Learning Variable Activity Initialisation for Lazy Clause Generation Solvers

Ronald van Driel, Emir Demirović, and Neil Yorke-Smith(✉)

Algorithmics, Delft University of Technology, Delft, Netherlands
R.A.vanDriel@student.tudelft.nl, {e.demirovic,n.yorke-smith}@tudelft.nl

Abstract. Contemporary research explores the possibilities of integrating machine learning (ML) approaches with traditional combinatorial optimisation solvers. Since optimisation hybrid solvers, which combine propositional satisfiability (SAT) and constraint programming (CP), dominate recent benchmarks, it is surprising that the literature has paid limited attention to machine learning approaches for hybrid CP–SAT solvers. We identify the technique of *minimal unsatisfiable subsets* as promising to improve the performance of the hybrid CP–SAT lazy clause generation solver Chuffed. We leverage a graph convolutional network (GCN) model, trained on an adapted version of the MiniZinc benchmark suite. The GCN predicts which variables belong to an unsatisfiable subset on CP instances; these predictions are used to initialise the activity score of Chuffed's Variable-State Independent Decaying Sum (VSIDS) heuristic. We benchmark the ML-aided Chuffed on the MiniZinc benchmark suite and find a robust 2.5% gain over baseline Chuffed on MRCPSP instances. This paper thus presents the first, to our knowledge, successful application of machine learning to improve hybrid CP–SAT solvers, a step towards improved automatic solving of CP models.

1 Introduction

Neuro-symbolic approaches to combinatorial optimisation problems include improving optimisation solver performance or robustness by incorporating machine learning (ML). This trend shows successful promise in integer programming [2,10,14,26], propositional satisfiability (SAT) [22,25] as well as constraint programming (CP) [1,9,24]. Hybrid CP–SAT solvers are the state of the art for CP according to recent MiniZinc Challenge competitions [19]. Such solvers, labelled as *Lazy Clause Generation* (LCG) solvers [21], combine the conflict learning ability from SAT solvers with finite domain propagation from CP solvers.

However to the best of our knowledge there have not been research to date on combining machine learning to improve the performance of hybrid CP–SAT solvers. For example, Song et al. [24] show that machine learning can be used to automatically learn variable ordering heuristics for traditional constraint satisfaction solving. Portfolio approaches have shown excellent performance [1,13], but such methods select one of the solving strategies in a portfolio rather than

P. J. Stuckey (Ed.): CPAIOR 2021, LNCS 12735, pp. 62–71, 2021.
https://doi.org/10.1007/978-3-030-78230-6_4

directly modify a LCG solver. Similarly, applications that combine machine learning and constraint programming have been studied (e.g., [5]), but the machine learning part does not influence the internal hybrid CP–SAT algorithm.

We aim to utilise machine learning to improve a single component of hybrid CP–SAT solvers, namely the activity-based variable selection heuristic (VSIDS). Our approach is motivated by *Neurocore* [23], a method that uses ML to influence the variable selection of SAT solvers. Given that LCG solvers use SAT solvers in their inner-workings, a natural question to ask is whether an approach such as Neurocore can be employed in constraint programming. While related ideas may be exploited, a direct application of Neurocore in CP is not possible. Neurocore trains its learned model on clauses derived from the proof of unsatisfiability. However, unsatisfiability proofs are not an established concept in CP. Current SAT techniques are not easily extendable, as CP considers *optimisation* problems, possibly using integer variables, and complex constraints that may require an exponential number of clauses when encoding into SAT. There has been progress in this direction using cutting planes reasoning, but only for specific problems or constraints [7,11,12]. For similar reasons, the machine learning features used in SAT may not directly translate to (LCG-based) CP.

This paper provides a first demonstration of the value of using ML within the LCG solver Chuffed [4]. We develop a modified version of Neurocore for constraint programming and employ it to learn initialisation values for the activities used in the variable-selection heuristic. We benchmark our ML-aided approach on problems from the MiniZinc benchmark suite and find a statistically-significant 2.5% average gain over the baseline Chuffed on MRCPSP instances.

2 Background

The Satisfiability problem (SAT) is concerned with deciding whether or not there exists an assignment of truth values to variables such that a given propositional logic formula is satisfied. A *SAT solver* is an algorithm that explores the space of possible assignments with the aim of either finding a satisfying assignment or proving that the formula is unsatisfiable. For the purposes of this paper, the search may be viewed as a backtracking algorithm over the variable assignments.

The *Variable-State Independent Decaying Sum* (VSIDS) [17,20], originally developed as a variable selection heuristic for SAT solver Chaff, is commonly used in LCG solvers. When using VSIDS in a SAT solver during search, variables are selected according to their *activity*. Intuitively, the activity score indicates the likelihood that the variable will quickly lead to a conflict, and selecting such variables early in the search is beneficial. Initially, the activity value of each variable is initialised to zero. Once the solver encounters a *conflict*, i.e., it is detected that the current partial assignment is infeasible, analysis is performed to determine the reason for the conflict. The reason is recorded as a *learned clause*, which consists of a subset of the variables from the partial assignment. Each time a variable is involved in a conflict, its activity is increased. To emphasise recent conflicts, the activity scores of all variables are periodically non-linearly

decreased. As a result, variables recently involved in conflicts have the highest scores. LCG solvers make use of VSIDS in their internal SAT solver.

SAT solvers, upon concluding that a problem is unsatisfiable, may provide a *certificate of unsatisfiability*. Intuitively, the certificate consists of a set of clauses and a sequence of logic derivation steps result that in the empty clause, i.e., unsatisfiability. A related concept in CP is a *minimal unsatisfiable subset*, which is a set of constraints that unsatisfiable together, but are not unsatisfiable if any constraint is removed from the set. Conceptually, SAT solvers operate on the low-level of propositional logic, whereas CP solvers consider a more expressive CP setting, e.g., complex constraints over integer variables. Hybrid CP–SAT solvers [21] maintain a dual view of the problem: in addition to the CP view, a portion of the problem is converted into propositional logic. An internal SAT solver is invoked on the propositional logic formula, augmented with CP propagators to infer variable assignments based on the current partial assignment. Once a conflict is encountered, the conflict analysis procedure from SAT operates as usual, with the exception that variables set by propagators are queried to provide the reason for their propagation in the form of a *clause*. Since all reasons are clausal, this allows the solver to use the SAT conflict analysis procedure while still retaining the benefit of CP. In this way, hybrid CP–SAT solvers combine SAT and CP solving techniques.

The *Neurocore* [23] approach uses machine learning to influence the variable selection heuristic of a SAT solver. Since a SAT solver may make thousands of decisions per second using VSIDS, a possible replacement of variable selection is expected to run with a tight time budget. Hence Neurocore does not *directly* use ML to replace the variable selection heuristic, but instead *indirectly* influences the selection procedure by periodically modifying the activity values of the variables. The ML model, represented as a graph convolutional network (GCN), is trained to assign a confidence value between zero and one for each variable depending on its features. The estimate represents the probability that the variable is part of an unsatisfiable core. The first assumption is that variables that are used in the proof of unsatisfiability are likely to quickly lead to conflicts during search, and therefore the solver should aim to select these variables as soon as possible. The second assumption is that, even though unsatisfiable cores do not exist in satisfiable instances, the GCN predictions will nevertheless be valuable even for satisfiable instances to identify highly conflict-inducing variables.

3 Approach

Recall our goal is to predict the initial values of variable activity for a CSP instance. Since it is difficult to formulate directly learning VSIDS initialisations as a feasible learning problem (as discussed later), instead we leverage the analogous precedent in SAT solving discussed above [23].

By default, the activity values in LCG solvers are set to zero or to random values at the start of the search. The scores do not provide any meaningful information to the solver in the beginning but they gradually become more

useful as search proceeds. By providing useful initial values we posit that the solver performance can be improved; improvements at the start of search are particularly valuable. In the absence of meaningful VSIDS values, *Chuffed* [4], the LCG solver used in this work, typically uses (user-specified) search annotations if provided, before switching to VSIDS for making branching decisions.

Our approach is to train a graph convolutional network model on *unsatisfiable* instances, to make a prediction on which variables belong to an unsatisfiable subset. The trained model is then used as part of the LCG solver to classify the variables of input instances at the start of the search. The classification is done by assigning a value between zero and one for each variable, which may be interpreted as the probability that the variable is in an unsatisfiable subset. These values are used to initialise the activity values for VSIDS.

Whereas training is done on unsatisfiable instances, the target instances used afterwards do not necessarily need to be unsatisfiable, e.g., it is expected the instances represent optimisation problems for which a feasible solution exists. Note that for satisfiable instances, no unsatisfaible subset exists, but the predictions made by the network are still valuable since, intuitively, higher predicted values indicate variables that are more likely to engage in a conflict.

It is important to note that, similar to the approach proposed by Selsam and Bjørner [23] – and with works in the predict-and-optimise paradigm [6,8] – our ambition is not to achieve the best possible ML predictions. The reason for this is that more accurate predictions do not necessarily imply that they are more useful for the solver; rather the metric to optimise is the runtime of the solver. The hypothesis is that, even though satisfiable instances do not have unsatisfiable cores, the *confidence* of classifying a variable to be part of an unsatisfiable set correlates with the effectiveness of branching on that variable. This can be seen as a surrogate for the runtime. The *true* metric that directly optimises the runtime remains an open question.

An alternative could be to learn based on the final VSIDS scores. However such scores are biased towards the last few conflicts before termination even though many other conflicts were needed to prove optimality. On a related note, in core-boosted MAxSAT [3], after the core-guided (lower bounding) phase, it was beneficial to *nullify* the VSIDS scores before switching to the linear search (upper bounding) phase, as opposed to keeping the final VSIDS scores of the lower bounding method, indicating that VSIDS scores that are good for one phase of the search may not be good for another phase.

3.1 Machine Learning Model

We adopt the Graph Convolution Network (GCN) model of Kipf and Welling [15].[1] A GCN learns a function of the features on a graph: in our case the constraint graph. The features we choose pertain to the variables: 1. Categorical features indicating if a variable is declared as a Boolean, integer, float or set. 2. Minimum value within the variable domain. 3. Maximum value within the

[1] Code available at https://github.com/tkipf/gcn; we use their default settings.

variable domain. 4. The range of the variable domain. 5. A set of identifiers of variables which co-occur in some constraint. Then the input of the GCN is:

1. **A feature matrix of size** $N \times D$. Here N represents the number of variables and D the number of selected features.
2. **An adjacency matrix of size** $N \times N$. In this matrix variables are considered adjacent if they co-occur in a constraint.
3. **The labels in an** $N \times C$ **matrix.** Here C represents the number of output classes, in our case two: one for variables which are part of a minimal unsatisfiable subset (MUS) and the other for variables which are not.

The output of the model is a $N \times C$ matrix, the softmax outputs – which can be interpreted as the probability for each variable to belonging to each class. Because we consider two classes only, it is possible to express the output of the ML predictions with a single value, which is the prediction confidence of a variable belonging to a MUS.

4 Empirical Study

We now examine experimentally the effectiveness of the proposed approach. We compiled from source three different versions of Chuffed: Chuffed0_OG, Chuffed1_Ex and Chuffed1_Inc. All three versions were configured to switch to VSIDS as soon as 100 conflicts have been encountered.[2] While all three versions have an identical configuration, they are different in the way the ML was integrated. Chuffed0_OG was otherwise left completely unmodified, and serves as a baseline. Chuffed1_Ex was modified to have the VSIDS scores initialised with the predictions obtained after being trained on a training set which contained only instances from *other* problem types. Similarly, Chuffed1_Inc was modified to initialise the VSIDS scores with predictions after being trained on *all* training instances, including from the same problem type.

4.1 Data Sets

We require two different datasets containing CP instances. One of these datasets should only contain unsatisfiable instances to train on; the other should contain satisfiable instances to solve for evaluation. The MiniZinc benchmark suite [18] supplies over 13,000 satsifiable instances for evaluation. Since we found no public CP dataset contained sufficiently many unsatisfiable instances for training a ML model, the constraint optimisation problem (COP) instances from the MiniZinc benchmark suite were modified to become unsatisfiable. This was done by first solving them for their optimal value; then the original instance was modified by bounded the objective variable to be strictly better than the optimal value.

Using this procedure allows the creation both the satisfiable datset as well as the unsatisfiable dataset. For the unsatisfiable dataset the labels were generated

[2] This is lower than the Chuffed default, in order to ensure that VSIDS is used.

Table 1. Experiments on MRCPSP benchmarks

Instances	Chuffed0_OG	Chuffed1_Ex	Chuffed1_Inc
	Avg. runtime (s)	Avg. runtime (s)	Avg. runtime (s)
mrcpsp10900	4.507	4.356	4.461
mrcpsp36	2.399	2.428	2.410
mrcpsp4425	311.565	296.139	302.595
mrcpsp4777	5274.736	5153.284	5155.367
mrcpsp4871	892.922	865.954	865.404
mrcpsp4960	32.713	32.241	32.099
mrcpsp7051	16.091	15.884	16.028
mrcpsp896	0.152	0.155	0.189
mrcpsp9880	0.236	0.241	0.240
mrcpsp9994	0.033	0.034	0.035
Total(s)	6535.354	6370.715	6378.829
Standard Deviation	282.493	273.983	271.103
Relative(%)	100.0%	97.5%	97.6%

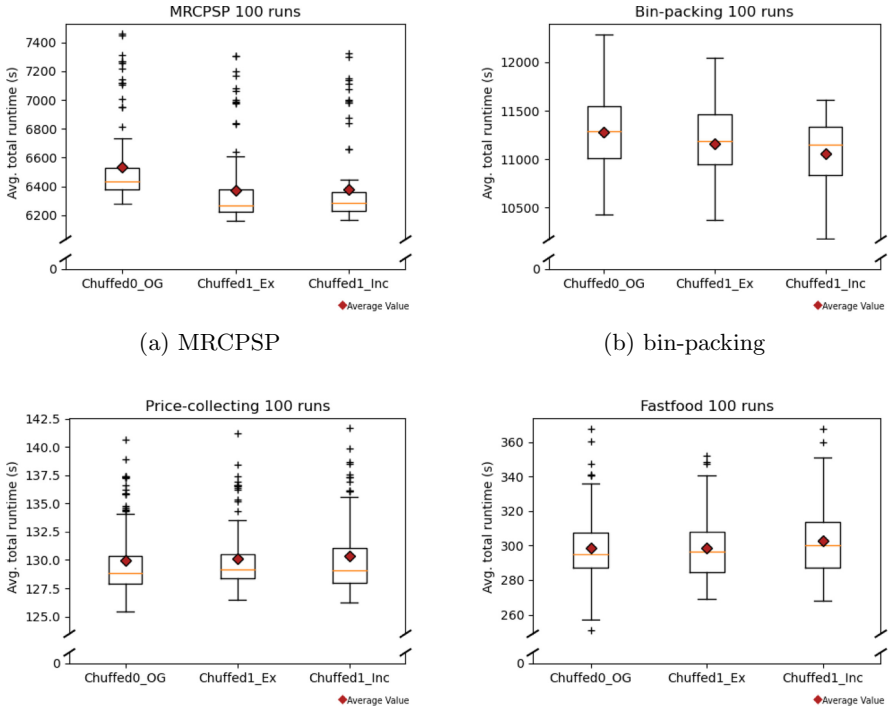
using MiniZinc's *findMUS* command [16]. Note findMUS often returns multiple different MUS combinations and a variable is deemed being part of a MUS if it is in *any* one of them. The datasets contained 13,667 problem instances for which features were available and 8,057 instances for which labels could be extracted. These latter instances contain 1,532,444 variables, of which 623,293 (40.7%) are part of at least one MUS. The dataset is dominated by a single problem type, namely the Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSP): over 90% of total instances. Additionally, over 80% of the instances in the dataset could be solved in less than 0.1 s. These non-challenging instances were excluded as being of limited use for training and testing.

4.2 Experimental Configuration and Results

In training the two learning Chuffed variants, the parameters of the GCN model were set as follows, based on initial trial runs: Learning rate: 0.3; Number of epochs: 200; Number of units in the first hidden layer: 16; Dropout rate: 0.1; Weight decay: $5e^{-4}$; Tolerance for early stopping: 10; Prediction accuracy at the point of early stopping was between 0.7 and 0.8.

The three Chuffed versions were used to solve test-sets containing instances respectively from the four largest problem types: MRCPSP, bin-packing, price-collecting and fastfood. The experiments were run on a Linux machine with a 16-core 2.50 GHz Xeon Gold 6248 CPU and 32 GB RAM. The code and datasets are available at doi.org/10.4121/14259635.

The box-plot in Fig. 1 shows the resulting distribution of the the total run-times of all instances from the each of the four largest problem types, averaged



(a) MRCPSP (b) bin-packing

(c) price-collecting (d) fastfood

Fig. 1. Box-plots of total runtime of all test instances averaged over 100 runs.

over a total of 100 runs. A more detailed summary of the results is presented for the two larger domains in Tables 1 and 2, which show the average runtime over 100 runs for each of the instances from the test-set as well as statistics on the total runtime. Table 3 reports the outcome of two-tailed t-tests.

The t-test analysis shows that the machine learning enhanced version significantly outperform the unmodified version for both MRCPSP and bin-packing instances. The gain is about 2.5% for MRCPSP and 1–2% for bin-packing. There is no sufficient statistical evidence to conclude any significant difference between the results obtained with Chuffed1_Inc and Chuffed1_Ex for MRCPSP. However, for bin-packing, there is a statistically significant difference between Chuffed1_Ex and Chuffed1_Inc, of about 1%. This may indicate that bin-packing shares less 'learn-able' concepts with other problem types than MRCPSP. For price-collecting and fastfood there is insufficient evidence to conclude statistically-significant differences between any of the different Chuffed versions. The most likely explanation is not about the dis-similarity of these instances to other problem types, but because the tested instances were not sufficiently large.

Table 2. Experiments on bin-packing benchmarks

Instances	Chuffed0_OG	Chuffed1_Ex	Chuffed1_Inc
	Avg. runtime (s)	Avg. runtime (s)	Avg. runtime (s)
2DLevelPacking238	171.700	151.000	152.580
2DLevelPacking23	1563.956	1499.611	1512.328
2DLevelPacking492	1221.866	1275.854	1237.965
2DPacking13	5065.462	5037.534	5025.021
2DPacking165	683.933	708.044	641.285
2DPacking168	2511.413	2430.075	2431.017
2DPacking62	58.744	57.180	57.587
Total(s)	11277.074	11159.298	11057.783
Standard Deviation	381.016	359.230	347.639
Relative(%)	100.0%	99.0%	98.1%

Table 3. Pairwise t-test analysis

MRCPSP	t-stat	p-value
Chuffed0_OG – Chuffed1_Ex	4.163	$4.693e^{-5}$
Chuffed0_OG – Chuffed1_Inc	3.978	$9.761e^{-5}$
Chuffed1_Ex – Chuffed1_Inc	-0.209	0.834

bin-packing	t-stat	p-value
Chuffed0_OG – Chuffed1_Ex	2.238	0.026
Chuffed0_OG – Chuffed1_Inc	4.230	$3.577e^{-5}$
Chuffed1_Ex – Chuffed1_Inc	-2.020	0.045

price-collecting	t-stat	p-value
Chuffed0_OG – Chuffed1_Ex	-0.226	0.821
Chuffed0_OG – Chuffed1_Inc	-1.506	0.134
Chuffed1_Ex – Chuffed1_Inc	-1.390	0.166

fastfood	t-stat	p-value
Chuffed0_OG – Chuffed1_Ex	-1.316	0.190
Chuffed0_OG – Chuffed1_Inc	-1.907	0.058
Chuffed1_Ex – Chuffed1_Inc	-0.648	0.518

5 Conclusion

This paper shows that it is possible to use machine learning approaches designed for solving SAT instances to improve lazy clause generation solving techniques. Specifically, we have shown how to use unsatisfiable core learning in its CP flavour as minimal unsatisfiable subsets, to improve the performance of the LCG solver Chuffed. We do this by learning the probability a variable is involved in a MUS, as a proxy for initial values of Chuffed's VSIDS scores. With CP–SAT approaches dominating recent MiniZinc benchmarks it is noteworthy that the proposed approach is able to consistently achieve an improved performance on sizeable instances. Although the relative margin of improvement is small (up to 2.5% on MRCPSP scheduling benchmarks), it is statistically significant in the largest two tested problem domains. This suggests that the *similarity* of a variable with variables from MUSs seen during training is a proxy for determining the conflicting nature of a variable.

Our work demonstrates the first, to our knowledge, successful application of machine learning to aid a CP–SAT optimisation solver. This paper thus opens the door to further research. For instance, integrating the classification part

directly into the solver can be investigated; this would require embedding the feature extraction part directly into the solver together with additional computational resources, e.g., a GPU as in the Neurocore approach. Moreover, one could consider alternative surrogates other than MUS membership to learn important variables for branching in CP–SAT solvers.

Acknowledgement. We thank the anonymous reviewers of CPAIOR. Thanks to S. van der Laan, K. Leo and P. J. Stuckey. This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under grant number 952215.

References

1. Amadini, R., Gabbrielli, M., Mauro, J.: SUNNY-CP: a sequential CP portfolio solver. In: Proceedings of the 30th ACM Symposium on Applied Computing, pp. 1861–1867 (2015)
2. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'Horizon. Eur. J. Oper. Res. **290**(2), 405–421 (2021)
3. Berg, J., Demirović, E., Stuckey, P.J.: Core-boosted linear search for incomplete MaxSAT. In: Rousseau, L.-M., Stergiou, K. (eds.) CPAIOR 2019. LNCS, vol. 11494, pp. 39–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19212-9_3
4. Chu, G., Stuckey, P.J., Schutt, A., Ehlers, T., Gange, G., Francis, K.: Chuffed, a lazy clause generation solver (2018). https://github.com/chuffed/chuffed
5. De Uña, D., Rümmele, N., Gange, G., Schachte, P., Stuckey, P.J.: Machine learning and constraint programming for relational-to-ontology schema mapping. In: Proceedings of IJCAI 2018, pp. 1277–1283 (2018)
6. Demirović, E., et al.: An investigation into prediction + optimisation for the Knapsack problem. In: Rousseau, L.-M., Stergiou, K. (eds.) CPAIOR 2019. LNCS, vol. 11494, pp. 241–257. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19212-9_16
7. Elffers, J., Gocht, S., McCreesh, C., et al.: Justifying all differences using pseudo-boolean reasoning. In: Proceedings of AAAI 2020, pp. 1486–1494 (2020)
8. Elmachtoub, A.N., Grigas, P.: Smart 'predict, then optimize'. CoRR abs/1710.08005 (2017). http://arxiv.org/abs/1710.08005
9. Galassi, A., Lombardi, M., Mello, P., Milano, M.: Model agnostic solution of CSPs via deep learning: a preliminary study. In: van Hoeve, W.-J. (ed.) CPAIOR 2018. LNCS, vol. 10848, pp. 254–262. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93031-2_18
10. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. In: Proceedings of NeurIPS 2019, pp. 15554–15566 (2019)
11. Gocht, S., McBride, R., McCreesh, C., Nordström, J., Prosser, P., Trimble, J.: Certifying solvers for clique and maximum common (connected) subgraph problems. In: Simonis, H. (ed.) CP 2020. LNCS, vol. 12333, pp. 338–357. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58475-7_20
12. Gocht, S., McCreesh, C., Nordström, J.: Subgraph isomorphism meets cutting planes: solving with certified solutions. In: Proceedings of IJCAI 2020, pp. 1134–1140 (2020)

13. Guerri, A., Milano, M.: Learning techniques for automatic algorithm portfolio selection. In: Proceedings of ECAI 2004, pp. 475–479 (2004)
14. Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., Dilkina, B.: Learning to branch in mixed integer programming. In: Proceedings of AAAI 2016, pp. 724–731 (2016)
15. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. CoRR abs/1609.02907 (2016). http://arxiv.org/abs/1609.02907
16. Leo, K., Tack, G.: Debugging unsatisfiable constraint models. In: Salvagnin, D., Lombardi, M. (eds.) CPAIOR 2017. LNCS, vol. 10335, pp. 77–93. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59776-8_7
17. Liang, J.H., Ganesh, V., Zulkoski, E., Zaman, A., Czarnecki, K.: Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In: Piterman, N. (ed.) HVC 2015. LNCS, vol. 9434, pp. 225–241. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26287-1_14
18. MiniZinc: The MiniZinc benchmark suite (2016). https://github.com/MiniZinc/minizinc-benchmarks
19. MiniZinc: Minizinc challenge 2020 (2020). https://www.minizinc.org/challenge2020/results2020.html
20. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of 38th Annual Design Automation Conference, pp. 530–535 (2001)
21. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
22. Selsam, D., Bjørner, N.: Guiding high-performance SAT solvers with unsat-core predictions. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 336–353. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_24
23. Selsam, D., Bjørner, N.: Neurocore: guiding high-performance SAT solvers with unsat-core predictions. CoRR abs/1903.04671 (2019). http://arxiv.org/abs/1903.04671
24. Song, W., Cao, Z., Zhang, J., Lim, A.: Learning variable ordering heuristics for solving constraint satisfaction problems. CoRR abs/1912.10762 (2019). http://arxiv.org/abs/1912.10762
25. Soos, M., Kulkarni, R., Meel, K.S.: CrystalBall: gazing in the black box of SAT solving. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 371–387. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_26
26. Yilmaz, K., Yorke-Smith, N.: A study of learning search approximation in mixed integer branch and bound: node selection in SCIP. AI **2**(2), 150–178 (2021). https://doi.org/10.3390/ai2020010