Towards Unsatisfiable Core Learning for Chuffed

van Driel, Ronald; Yorke-Smith, N.

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Towards Unsatisfiable Core Learning for Chuffed

Ronald van Driel[1] and Neil Yorke-Smith[1][0000−0002−1814−3515]⋆

Algorithmics group, Delft University of Technology, Netherlands
R.A.vanDriel@student.tudelft.nl, 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 a recent technique in the SAT literature called *unsatisfiable core learning* 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 cores 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

As part of the vision of the computer automatically solving the problem stated by the user – the 'Holy Grail' of computer science [2] – the computer must indeed solve the given problem. We suppose that the problem at hand is a combinatorial optimisation problems, and that it has been translated into a constraint programming (CP) model. In this paper we progress towards efficient self-adaptive solving by the computer.

Neuro-symbolic approaches to combinatorial optimisation problems include improving optimisation solver performance or robustness by incorporating machine learning. This trend shows successful promise in mixed integer programming [4], propositional satisfiability (SAT) [11] as well as CP [3].

Hybrid CP–SAT solvers are the state of the art for solving constraint programming problems [7]. For instance, Lazy Clause Generation (LCG) combines the conflict learning ability from SAT solvers with finite domain propagation from CP solvers [10].

⋆ Contact author

2   R. van Driel and N. Yorke-Smith

However the literature has not paid attention to using machine learning (ML) to improve hybrid CP–SAT methods. This paper provides a first demonstration of the value of using ML within the LCG solver Chuffed [1]. We develop a modified version of the SAT technique of unsatisfiable core learning [12] and employ it to learn initialisation values for Chuffed's Variable-State Independent Decaying Sum (VSIDS) variable selection heuristic. We benchmark the ML-aided Chuffed on problems from the MiniZinc benchmark suite and find a robust 2.5% average gain over the baseline Chuffed on MRCPSP instances.

The remainder of the paper is structured as follows. Section 2 describes our approach at a high level. Section 3 describes the implementation at a lower level. Section 4 reports the empirical evaluation. Section 5 briefly gives the context in the literature. Section 6 concludes.

2   Approach

This section provides a high-level design of the proposed approach for improving Chuffed with machine learning.

Similar to SAT solvers, Chuffed is able to use a Variable-State Independent Decaying Sum (VSIDS) heuristic. VSIDS is "a family of branching heuristics widely used in modern . . . SAT solvers that rank all variables of a Boolean formula during the run of a solver" [6].

VSIDS is usually implemented by keeping track of an activity score for each variable which indicates the value of branching on that variable. Normally the score for each variable is initialised at zero and incremented when the corresponding variable occurs as part of a conflict. To emphasise variables visited recently the activity scores are periodically decreased.

Consequently, initially the scores do not provide any information to the solver but they gradually become more useful. Chuffed typically uses (user-specified) search annotations before switching to VSIDS for making branching decisions. However, with machine learning the activity scores can be directly initialised, which may benefit the solver also in early stages of the solving procedure.

To achieve this VSIDS activity score initialisation, a Graph Convolutional Network (GCN) model is trained on unsatisfiable instances to make a prediction on which variables belong to an unsatisfiable core. An unsatisfiable core is a minimal subset of variables which can not not be simultaneously satisfied. The trained model is then used to classify the variables of an instances which needs to be solved and the softmax probabilities of this classification are used to initialise Chuffed's VSIDS scores.

2.1   Data

The approach just described requires two different datasets containing CP instances. One of these datasets should only contain unsatisfiable instances to train on; the other should contain satsifiable instances to solve for evaluation.

The MiniZinc benchmark suite [8] was used to supply over 13,000 satsifiable instances for evaluation. Since we found no public CP dataset contained sufficiently many unsatisfiable instances for training any machine learning model on, the constraint optimisation problem (COP) instances from the MiniZinc benchmark suite were also modified to become unsatisfiable. This was done by first solving them for their optimal value. Then the original instance was modified by setting the domain of the objective variable to only include values better than the optimal value, which makes the instances unsatisfiable. While less computationally intensive alternatives exist, this procedure was selected with the intention to not introduce any unwanted bias for learning the unsatisfiable cores.

Using this procedure allows the creation both the satisfiable datset as well as the unsatisfiable dataset. For the unsatisfiable dataset the labels were generated using MiniZinc's 'findMUS' command. Ultimately the datasets contained 13,667 instances for which features were available and 8,057 instances for which labels could be extracted. The dataset is skewed in that over 90% of the data belonged to a single problem type, namely the Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSP). This is because over 80% of the instances in the dataset could be solved in less than 0.1 second. These non-challenging instances were not useful for training

3   Implementation

This section describes the techniques used to integrate machine learning with Chuffed. Our implementation is in Chuffed version 0.10.4.[1]

It is important to note that, similar to the approach proposed by Selsam and Bjørner [12], our intention is not to achieve the best possible predictions. The reason for this is that more accurate predictions do not necessarily imply that they are more useful for the solver. In fact, if all variables of a satisfiable instance would be correctly classified as not being part of an unsatisfiable core with 100% certainty, it would not provide any information to the solver at all. Instead, the assumption is that the confidence of classifying a variable to be part of an unsatisfiable core correlates with the effectiveness of branching on that variable.

3.1   Features and ML model

We follow the Graph Convolution Network model of Kipf and Wellig [5].[2] A GCN works by learning a function of the features on a graph. In this case no actual graph was constructed but it is sufficient that the data is structured in such way that it could be represented with a graph. A simplified overview of the architecture is shown in Figure 1.

We adopt the following feature representation for the GCN model:

---

[1] Code available at `https://github.com/chuffed/chuffed`.
[2] Code available at `https://github.com/tkipf/gcn`.
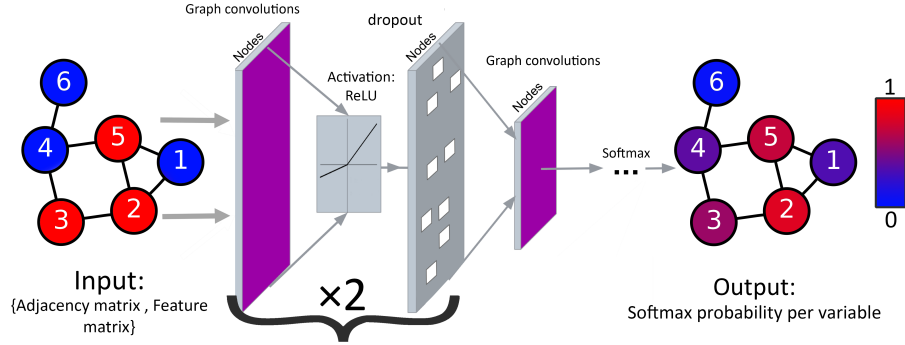
4   R. van Driel and N. Yorke-Smith

Fig. 1: Visualisation of the Graph Convolutional Network architecture

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

The input of the GCN model is threefold:

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 an unsatisfiable core and the other for variables which are not.

The output of the model is a $N \times C$ matrix which contains the softmax output 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 machine learning predictions with a single value, which is the prediction confidence of a variable belonging to an unsatisfiable core.

The following parameters of the 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. As found in Section 4, this accuracy was sufficient to show improvement in solving runtime when the ML model is used in Chuffed.

Instances	**Chuffed0_OG** Avg. runtime(s)	**Chuffed1_Ex** Avg. runtime(s)	**Chuffed1_Inc** 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%

Table 1: MRCPSP

4   Evaluation

This section reports an experiment to evaluate the effectiveness of the proposed approach. For this experiment three different versions of Chuffed were compiled from source: *Chuffed0_OG*, *Chuffed1_Ex* and *Chuffed1_Inc*. All three of them were configured to switch to VSIDS as soon as 100 conflicts have been encountered.[3] While all three versions have an identical configuration, they are different in the way the machine learning was integrated. Chuffed0_OG was otherwise left completely unmodified, and serves purpose 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.

These three different version were used to solve different selected test-sets containing instances 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 Xeon Gold 6248 CPU @ 2.50 GHz and 32 GB RAM.

The box-plot in Figure 2 shows the resulting distribution of the the total runtimes of all instances from the each of the four largest problem types, averaged over a total of 100 runs. A more detailed summary of the results is presented in Tables 1–4, 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 5 reports the outcome of two-tailed t-tests.

---

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

Instances	Chuffed0_OG Avg. runtime(s)	Chuffed1_Ex Avg. runtime(s)	Chuffed1_Inc 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 2: bin-packing

Instances	Chuffed0_OG Avg. runtime(s)	Chuffed1_Ex Avg. runtime(s)	Chuffed1_Inc Avg. runtime(s)
pc52	12.211	12.326	12.152
pc56	8.777	8.750	8.750
pc58	15.283	15.409	15.431
pc61	10.501	10.648	10.644
pc65	12.111	11.908	12.049
pc73	42.743	42.407	42.948
pc77	7.886	8.013	8.040
pc79	20.479	20.631	20.709
Total(s)	129.991	130.092	130.722
Standard Deviation	3.373	2.895	3.452
Relative(%)	100.0%	100.1%	100.6%

Table 3: price-collecting

Instances	Chuffed0_OG Avg. runtime(s)	Chuffed1_Ex Avg. runtime(s)	Chuffed1_Inc Avg. runtime(s)
fastfood15	30.568	31.614	31.580
fastfood17	23.212	25.660	23.382
fastfood20	8.492	6.361	6.867
fastfood36	6.414	6.464	6.396
fastfood53	80.526	86.375	86.946
fastfood58	49.063	40.309	45.349
fastfood61	18.369	20.553	20.467
fastfood74	81.844	84.775	82.858
Total(s)	298.487	302.111	303.844
Standard Deviation	20.318	18.402	19.197
Relative(%)	100.0%	101.2%	101.8%

Table 4: fastfood

(a) MRCPSP

(b) bin-packing
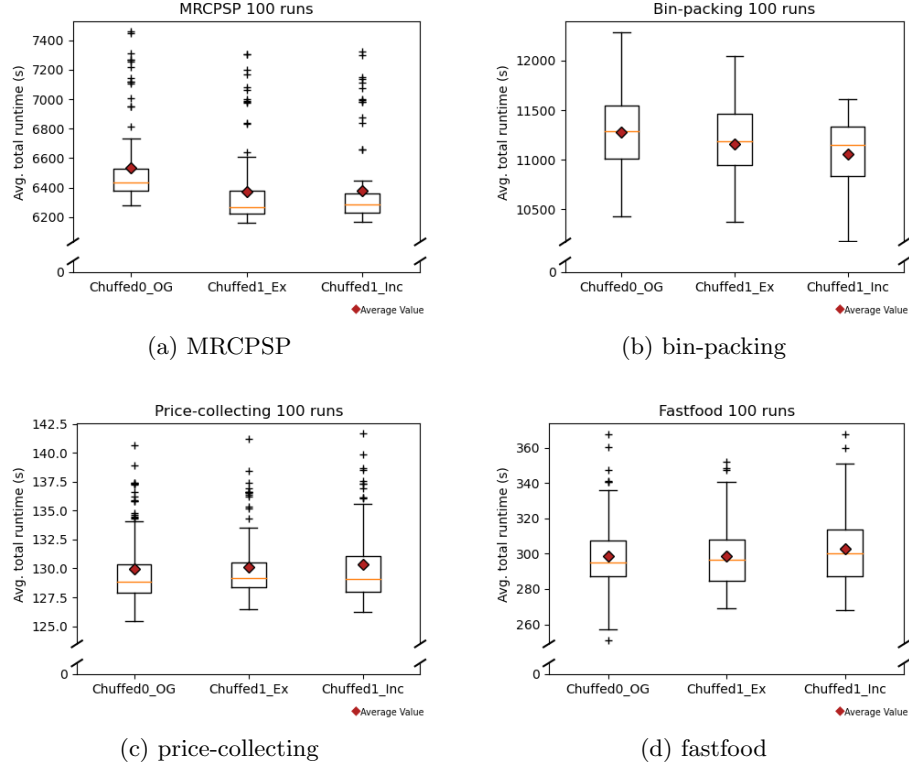

(c) price-collecting

(d) fastfood

Fig. 2: Box-plots showing the total runtime of all test instances averaged over 100 runs for the four largest problem types.

The t-test analysis shows that the machine learning enhanced version significantly outperform the unmodified version for both MRCPSP and bin-packing instances. The probability for obtaining less similar results on the MRCPSP test-set compared to Chuffed0_OG is less than 0.01% for Chuffed1_Inc and less than 0.005% for Chuffed1_Ex. For bin-packing these probabilities are 2.6% and 0.0036% respectively. This means the hypothesis that they follow the same distribution as Chuffed0_OG can be rejected with over 99.99% certainty for MRCPSP and 97% certainty for bin-packing. Therefore, it makes sense to conclude that the ML-enhanced versions both outperform the unmodified version on MRCPSP and bin-packing instances.

There is, however, no sufficient statistical evidence to conclude any significant difference between the results obtained with Chuffed1_Inc and Chuffed1_Ex for MRCPSP. The probability of obtaining less similar results with identical distributions is over 83%. However, for bin-packing, in addition to the difference between the machine learning enhanced version compared to Chuffed0_OG, it is also 95% certain that there is a statistically significant difference between

8   R. van Driel and N. Yorke-Smith

MRCPSP			bin-packing		
Version Pair	t-stat	p-value	Version Pair	t-stat	p-value
Chuffed0_OG - Chuffed1_Ex	4.163	$4.693e^{-5}$	Chuffed0_OG - Chuffed1_Ex	2.238	0.026
Chuffed0_OG - Chuffed1_Inc	3.978	$9.761e^{-5}$	Chuffed0_OG - Chuffed1_Inc	4.230	$3.577e^{-5}$
Chuffed1_Ex - Chuffed1_Inc	-0.209	0.834	Chuffed1_Ex - Chuffed1_Inc	-2.020	0.045

price-collecting			fastfood		
Version Pair	t-stat	p-value	Version Pair	t-stat	p-value
Chuffed0_OG - Chuffed1_Ex	-0.226	0.821	Chuffed0_OG - Chuffed1_Ex	-1.316	0.190
Chuffed0_OG - Chuffed1_Inc	-1.506	0.134	Chuffed0_OG - Chuffed1_Inc	-1.907	0.058
Chuffed1_Ex - Chuffed1_Inc	-1.390	0.166	Chuffed1_Ex - Chuffed1_Inc	-0.648	0.518

Table 5: Pairwise t-test analysis

Chuffed1_Ex and Chuffed1_Inc. 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 any significant differences between results of the different Chuffed versions. The most likely explanation is that, because of the limited data available for these problem types, none of the price-collecting or fastfood instances required considerable solving time. The average runtime per instance stated in the tables indicate that the machine learning integration works better for sizeable instances. Therefore it is most likely that lack of improvement on price-collecting and fastfood is not because they are less similar to other problem types but because the tested instances were not sufficiently large.

5   Related Work

Stuckey [10] proposed a hybrid CP–SAT solver based on Lazy-Clause-Generation (LCG). LCG combines finite domain propagation with the conflict learning ability of SAT. Because of this LCG solvers are able to use conflict driven heuristics such as VSIDS, which were originally developed for SAT solver Chaff [9,6].

Multiple approaches have been proposed to combine machine learning with traditional SAT or CP solvers. For example, Song et al. [13] show that machine learning can be used to automatically learn variable ordering heuristics for CSP solving. However to the best of our knowledge there have not been any research to date to combining machine learning with hybrid CP–SAT solvers. This paper draws inspiration from the work by Selsam and Bjørner [12]. Their work describes how unsatisfiable core learning can be used to initialise the values of the VSIDS for a selection of well-known SAT solvers. With this approach they manage to solve between 6 and 20% more instances within the same amount of time compared to the original solver.

6   Conclusion

This paper shows that it is possible to use machine learning approaches which are designed for solving SAT instances to improve LCG solving techniques. This results in a step towards part of the 'Holy Grail' goal: the computer automatically solving a given combinatorial optimisation problem.

Specifically, we have shown that it is possible to use unsatisfiable core learning, which originates from the work of Selsam and Bjørner [12], for improving the performance of the LCG solver Chuffed. With LCG-based approaches dominating recent benchmarks it is interesting that the proposed approach is able to consistently achieve an improved performance on sizeable instances. Although the margin of improvement is small, it is statistically significant.

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 should be investigated; this would require embedding the feature extraction part directly into the solver. Second, in order to examine the effect across different problem types this experiment it may be valuable to repeat this study with more evenly distributed datasets. Last, ML may also be useful for predicting no-goods or their activity scores.

Acknowledgements

References

1. 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
2. Freuder, E.: In pursuit of the Holy Grail. Constraints **2**, 57–61 (1997)
3. Galassi, A., Lombardi, M., Mello, P., Milano, M.: Model agnostic solution of CSPs via deep learning: A preliminary study. In: Proc. of CPAIOR'18. Lecture Notes in Computer Science, vol. 10848, pp. 254–262. Springer (2018)
4. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. In: Proc. of NeurIPS'19. pp. 15554–15566 (2019)
5. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. CoRR **abs/1609.02907** (2016), http://arxiv.org/abs/1609.02907
6. Liang, J.H., Ganesh, V., Zulkoski, E., Zaman, A., Czarnecki, K.: Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers. In: Proc. of HVC'15. Lecture Notes in Computer Science, vol. 9434, pp. 225–241. Springer (2015)
7. The MiniZinc Challenge 2019 results, https://www.minizinc.org/challenge2019/results2019.html
8. The MiniZinc benchmark suite (2016), https://github.com/MiniZinc/minizinc-benchmarks

9. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. of 38th Annual Design Automation Conference. pp. 530–535 (2001)
10. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
11. Selsam, D., Bjørner, N.: Guiding high-performance SAT solvers with unsat-core predictions. In: Proc. of SAT'19. Lecture Notes in Computer Science, vol. 11628, pp. 336–353. Springer (2019)
12. 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`
13. 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`