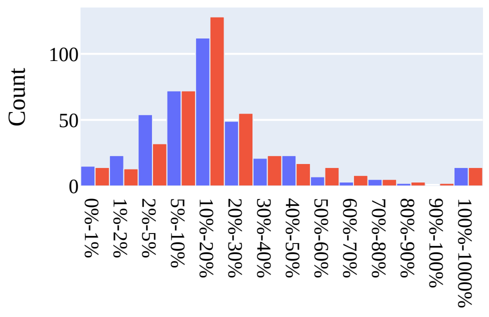
(d) 2022 Comp. kissat-sc2022-bulky.



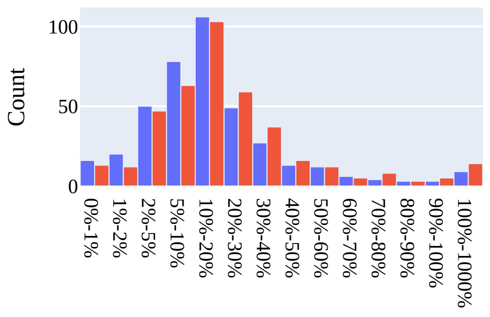
(e) 2022 Comp. kissat-sc2022-hyper.



(f) 2022 Comp. kissat-sc2022-light.

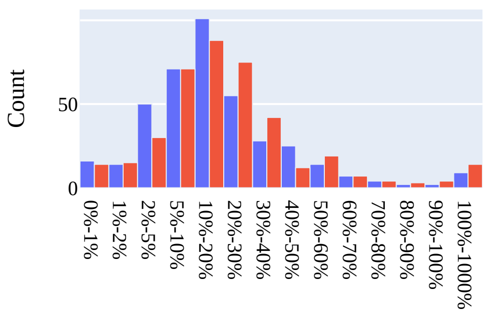


(g) 2022 Comp. kissat-watchsat-lto.

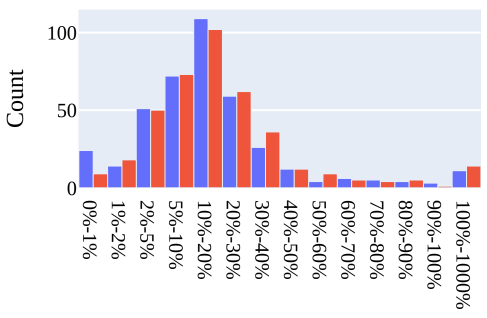


(h) 2022 Comp. kissat_inc.

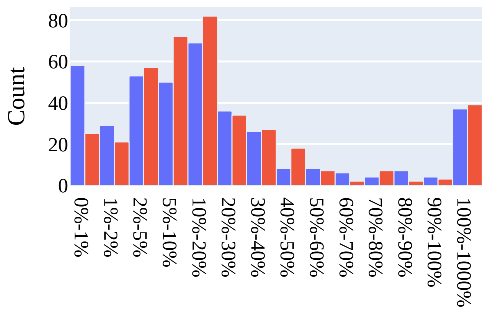
Figure 13 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



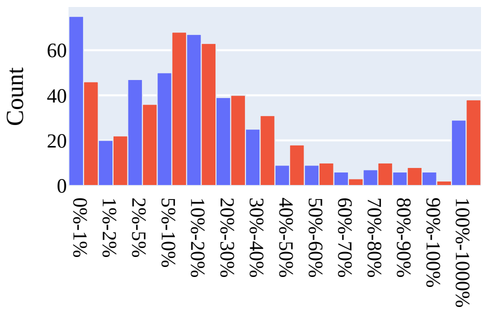
(a) 2022 Comp. kissat_pre.



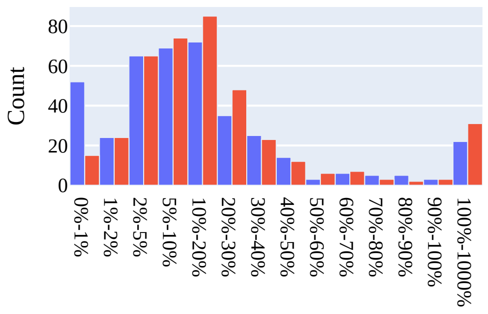
(b) 2022 Comp. kissat_relaxed.



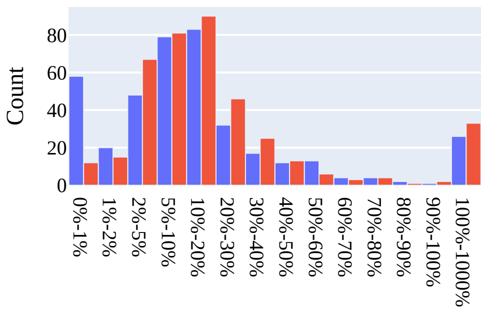
(c) 2023 Comp. AMSAT_.



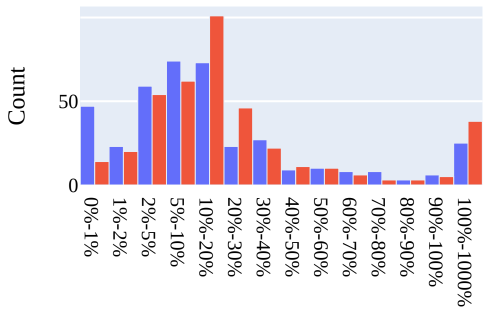
(d) 2023 Comp. BreakID-kissat-low.sh



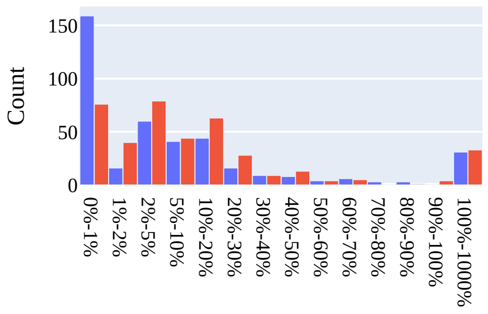
(e) 2023 Comp. CaDiCaL_vivinst.



(f) 2023 Comp. Cadical_ESA.

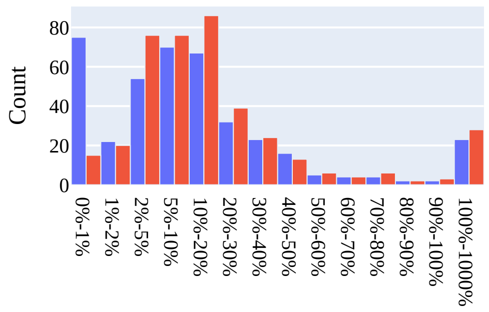


(g) 2023 Comp. Cadical_rel_1.5.3.Scavel.

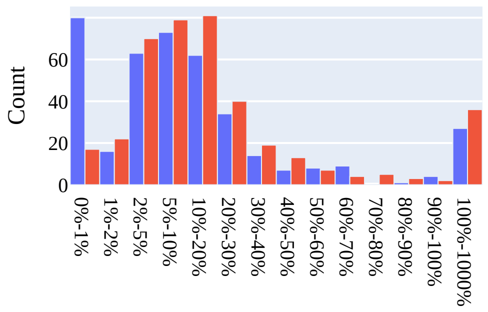


(h) 2023 Comp. IsaSAT.

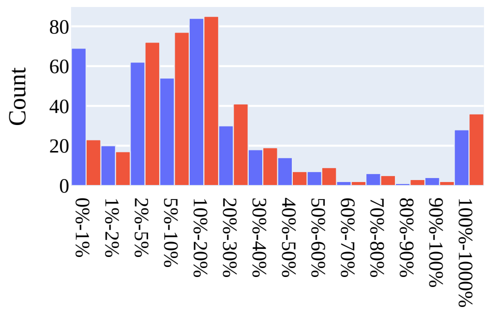
Figure 14 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



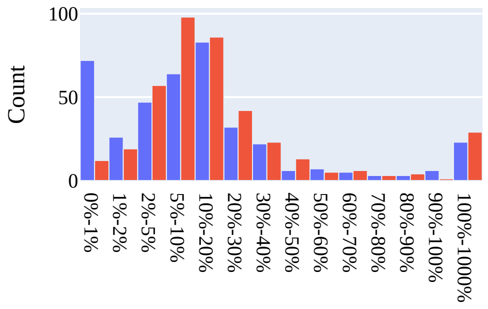
(a) 2023 Comp. Kissat_Inc_ESA.



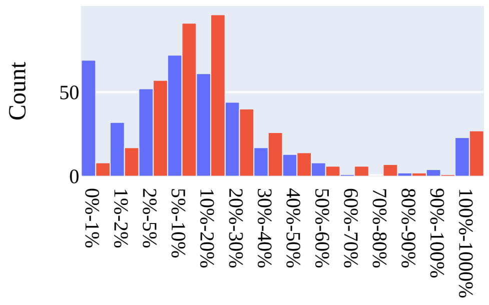
(b) 2023 Comp. Kissat_MAB_Binary.



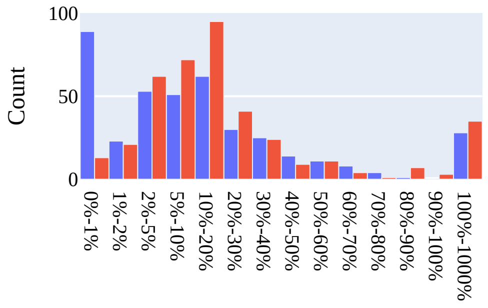
(c) 2023 Comp. Kissat_MAB_Conflict.



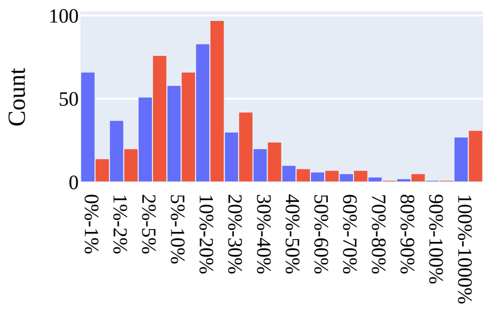
(d) 2023 Comp. Kissat_MAB_Conflict+.



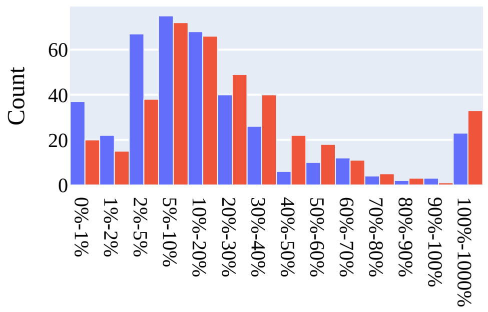
(e) 2023 Comp. Kissat_MAB_DeepWalk+.



(f) 2023 Comp. Kissat_MAB_ESA.

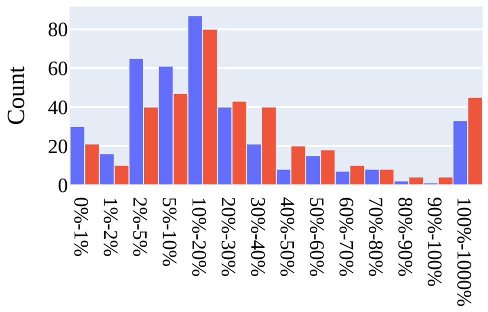


(g) 2023 Comp. Kissat_MAB_Rephases.

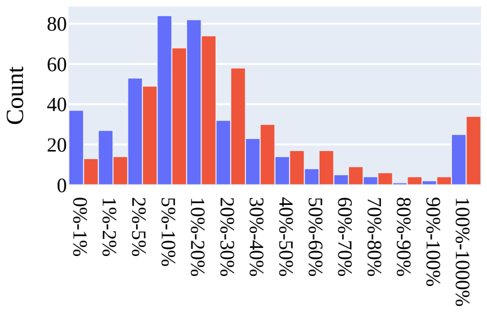


(h) 2023 Comp. Kissat_MAB_prop.

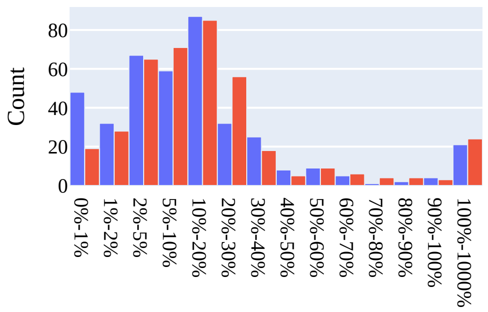
Figure 15 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



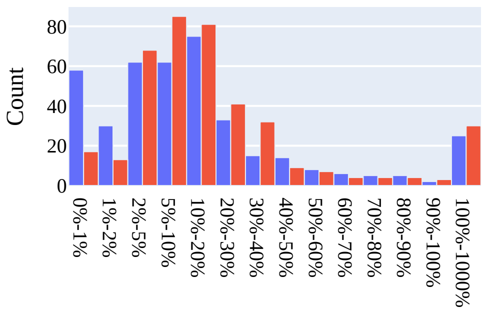
(a) 2023 Comp. Kissat_MAB_prop-no_sym.



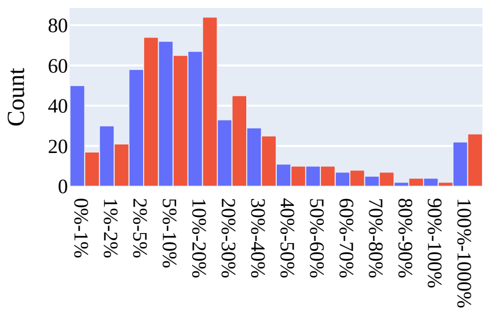
(b) 2023 Comp. Kissat_MAB_prop_pr-no_sym.



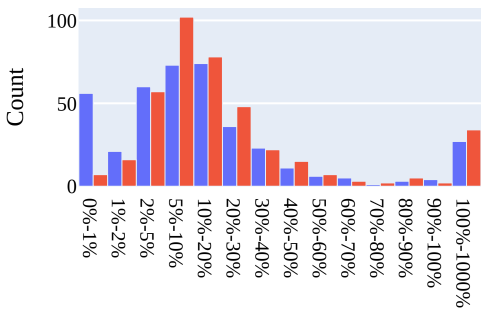
(c) 2023 Comp. MapleCaDiCaL_LBD-990_275.



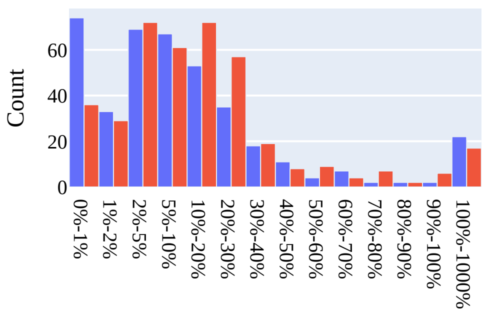
(d) 2023 Comp. MapleCaDiCaL_LBD-990_500.



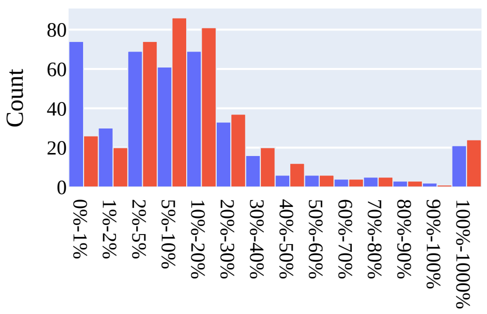
(e) 2023 Comp. MapleCaDiCaL_PPD-500_500.



(f) 2023 Comp. MapleCaDiCaL_PPD-950_950.

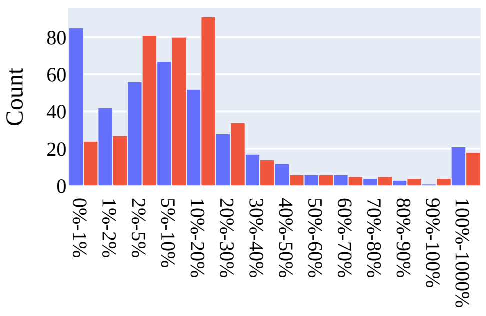


(g) 2023 Comp. MergeSat-bve_gates.

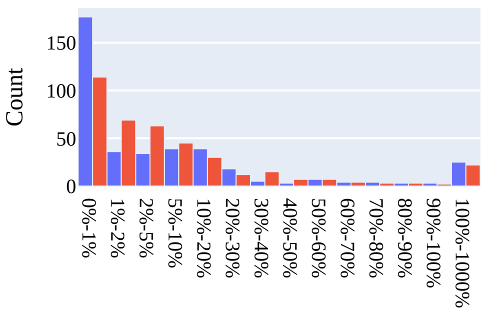


(h) 2023 Comp. MergeSat-bve_semgates.

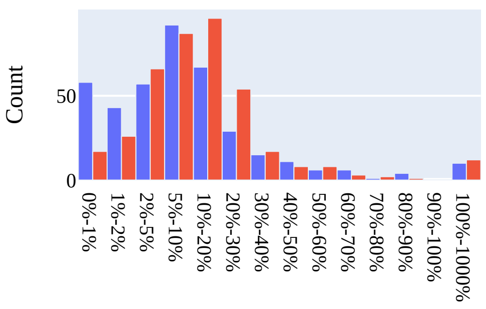
Figure 16 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



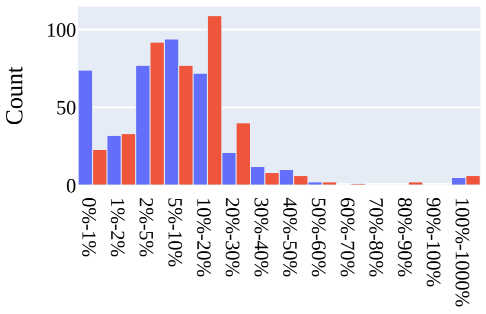
(a) 2023 Comp. MergeSat-thread1.



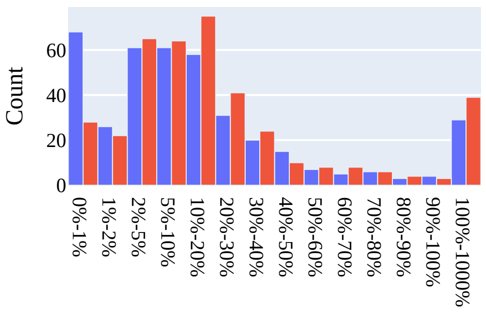
(b) 2023 Comp. MiniSat+XorEngine.



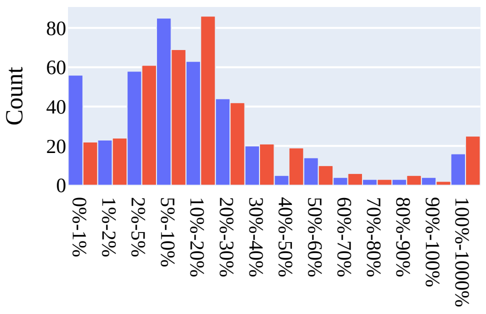
(c) 2023 Comp. PReLearn-kissat-PReLearn-kissat.sh.



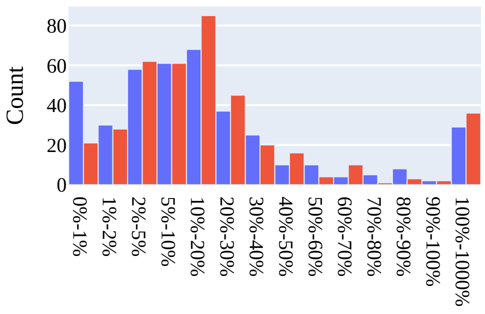
(d) 2023 Comp. PReLearn-kissat-PReLearn-tern-kissat.sh.



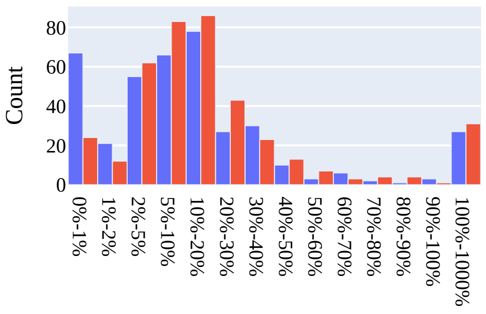
(e) 2023 Comp. ReEncode-kissat-ReEncode-pair-kissat.sh.



(f) 2023 Comp. SBVA-sbva_cadical.

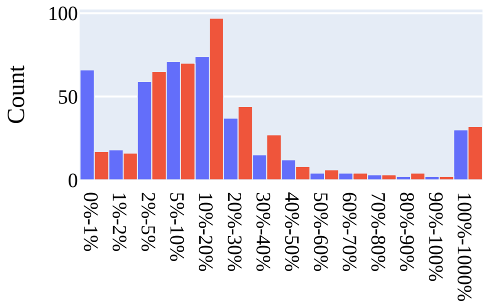


(g) 2023 Comp. SBVA-sbva_kissat.

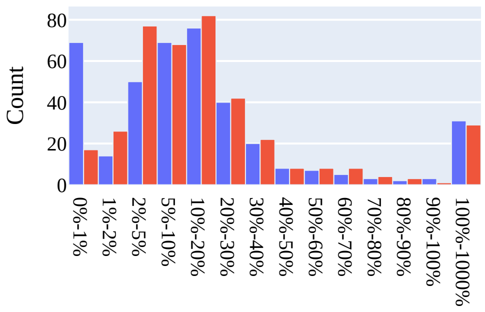


(h) 2023 Comp. SeqFROST.

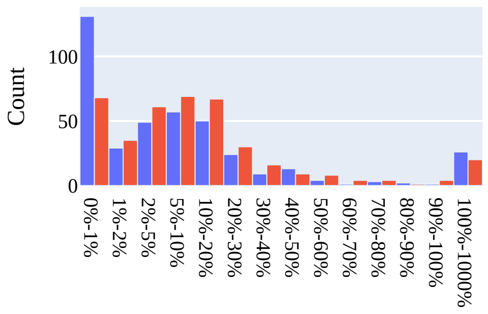
Figure 17 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



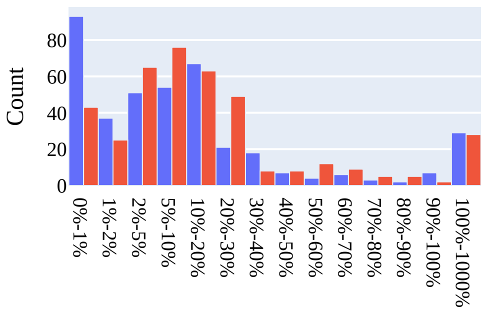
(a) 2023 Comp. SeqFROST-ERE-All.



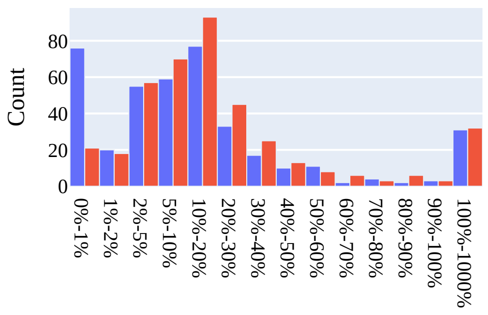
(b) 2023 Comp. SeqFROST-NoExtend.



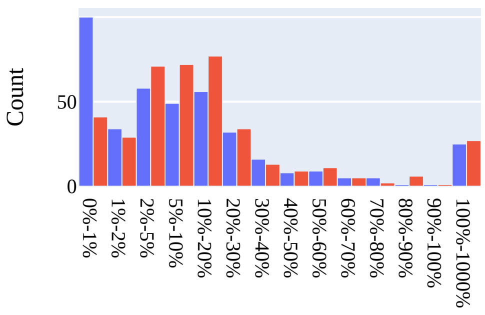
(c) 2023 Comp. hKis-psids.



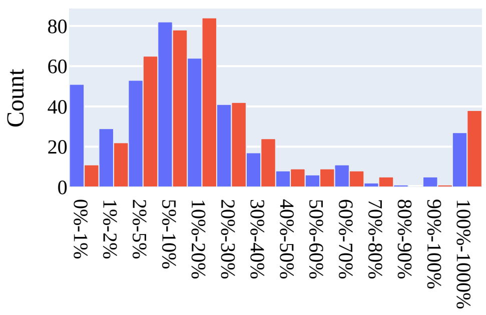
(d) 2023 Comp. hKis-sat_psids.



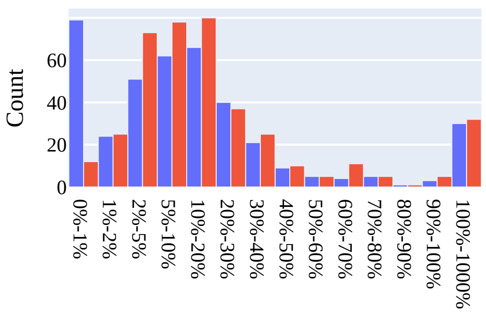
(e) 2023 Comp. hKis-unsat.



(f) 2023 Comp. hKissatInc-unsat.

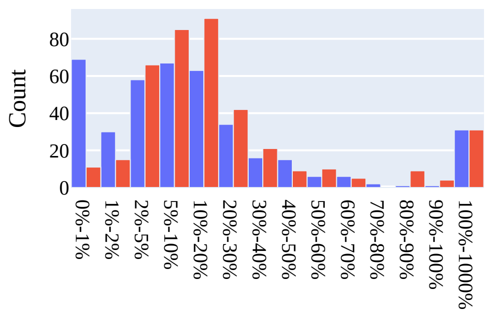


(g) 2023 Comp. kissat-3.1.0.

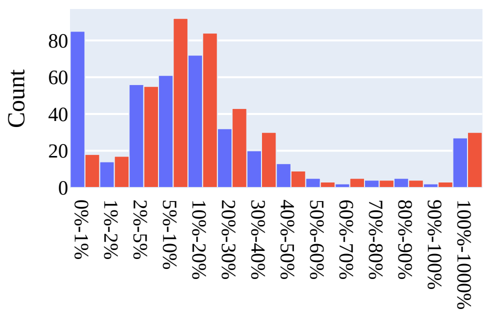


(h) 2023 Comp. kissat-hywalk-exp.

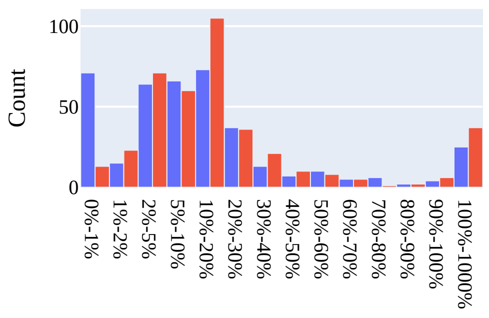
Figure 18 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).



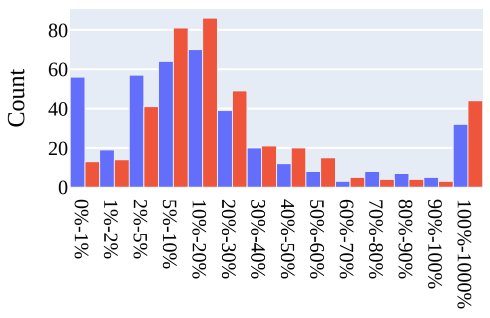
(a) 2023 Comp. kissat-hywalk-exp-gb.



(b) 2023 Comp. kissat-hywalk-gb.



(c) 2023 Comp. kissat_incsp.



(d) 2023 Comp. tabularasat-1.0.0.

Figure 19 Histogram of the error percentage of the root mean square error (RMSE) of (log-transformed) running time prediction using a random forest with features extracted by the old (SATZilla 2012; in red) and the new tool (SATZilla 2024; in blue), on SAT solvers from the 2022 and 2023 SAT Competitions. Results are presented per solver (contd.).