SC-Square: Future Progress with Machine Learning?

Matthew England

Coventry University, Coventry, UK

Abstract

The algorithms employed by our communities are often underspecified, and thus have multiple implementation choices, which do not effect the correctness of the output, but do impact the efficiency or even tractability of its production. In this extended abstract, to accompany a keynote talk at the 2021 SC-Square Workshop, we survey recent work (both the author's and from the literature) on the use of Machine Learning technology to improve algorithms of interest to SC-Square.

Keywords

machine learning, symbolic computation, computer algebra systems, satisfiability checking, SMT solvers

1. Introduction

SC-Square brings together the two communities of Symbolic Computation and Satisfiability Checking, and their associated technologies of Computer Algebra Systems (CASs) and Satisfiability Modulo Theory (SMT) Solvers. One commonality of these communities and technologies is that they value and produce exact, rather than approximate, answers to their problems.

Machine Learning (ML) refers to statistical techniques that give computer systems the ability to *learn* rules from data. It may seem that the probabilistic nature of ML means it is of little interest to SC-Square. However, we suggest there is great potential to use ML to uncover better strategies to optimise SC-Square algorithms and technology.

E.g., consider Buchberger's algorithm to produce the Gröbner Basis for an ideal: a seminal result in Symbolic Computation, used as theory solver for several SMT logics. This algorithm does not specify the order in which S-pairs are studied, the order is which the corresponding S-polynomial is reduced by the generating set, the monomial ordering to be used, and the underlying variable ordering. Any decision for these choices allows the production of a Gröbner Basis but each decision effects the size of the basis produced and the time taken to compute it.

ML may be able to assist with such decisions. However, applying ML to such symbolic algebra and logic is not trivial: there are difficult questions on how to find appropriate data; how to encode that data for ML tools; and which ML paradigm to use. We start this extended abstract by describing our work in Section 2 which attempted to use ML classification to choose the variable ordering for a Computer Algebra algorithm. We then proceed in Section 3 to survey the literature for similar application of ML to mathematics and logic, to look for inspiration to make further progress.

SC^2 2021: 6th International Workshop on Satisfiability Checking and Symbolic Computation, August 19-20 2021

Matthew.England@coventry.ac.uk (M. England)

https://matthewengland.coventry.domains (M. England)

0000-0001-5729-3420 (M. England)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

2. ML Classification for CAD Variable Ordering

2.1. Cylindrical algebraic decomposition

A *Cylindrical Algebraic Decomposition* (CAD) is a *decomposition* of ordered $\mathbb{R}^n$ space into cells arranged *cylindrically*, meaning the projections of cells are all arranged within cylinders. The cells are (semi)-algebraic meaning each may be described by a finite sequence of polynomial constraints. A CAD is usually produced for a set of polynomials such that each polynomial has constant sign on each cell. This allows us to query a finite set of sample points to understand the behaviour of the polynomials (or logical formula involving them) everywhere.

The most important application of CAD is to perform Quantifier Elimination (QE) over the reals: given a quantified formula, find an equivalent quantifier-free formula. E.g., QE would transform $\exists x, ax^2 + bx + c = 0 \wedge a \neq 0$ into the equivalent $b^2 - 4ac \geq 0$. Although CAD emerged in the Symbolic Computation community, since SAT is a sub-problem of QE, it can be used to tackle problems in the NRA and QF_NRA logics of the SMT-LIB. Adaptations of the original CAD algorithm have been designed for use in SMT [25], [27], [1] and we also note the adaptation [4] which is for general QE but with features inspired by Satisfiability Checking.

CAD was introduced in 1975 [11] and is still an active area of research: for a deeper introduction see, for example, the background section of [15].

2.2. CAD variable ordering choice

CAD requires a variable ordering. For QE the ordering must match the quantification, but variables in blocks of the same quantifier and the free variables can be swapped. So in the example above we must decompose (x, a, b, c)-space with x last, but the other variables can be in any order. The ordering can have a great effect on the time / memory use of CAD, the number of cells, and even the underlying complexity [5]. In our example using $a \prec b \prec c$ requires 27 cells but $c \prec b \prec a$ requires 115. Human-designed heuristics [14], [3], [2], [16] usually make the choice in implementations. In 2014, we trained a Support Vector Machine (SVM) to choose which of these heuristic to follow [24]. The SVM significantly outperformed any one heuristic, identifying subclasses where each excelled. This led to an EPSRC project and the work described in the remainder of this section.

2.3. Results from CICM 2019

The 2014 work choose between existing heuristics, in order to fix the number of classes. However, there were many problems in the dataset where none of those heuristics gave an optimal choice. So we revisited these experiments for CICM 2019 [17] this time allowing ML to predict the optimal ordering directly (fixing the number of variables and thus classes). We explored a variety of ML classification methods available in Python's `sklearn` library [34]: K-nearest neighbour classifiers, multi-layer perceptions, decision trees, and support vector machines. We used the CAD implementation in Maple's Regular Chains Library [9]. All the ML models outperformed the human-made heuristics for our dataset.

2.4. Results from SC-Square 2019

The first step to use such an ML classification model is to represent the input (in our case a set of polynomials) as a vector of floating point numbers: the *features*. In [24] and [17] we used measures of degree and frequency of occurrence for each variable, inspired by [3]. Then for SC-Square 2019 [18] we developed a new feature generation procedure which evaluates combinations of basic functions (average, sign, maximum) on the degrees of the variables for individual polynomials and the system. The extra features improved the performance of all the aforementioned ML models. Note that this feature generation procedure can be used for similar classifications where the input is a set of polynomials.

2.5. Results from MACIS 2019

Metrics for judging a CAD variable ordering choice should correspond to CAD runtime[1]. The prior work trained ML classifiers to pick the ordering with minimal runtime for a problem, with selections deemed accurate only if that optimal ordering was chosen. However, this meant that ML training does not distinguish between different non-optimal orderings, even though the differences are often huge. For MACIS 2019 [19] we used an alternative definition of an accurate choice: one leading to a runtime within $x\%$ of the minimum. We then wrote a new version of the `sklearn` cross-validation procedure to select model hyper-parameters to minimise CAD runtime of the choices, rather than maximising the number of times the ordering that gives the minimal time for a problem is taken. This improved the performance of all ML models. Note that the new accuracy definition and procedure are suitable for any classification where we are seeking to have ML make a choice to minimise computation time.

2.6. Software release for ICMS 2020

For ICMS 2020 [20] we presented a software pipeline that implements our work described in the previous sub-sections. Given two datasets (training and testing) the pipeline automates: generation of CAD runtimes for each set of polynomials under each admissible variable ordering; using the runtimes from the training dataset to select the hyper-parameters with cross-validation and tune the parameters of the ML models; and evaluating the performance of those classifiers on the testing dataset. The pipeline could be used to pick the variable ordering for other algorithms which take sets of polynomials as input by changing the calls to Maple's CAD procedure with those of another implementation / algorithm. The code is freely available at: https://doi.org/10.5281/zenodo.3731703.

2.7. Success and limitations

We experimented on the SMT-LIB benchmarks[2] which are mostly real-world applications and extracted two datasets of 3 and 4 variable problems that could be tackled by CAD. On our 3-variable dataset human-made heuristics achieved runtimes 27% above the minimum and ML achieved runtimes 6% above. So here, the ML classifiers offer close to optimal performance.

[1] A hardware independent alternative would be the number of cells produced.
[2] http://smtlib.cs.uiowa.edu/

However, on the 4-variable dataset ML achieved runtime 67% above the minimum (compared to 98% above for human-made heuristics) and so there is room for improvement. Of course, with 4 variables this is a much harder classification problem (24 orderings rather than 6).

To inspire further progress we next consider related work in the literature.

3. Inspiration from the Literature

3.1. Other applications of ML for CAD

The methodology of [24] was applied later to decide the order of sub-formulae solving in [26] and whether to precondition CAD input in [23].

Two more recent works with alternative methodologies are [10] to choose CAD variable ordering and [6] to choose the ordering of constraints to process using the adapted CAD algorithm of [4]. Both papers employed neural networks for the classification and obtained the quantity of data these need through random polynomial generation. We note that care needs to be taken as random polynomials are known to behave quite differently to those which appear in the literature, e.g. [13] and so validation on non-random data should be encouraged. Both papers also took steps to tackle the large number of classes: [10] used an iterative greedy approach to select the ordering; while [6] derived an ordering on the constraints from multiple binary classification on pairs.

Applications of ML elsewhere in Computer Algebra are fairly rare[3] but the following recent one may offer a blueprint for progress.

3.2. Reinforcement learning to optimise Buchberger's algorithm

Buchbergers' Algorithm to produce a Gröbner Basis [7] must process a list of pairs of polynomials (S-pairs); with that processing potentially adding further pairs to the list. Pairs may be processed in any order, but some orders result in more pairs to study and thus more computational resources. There exist well established strategies to make this decision (see e.g. [21]).

In [35] the authors described how an Agent could be trained to make this decision using reinforcement learning: where instead of having a labelled dataset an Agent makes a decision and receives a reward that informs future decisions. In [35] the Agent chooses an S-Pair and received a reward based on the number of polynomial additions required[4].

The study ensures a constant size of polynomial by studying only binomials (so no term swell) and working in a modular coefficient field (so no coefficient swell). They can then represent polynomials to a neural network via consistently sized exponent vectors. Similar to our work in Section 2, this allows the network to judge sparsity and degree but not the actual coefficients (to avoid over-fitting).

[3] We note the early example in [29] which uses a Monte-Carlo tree search to find the representation of polynomials that are most efficient to evaluate.

[4] Actually, the reward is based on the number of polynomial additions required to complete a full run of Buchberger's algorithm after selecting that S-Pair and continuing with a an existing heuristic. The rationale for this given is to reduce variability but it seems equally compelling for allowing the Agent to judge the effects of a choice not just on the next step but on the remainder of the algorithm. This does however greatly increase the training cost.

The experiments in [35] are run on separate distributions of random polynomials based on the number of variables, generators, and degree. The Agent significantly outperformed the established strategies on such data, but the performance on real problems remains to be observed.

Most interestingly, some simple components of the Agent's strategy were observed such as a preference for pairs whose S-polynomials are monomials and a preference for pairs whose S-polynomials are low degree. Such strategies had never been studied[5] and when used alone outperform the established heuristics.

3.3. ML to predict algebraic computation directly

There has been recent work on the use of ML to predict the outcome of algebraic computations directly. Most notably, in [30] the authors predict the output of symbolic integration and the symbolic solutions to first and second order ordinary differential equations using neural networks[6]. Their experiments outperform various CASs, in the sense that the model predicts correct outcomes for examples where the CAS times out. However, we note that from the viewpoint of a CAS developer the cases where the model predicts the wrong model would be more critical than the timeouts[7]. We also note the recent preprint [28] which repeats the study to make the argument that better generalisability will be achieved with a learning model based on the relative positions of mathematical symbols rather than the absolute positions. These are very different applications of ML to the algorithm optimisation we are interested in, but lessons on how best to represent symbolic data to ML tools may well be transferable.

3.4. ML in satisfiability checking

An early use of ML in Satisfiability Checking was the development of the portfolio solver SATZilla [39]. There is rarely a single dominant SAT solver for all problems, so SATZilla uses ML to predict which solver to use for a given instance. This inspired the recent MachSMT which selects algorithms for SMT-solvers [36] using Principal Component Analysis for dimensionality reduction.

The core algorithm of Satisfiability Checking, CDCL [33], allows us to proceed through the exponential search space in an intelligent manner: generalising from the conflicts uncovered at a specific sample to rule out additional branches. However, even with this conflict learning, there are decisions in the search that must be taken without guidance and poor luck can lead a search to an unproductive area, motivating for example the use of restarts.
Thus CDCL itself has potential to be guided by ML. For example:

- [38] makes the choice of initial value to variable allocation to begin the search using a regression model that predicted satisfiability of formulae after fixing the values of a certain fraction of the variables.

[5] Perhaps because S-polynomials are rarely examined before being reduced.
[6] Specifically `seq2seq` models more typically used in natural language processing: for example they view integration as translating from integrands to integrals
[7] The review [12] offer some other qualifications on the claim of superiority over CASs.

- [32] uses machine learning to determine a policy for restarts in SAT-solvers: ML is used to predict the quality of the next learnt clause based on previously learnt clauses; restarting when the quality is predicted below a threshold.
- NeuroSAT [37] can predict unsatisfiable cores (subsets of the constraints that cannot be satisfied together) to inform variable selection in the search.

The most prominent use of ML in satisfiability is probably the following one.

3.5. Reinforcement learning for SAT-solver branching

The MapleSAT solver introduced and utilises the learning rate, the propensity for variables to generate learnt clauses, as a key metric for making decisions and the first to outperform the previously dominant VSIDS heuristic.

In [31] they view the question of branching in SAT-solving (selecting the next variable in the search) as an optimisation problem to maximize this metric. In particular, they apply reinforcement learning, where the learning rate informs the reward function. Variable selection is modelled in the multi-armed bandit (MAB) framework and tackled using a well known MAB algorithm. This led MapleSAT to victory in the annual SAT competition[8].

3.6. ML to predict mathematical structure

Finally, we note also the use of supervised ML to predict mathematical properties elsewhere in mathematics, where the primary motivation is the formation of new conjectures. We refer to the survey [22] which includes examples in algebraic geometry, representation theory, combinatorics and number theory, in which most applications are expressed as ML classification problems.

3.7. Summary

There is huge potential to apply ML to algorithms of interest to SC-Square. The author's own experience in Section 2 shows the potential benefits.

However, our experience using ML classification has clear limits. Such supervised learning requires labelled datasets. Although in theory infinite symbolic data could be manufactured, in practice it would be computationally infeasible to label all that data. A reinforcement learning approach such as for the examples in Sections 3.2 and 3.5 looks far more promising.

Still unclear is the optimal way to represent symbolic data for ML tools, and how best to generate training data so maximise generalisation onto the problems of interest in the real world. Such questions deserve more focussed study.

[8] https://baldur.iti.kit.edu/sat-competition-2016/index.php?cat=results

Acknowledgements

The author's work surveyed in Sections 2.3—2.7 was joint with Dorian Florescu, and funded by EPSRC Project EP/R019622/1: *Embedding Machine Learning within Quantifier Elimination Procedures*. The author is now supported by EPSRC Project EP/T015748/1: *Pushing Back the Doubly-Exponential Wall of Cylindrical Algebraic Decomposition*.

We thank the reviewer of this paper, and the reviewers of the author's surveyed work, for useful comments. We thank the DEWCAD Journal Club for interesting discussions on some of the other papers.

References

[1] Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. Journal of Logical and Algebraic Methods in Programming **119**, 100633 (2021), https://doi.org/10.1016/j.jlamp.2020.100633

[2] Bradford, R., Davenport, J.H., England, M., Wilson, D.: Optimising problem formulations for cylindrical algebraic decomposition. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) Intelligent Computer Mathematics, Lecture Notes in Computer Science, vol. 7961, pp. 19–34. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-39320-4_2

[3] Brown, C.W.: Companion to the tutorial: Cylindrical algebraic decomposition, presented at ISSAC '04. URL http://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf (2004)

[4] Brown, C.W.: Open non-uniform cylindrical algebraic decompositions. In: Proceedings of the 2015 International Symposium on Symbolic and Algebraic Computation. pp. 85–92. ISSAC '15, ACM (2015), https://doi.org/10.1145/2755996.2756654

[5] Brown, C.W., Davenport, J.H.: The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation. pp. 54–60. ISSAC '07, ACM (2007), https://doi.org/10.1145/1277548.1277557

[6] Brown, C.W., Daves, G.C.: Applying machine learning to heuristics for real polynomial constraint solving. In: Bigatti, A., Carette, J., Davenport, J.H., Joswig, M., de Wolff, T. (eds.) Mathematical Software – ICMS 2020. Lecture Notes in Computer Science, vol. 12097, pp. 292–301. Springer International Publishing (2020), https://doi.org/10.1007/978-3-030-52200-1_29

[7] Buchberger, B.: Bruno Buchberger's PhD thesis (1965): An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. Journal of Symbolic Computation **41**(3-4), 475–511 (2006), https://doi.org/10.1016/j.jsc.2005.09.007

[8] Caviness, B., Johnson, J.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts & Monographs in Symbolic Computation, Springer-Verlag (1998), https://doi.org/10.1007/978-3-7091-9459-1

[9] Chen, C., Moreno Maza, M.: Quantifier elimination by cylindrical algebraic decomposition

based on regular chains. Journal of Symbolic Computation **75**, 74–93 (2016), https://doi.org/10.1016/j.jsc.2015.11.008

[10] Chen, C., Zhu, Z., Chi, H.: Variable ordering selection for cylindrical algebraic decomposition with artificial neural networks. In: Bigatti, A., Carette, J., Davenport, J.H., Joswig, M., de Wolff, T. (eds.) Mathematical Software – ICMS 2020. Lecture Notes in Computer Science, vol. 12097, pp. 281–291. Springer International Publishing (2020), https://doi.org/10.1007/978-3-030-52200-1_28

[11] Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages. pp. 134–183. Springer-Verlag (reprinted in the collection [8]) (1975), https://doi.org/10.1007/3-540-07407-4_17

[12] Davis, E.: The use of deep learning for symbolic integration: A review of (Lample and Charton, 2019). Unpublished, available as ArXiv:2105.11479 (2019), https://arxiv.org/abs/2105.11479

[13] Dembo, A., Poonen, B., Shao, Q., Zeitouni, O.: Random polynomials having few or no real zeros. Journal of the American Mathematical Society **15**(4), 857–892 (2002), https://www.ams.org/journals/jams/2002-15-04/S0894-0347-02-00386-7/S0894-0347-02-00386-7.pdf

[14] Dolzmann, A., Seidl, A., Sturm, T.: Efficient projection orders for CAD. In: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation. pp. 111–118. ISSAC '04, ACM (2004), https://doi.org/10.1145/1005285.1005303

[15] England, M., Bradford, R., Davenport, J.H.: Cylindrical algebraic decomposition with equational constraints. Journal of Symbolic Computation **100**, 38–71 (2020), https://doi.org/10.1016/j.jsc.2019.07.019

[16] England, M., Bradford, R., Davenport, J.H., Wilson, D.: Choosing a variable ordering for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In: Hong, H., Yap, C. (eds.) Mathematical Software – ICMS 2014. Lecture Notes in Computer Science, vol. 8592, pp. 450–457. Springer Heidelberg (2014), http://dx.doi.org/10.1007/978-3-662-44199-2_68

[17] England, M., Florescu, D.: Comparing machine learning models to choose the variable ordering for cylindrical algebraic decomposition. In: Kaliszyk, C., Brady, E., Kohlhase, A., Sacerdoti, C.C. (eds.) Intelligent Computer Mathematics. Lecture Notes in Computer Science, vol. 11617, pp. 93–108. Springer International Publishing (2019), https://doi.org/10.1007/978-3-030-23250-4_7

[18] Florescu, D., England, M.: Algorithmically generating new algebraic features of polynomial systems for machine learning. In: Abbott, J., Griggio, A. (eds.) Proceedings of the 4th Workshop on Satisfiability Checking and Symbolic Computation (SC^2 2019). No. 2460 in CEUR Workshop Proceedings (2019), http://ceur-ws.org/Vol-2460/

[19] Florescu, D., England, M.: Improved cross-validation for classifiers that make algorithmic choices to minimise runtime without compromising output correctness. In: Slamanig, D., Tsigaridas, E., Zafeirakopoulos, Z. (eds.) Mathematical Aspects of Computer and Information Sciences (Proc. MACIS '19). Lecture Notes in Computer Science, vol. 11989, pp. 341–356. Springer International Publishing (2020), https://doi.org/10.1007/978-3-030-43120-4_27

[20] Florescu, D., England, M.: A machine learning based software pipeline to pick the variable ordering for algorithms with polynomial inputs. In: Bigatti, A., Carette, J., Davenport,

J.H., Joswig, M., de Wolff, T. (eds.) Mathematical Software – ICMS 2020. Lecture Notes in Computer Science, vol. 12097, pp. 302–322. Springer International Publishing (2020), https://doi.org/10.1007/978-3-030-52200-1_30

[21] Giovini, A., Mora, T., Niesi, G., Robbiano, L., Traverso, C.: One sugar cube, please; or selection strategies in the Buchberger algorithm. In: Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation. pp. 49–54. ISSAC '91, ACM (1991), https://doi.org/10.1145/120694.120701

[22] He, Y.H.: Machine-learning mathematical structures. International Journal of Data Science in the Mathematical Sciences **1**(1), 1–25 (2022), https://doi.org/10.1142/S2810939222500010

[23] Huang, Z., England, M., Wilson, D., Bridge, J., Davenport, J.H., Paulson, L.: Using machine learning to improve cylindrical algebraic decomposition. Mathematics in Computer Science **13**(4), 461–488 (2019), https://doi.org/10.1007/s11786-019-00394-8

[24] Huang, Z., England, M., Wilson, D., Davenport, J.H., Paulson, L., Bridge, J.: Applying machine learning to the problem of choosing a heuristic to select the variable ordering for cylindrical algebraic decomposition. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) Intelligent Computer Mathematics, Lecture Notes in Artificial Intelligence, vol. 8543, pp. 92–107. Springer International (2014), http://dx.doi.org/10.1007/978-3-319-08434-3_8

[25] Jovanovic, D., de Moura, L.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning: 6th International Joint Conference (IJCAR), Lecture Notes in Computer Science, vol. 7364, pp. 339–354. Springer (2012), https://doi.org/10.1007/978-3-642-31365-3_27

[26] Kobayashi, M., Iwane, H., Matsuzaki, T., Anai, H.: Efficient subformula orders for real quantifier elimination of non-prenex formulas. In: Kotsireas, S.I., Rump, M.S., Yap, K.C. (eds.) Mathematical Aspects of Computer and Information Sciences (MACIS '15). Lecture Notes in Computer Science, vol. 9582, pp. 236–251. Springer International Publishing (2016), https://doi.org/10.1007/978-3-319-32859-1_21

[27] Kremer, G., Ábrahám, E.: Fully incremental CAD. Journal of Symbolic Computation **100**, 11–37 (2020), https://doi.org/10.1016/j.jsc.2019.07.018

[28] Kubota, H., Tokuoka, Y., Yamada, T.G., Funahashi, A.: Symbolic integration by integrating learning models with different strengths and weaknesses. IEEE Access **10**, 47000–47010 (2022), https://doi.org/10.1109/ACCESS.2022.3171329

[29] Kuipers, J., Ueda, T., Vermaseren, J.A.M.: Code optimization in FORM. Computer Physics Communications **189**, 1–19 (2015), https://doi.org/10.1016/j.cpc.2014.08.008

[30] Lample, G., Charton, D.: Deep learning for symbolic mathematics. In: Mohamed, S., White, M., Cho, K., Song, D. (eds.) Eighth International Conference on Learning Representations (ICLR 2020) (2020), https://iclr.cc/virtual_2020/poster_S1eZYeHFDS.html

[31] Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: Creignou, N., Le Berre, D. (eds.) Theory and Applications of Satisfiability Testing – SAT 2016, Lecture Notes in Computer Science, vol. 9710, pp. 123–140. Springer International Publishing (2016), https://doi.org/10.1007/978-3-319-40970-2_9

[32] Liang, J.H., Oh, C., Mathew, M., Thomas, C., Li, C., Ganesh, V.: Machine learning-based restart policy for CDCL SAT solvers. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) Theory and Applications of Satisfiability Testing – SAT 2018, Lecture Notes in Computer Science,

vol. 10929, pp. 94–110. Springer International Publishing (2018), https://doi.org/10.1007/978-3-319-94144-8_6

[33] Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. Computers, IEEE Transactions on **48**(5), 506–521 (1999), https://doi.org/10.1109/12.769433

[34] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011), http://www.jmlr.org/papers/v12/pedregosa11a.html

[35] Peifer, D., Stillman, M., Halpern-Leistner, D.: Learning selection strategies in Buchberger's algorithm. In: Daumé III, H., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning (ICML 2020). Proceedings of Machine Learning Research, vol. 119, pp. 7575–7585. PMLR (2020), https://proceedings.mlr.press/v119/peifer20a.html

[36] Scott, J., Niemetz, A., Preiner, M., Nejati, S., Ganesh, V.: MachSMT: A machine learning-based algorithm selector for SMT solvers. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 12652, pp. 303–325. Springer International Publishing (2021), https://doi.org/10.1007/978-3-030-72013-1_16

[37] Selsam, D., Bjørner, N.: Guiding high-performance SAT solvers with unsat-core predictions. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing – SAT 2019. Lecture Notes in Computer Science, vol. 11628, pp. 336–353. Springer International Publishing (2019), https://doi.org/10.1007/978-3-030-24258-9_24

[38] Wu, H.: Improving SAT-solving with machine learning. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. pp. 787–788. SIGCSE '17, ACM (2017), https://doi.org/10.1145/3017680.3022464

[39] Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. Journal Of Artificial Intelligence Research **32**, 565–606 (2008), https://doi.org/10.1613/jair.2490