Revealing the limits of Machine Learning for Satisfiability Testing: a dataset for learning minimal moves

Laurent Simon
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France

Abstract

Despite the progress observed in machine learning, its successful application to SAT is still very modest. One of the reasons for this is probably the lack of clear challenges and datasets for the ML community to focus on. This would stimulate the community to propose, for example, new neural network architectures that are well suited for SAT search. We propose a methodology and a dataset for training machine learning systems to solve satisfiable formulas. Our dataset consists of random formulas of relatively small size (250 variables) at the threshold, all of which are satisfiable. The task to be predicted is to identify, given an overall assignment, the set of variables that must be flipped to make the formula satisfiable. Our dataset consists of hundreds of thousands of examples (each on a different random formula) with the minimum number of flips that need to be performed, given a total assignment.

One of our result is to show that, despite impressive predicting qualities of a Graph Neural Network predictive model on them, a pure machine learning method based solely on the prediction failed on 100% of the benchmarks while they are trivial for local search SAT solvers from the late 90's.

We think that this unveils a key missing piece of current ML approach that could be identified by focusing on our proposed dataset.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Satisfiability, Experimental Study, Machine Learning

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Category Tool Paper

Funding *Laurent Simon*: Partially Funded by the ANR-?? and the Chaire Fondation Bordeaux Université

1 Introduction

As the number of successful applications of machine learning continues to grow, it is increasingly common to see new work applying it to combinatorial search algorithms like SAT. However, while this direction is appealing, most of the proposed approaches attempt to beat current SAT solvers more or less directly, resulting in hybrid approaches where the impact of ML is difficult to measure: genuine ML approaches are not yet ready to beat any SAT solver that was proposed less than 30 years ago.

In this paper, we propose to use a very simple set of benchmarks for the ML community to work on. Most of the progress in the ML community in recent years has been based on clean and exhaustive datasets, which allowed to design efficient ML architectures for each targeted application (image, speech, text, ...). We lack such datasets in SAT. Moreover, we hardly understand the real capabilities of ML-based approaches for SAT. We therefore propose to take a step back and take time to understand the power of neural networks for SAT problems before trying to hybridise them with already complex solvers. SAT solvers are very sensitive to noise and side effects, so with a good choice of problems or solvers it may be easy to observe what could be considered progress (but would be difficult to generalise).

In a broader context, we expect that this dataset will allow the development of efficient neural network architectures designed for SAT. If our proposal to use ML is limited to a simple solver

design (as we will see in the latter), we expect other approaches (such as a reinforcement learning approach directly in a CDCL solver) to emerge afterwards, once the right architecture for learning the dependencies of variables has been designed and validated. This is exactly what our dataset should provide. Our dataset consists of a set of hundreds of thousands of very simple uniform random formulae, with the minimum number of flips to be performed to make the formula true. Building this set of benchmarks was not trivial, and a special version of the SAT solver Glucose was developed to prove the optimality of the set of flips.

To illustrate the usefulness of the dataset, we show how a classical graph neural network could be trained on it and how, besides impressive performances in predictions, it is still very difficult for it to beat even very simple local search algorithms: very small errors in prediction prevent this approach from converging by itself. For this reason, we believe that the problem of proposing a new neural network architecture for SAT needs to be studied first. We think that there is a missing piece in the design of ML-based SAT solvers and providing a clear and estabilised dataset could help finding this piece.

Our dataset is aimed to give a clear and separate performance measure on the usefulness of ML for SAT problems before considering to hybrid them with SAT solvers.

After a short introduction of the different works that have been proposed so far to use ML techniques for SAT, we present the dataset itself and how (and why) it was generated. In the last section of this paper, we show how a true ML approach could use this dataset to tackle the SAT problem, pointing out a clear limitations of a classical Graph Neural Network encoding of SAT problems.

2 Approaches of Machine Learning for SAT

We suppose the reader familiar with SAT, Conflict-Driven Conflict Analysis (CDCL) [10, 4, 2] and Stochastic Local Search (SLS) [5] algorithms. The use of machine learning for SAT solving is not new. The first approaches were introduced in the context of solver selection for portfolios [6] (how to choose the right solver for the right benchmark), or in the design of SAT solvers (by fixing a set of constants).

In the context of pure machine learning based approaches, the picture is not so successful. We can cite Neurosat [9], which attempts to predict the satisfiability of CNF formulas, and NeuroCore [8] and NeuroCom [13], which learns to predict a set of unsat cores to guide the heuristics. In [14], the classic Walksat selection for flipping a variable is replaced by a GNN-based predictor. The last part of our work will rely heavily on this work to test the capabilities of ML for SAT. In [7], they propose a GNN-based SAT solver that tries to force the VSIDS heuristic to favour certain variables. In [12], they used a RL algorithm based on a game representation of SAT. However, all of these approaches lack the crucial (an somehow simple) information that we want to study: which neural network architecture is best suited to handle SAT formulas? What are the intrinsic limitations of message-passing neural networks models of SAT problems?

3 A new local search algorithm as a testbed

Despite mixed results so far, the use of ML for SAT is still attractive: The VSIDS heuristic is not yet fully understood, and the structure behind real-world examples is also difficult to define in practice, except through examples. The use of an ML approach would certainly be of interest. However, ML and SAT are not based on the same paradigms: (1) CDCL solvers and local search solvers are based on very fast decision heuristics that are orthogonal to ML approaches, especially neural network architectures, where each prediction may require millions of computations. (2) SAT problem sizes

```
// maxtries limits the hamming distance from the reference model
nbtries ← 0
found ← False
while not found and nbtries < maxtries do
    // Starts with a fresh assignment
    assignment ← randomAssignment()
    found ← isSAT(assignment)
    nbmoves ← 0
    while not found and nbmoves < maxmoves do
        // moves is a set of flips to perform, predicted by ML
        moves ← predict(assignment)
        assignment ← flipMoves(assignment, moves)
        nbmoves ← nbmoves + 1
        found ← isSAT(assignment)
    end
    nbtries ← nbtries + 1
end
```

Algorithm 1 Slow Local Search Algorithm. The only difference with classical Local Search algorithms is the fact that a set of flips is performed at each inner loop. The algorithm ends with either a solution, or the limit of mxnbtries reached

can range from a few hundred to billions of variables, making the encoding of inputs very unusual for ML approaches. In addition, (3) VSIDS is strongly related to the clause learning mechanism. CDCL solvers add tens of thousands of new clauses every second and regularly remove hundreds of thousands of them, resulting in a constantly changing internal representation of the formula to be handled, which is another source of hardness for ML approaches.

In the proposed dataset, we focus on a simple, clear, but challenging problem that is orthogonal to what has been done in SAT so far. First, we have restricted ourselves to only random SATisfiable formulas, on which it is possible to compute the truth for predicting labels. We propose to investigate a new paradigm for local search in the hope of answering one of the last open challenges proposed 25 years ago in [1]: "*Improve stochastic local search on structured problems by efficiently handling variable dependencies*".

Our setup challenges the ML community to replace the fast and greedy flipping mechanism of classical local search solvers with a slow but robust ML-guided flipping mechanism. Instead of performing one flip at a time, our proposal is to start from a global, random, initial assignment and let the ML prediction to recommend any number of flips to be performed simultaneously. The challenge is to converge to a global allocation, as described in Algorithm 1.

4 Building the Dataset thanks to a specialised Solver

In order to allow ML techniques to learn the correct flips to perform, we need to build a set of problems on which the exact flips are labelled. However, it is not possible to simply start from a solution and randomly flip variables (like adding noise to the global solution to train diffusion models [11]): depending on the structure of the problem, we may move from one cluster of solutions to another. If we want to correctly label the right flips to perform, we need to prove that there are no other solutions with a lower hamming distance to the initial assignment. To do this, we modified Glucose to constantly measure the Hamming distance of the current partial assignment (the trail) from a given global reference model during propagation. When the current boundary is reached, a

```
// maxbound limits the hamming distance from the reference model
bound ← maxbound
while result ≠ Unsat do
    // The following call will return after each restart
    result ← search(referenceModel, bound)
    if result = SAT then
        // found a solution
        bestBound = HammingDistance(foundModel, referenceModel)
        bound ← bestBound − 1
    end
end
```

Algorithm 2 Modified Search main loop of a typical CDCL solver. $isProbing$ is true if the current process is not the main process.

clause is learned. This clause will contain the negation of all decision variables plus a special literal (an assumption literal, meaning that the limit has been reached).

As described in Algorithm 2, the outer loop of Glucose is modified so that the bound is lowered after each solution. By using the assumption literal, we ensure that all learned clauses can be kept after each iteration (a clause containing the "reached limit" assumption literal can be kept if the limit is only lowered: its properties will hold until the end of the run). We can then rely on Glucose's classical clause database reduction, with no special rules, to handle the set of clauses containing the assumption literals. When the algorithm finds UNSAT (with respect to the assumption literal), we know that the last foundModel was the closest to the reference model. We can then label all literals that do not match the reference model as the minimum flips to perform.

4.1 Limiting the search around the reference model

We had to add a few other critical features to our solver. Firstly, it can start with a random order of variables and a random phase, allowing it to find a first solution in a completely random direction. Second, the solver works in two phases, which is not explicitly mentioned above. In the first phase, starting from a random order of variables and phases, it tries to find a SAT solution, and if a solution is found, it is used as the reference model. Then we pick a random number of variables and switch them. This number is to be used as the initial bound in the algorithm 2. The number of literals to randomly fork is a parameter of the algorithm. We did this because, in our early experiments and on large (industrial) formulas, it was practically impossible for the solver to simply search with an initial bound initialised to the number of variables. Adding the bound makes it possible to work on any type of SAT formula (random, industrial, cryptographic, ...).

4.2 Focusing on Random Problems

We first tried to generate examples based on industrial problems from recent competitions. However, there were a number of questions that needed to be answered before we could build such a dataset. Some examples had only one solution. Some had many many. How to represent these different situations in the data set? How to balance the different types of problems in the different competitions? Also, how to take into account problems that do not scale in difficulty?

We decided to focus on a set of problems that are very well known and for which message-passing algorithms (Survey Propagation for example [3]) have already been studied. This is a challenging

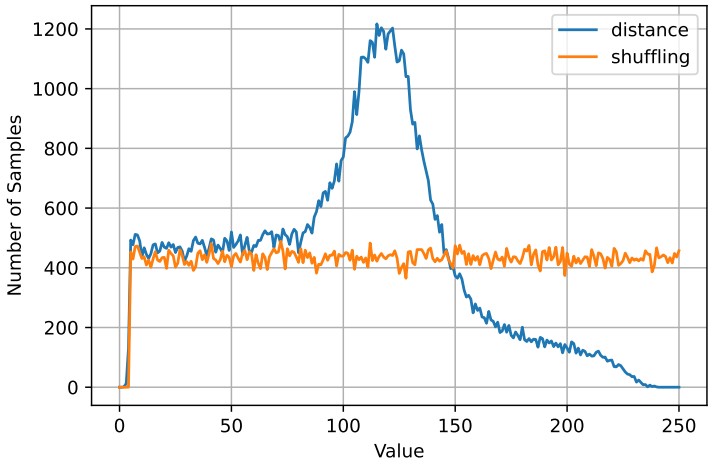

Figure 1 Distribution of data. We see that a uniform random selection of shuffling leads to a non uniform distribution of distance, which is typical for uniform random CNF instances. This observation is related to the existence of backbone variables that can take any values in all the models.

set of problems because at the threshold, variables can have a critical long-range influence on the solution.

In addition, by considering only one unique point per (formula, solution, noise), there is no over-representation of a particular problem. However, we plan to enrich the dataset with industrial problems when we have a way to address the above problems.

4.3 Characteristics of the dataset

The dataset is released with a Python code to completly re-generate a CNF formula given its random seed, its number of variables and clauses. Each entry of our dataset is thus simply this three characteristics numbers and a compact way of representing the total assignements and the flips to perform. We provide the Python code to decode this compact representation into two strings of "1" and "0". One string encodes the signs of the literals for the reference model, and the other string encodes the signs ot the literals for the solution to the random formula. The hamming distance between the two complete assignements is another entry of the dataset (for easy selection of points).

We generated two datasets, and an additional test set. The first dataset have entries with hamming distances ranging from 5 to 250. This dataset has 106675 points. For each point, we first randomly generated a SAT formula, found a solution (with randomness as describe above) and then randomly pick a number of variables to flip (ranging uniformly from 5 to 250), then computed the solution with the minimal hamming distance from this reference.

However, as reported in Figures 1 and 2, the uniform distribution in the shuffling parameter did not lead to a uniform distribution in the minimum distance (as expected for random problems). There is an overrepresentation of the distance by about half the number of variables.

When we designed our dataset, we did some preliminary experimental studies on the ability of our ML model to generalise. This showed that it was more efficient to learn only on examples with a small distance. Therefore, we generated an even larger set of problems with the same conditions, but with a shuffle value of only 5 to 25. For this second set of problems, we collected 295053 samples.

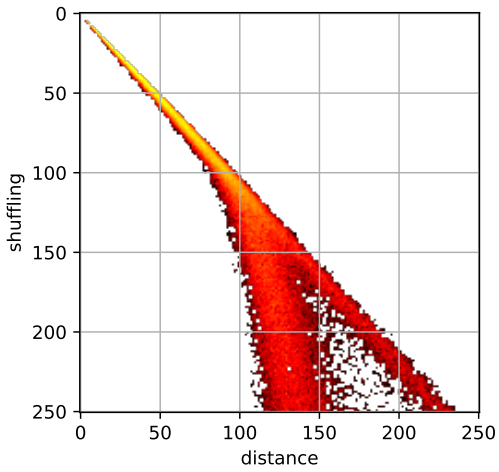

Figure 2 . A heatmap view of the distribution of values (distance w.r.t. shuffling value).

At last, we generated a set of random seeds that just produced SAT instances, to ease the tests of learnt architectures. We collected a set of 20005 of such random seeds.

5 Pure ML techniques are failing on our trivial SAT formulas

To ensure that our dataset is useful and already challenging for the ML community, we wanted to illustrate how such a simple task is very hard despite its simplicity. These benchmarks are trivial for local search for SAT / Modern SAT solvers.

For this, we show in this section that a full ML approach, with high precision and recall fails to generalize for any new problem.

Of course, hybrid approaches are possible and used in practice, but we advocate here that gains of hybrid approaches may be related to other parameters of the search and side effects, like diversifications or restarts.

Thus, we propose to encourage the ML community to focus on a pure ML approach to transform the precision and recall of the decisions into actual solving.

We used the code released in [14] to learn a Graph Neural Network (GNN) model to predict which literals to flip.

In the ML model, a CNF instances is represented as a bipartite variable–clause graph and iteratively updates variable and clause embeddings via learned MLP message functions and sparse matrix multiplications. We used the `ReinforcePolicy` suggested in the source repository to learn the flipping flags. We tested this simple GNN architecture with 10 iterations, an MLP size of 32 and simple defaults proposed in [14] (GNN hidden size of 32, readout hidden size of 64 and ReLU activation function, with cross entropy loss, Adam optimiser (with a learning rate of 0.0005)). Training was performed on a GeForce RTX 3060 with 12 GB of VRAM. We used batches of 400 examples with 300 epochs and 33% of the points for validation. The batches were randomly rebuilt after each epoch. It is important to note that, unlike [14], our test is purely ML-based (in [14], they randomly select a variable to flip in an unsatisfied clause half the time).

What is striking, as shown Figure 3, is that the network quickly learns to perfom only positive moves (the number of correct flips - the number of incorrect flips) with very high quality. This is

showed by the strong results of the F1 score of the predicions, as seen (and commented) . We used the F1-score because the prediction are unbalanced (average of 12 flips to predict over 250 variables). The progress measure (see caption) also suggests that iteratively predicting the flips will converge.

With such positive results, it seems easy to solve any SAT instance. However, our negative results comes. For each of the 20005 examples of our third data set, we implemented a very single greedy descent, starting with a single random assignment, as long as the number of unsatisfied clauses strictly decreases after each move. The idea of this test was just to check its performance and see how well it performs in reducing the number of UNSAT clauses. We thus didn't want to rely on noise introduction or other non-ML techniques to solve the instance.

The results are reported in Figure 4. They are particularly disappointing. They show that, despite the very positive results so far, pure ML techniques need to be improved, and this is exactly what this dataset is about. None of the 20005 examples were solved by a ML-based greedy descent. One of the possible reasons for this is that, when measuring progress in the above experiment, the most important variables to flip (with far-reaching effects) were not identified. Of course, a more sophisticated local search would probably solve all examples (e.g. by introducing a taboo list for flips, a novelty heuristics, a special noise, by parallelizing the search with as many initial assignments as possible, ...). But this is not the purpose of the current work. One question arises: if GNN based approaches fails to solve trivial benchmarks, are the current results in ML strong enough if we replace the prediction with local search algorithms? This is a research to be conducted.

6 Conclusion

This paper proposes to provide a dataset of about 300,000 examples to challenge the ML community. These examples are trivial for current SAT solvers, but surprisingly hard for pure ML techniques. We hope that this dataset will stimulate the development of new neural network architectures that will be beneficial for SAT.

We plan to release other sets of benchmarks, for example based on SAT problems from the SAT competition. To do this, we are thinking of simplifying a hard problem by a partial mapping, and then using our technique to produce examples. However, the fact that a single seed can define an entire formula is also an attractive feature of our approach, allowing a very compact representation of the problem set.

References

1. David Mc Allester Bart Selman, Henry Kautz. Ten challenges in propositional reasoning and search. In *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1997.
2. Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
3. Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. Survey propagation: an algorithm for satisfiability. *CoRR*, cs.CC/0212002, 2002. URL: http://arxiv.org/abs/cs/0212002.
4. N. Eén and N. Sörensson. An Extensible SAT-solver. In *Proc. SAT'03*, pages 333–336, 2003.
5. Bart Selman Henry Kautz, Ashish Sabharwal. *Handbook of Satisfiability*, chapter Incomplete Algorithms. 2008.
6. Kevin Leyton-Brown Holger H. Hoos, Frank Hutter. *Handbook of Satisfiability (second edition)*, chapter Automated Configuration and Selection of SAT Solvers. Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh, 2021.
7. Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver? In *Advances in Neural Information Processing Systems 32*, 2020.

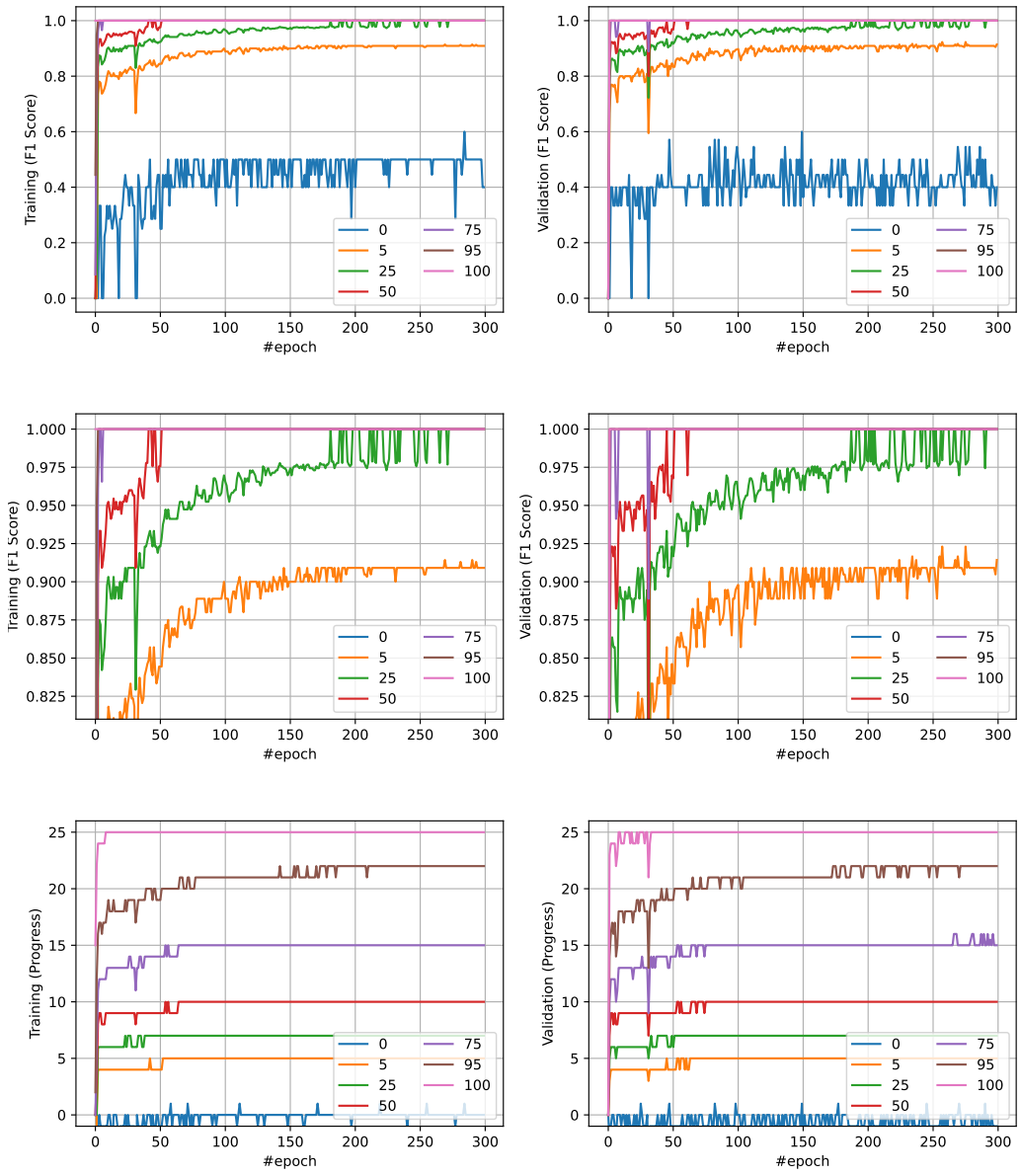

Figure 3 . Characteristics of the learning performances on the set of problems with small distances (<26). Each curve shows the evolution of a given quantile (given in the legend) over the values on the whole set considered (training, validation). The second row is a focus on the high values for the F1-scores. The 0 curve is the minimum of all values. The first four figures show that the F1-scores values are very high (above 0.9 for 95% of the predictions and above 0.975 for 75% of them) and that the model generalises well on the validation examples. The last two figures show the distribution of the "progress" obtained after each epoch (number of correct flips - number of incorrect flips). The fact that this value is positive in almost all cases should be a strong indicator that our ML models always know where to go.

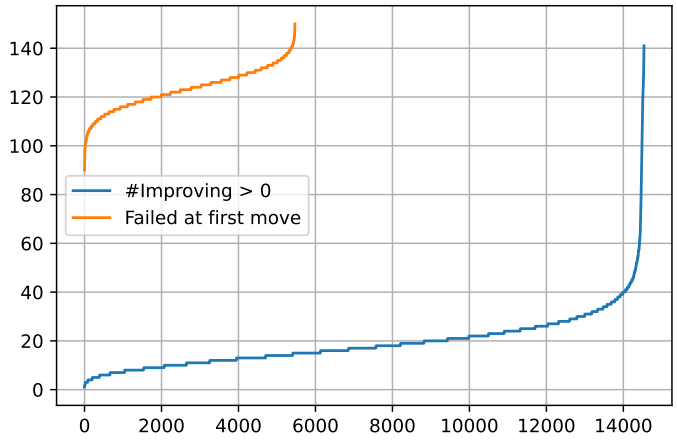

Figure 4 Final number of unsatisfiable clauses obtained on the dataset of 20005 examples, starting with only one random assignment and with only one descent. We divided the results into two parts. On almost 6000 examples, the ML model failed to improve even the first step ! The upper right curve is thus the distribution of the number of UNSAT clauses with random assignments where the greedy descent stopped immediatly. The lower curve shows results where at least one step was benefic. For instance, we see that, for at least 8000 examples, the ML model was able to optimise the number of UNSAT clauses below 20. It is not visible here, but no example ended with 0 UNSAT clauses.

8 Daniel Selsam and Nikolaj S. Bjørner. Neurocore: Guiding high-performance SAT solvers with unsat-core predictions. *CoRR*, abs/1903.04671, 2019. URL: http://arxiv.org/abs/1903.04671, arXiv:1903.04671.

9 Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. *CoRR*, abs/1802.03685, 2018. URL: http://arxiv.org/abs/1802.03685, arXiv:1802.03685.

10 J. P. Marques Silva and K. Sakallah. Grasp – a new search algorithm for satisfiability. In *Proc. CAD'96*, pages 220–227, 1996.

11 Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. arXiv:1503.03585.

12 Fei Wang and Tiark Rompf. From gameplay to symbolic reasoning: Learning SAT solver heuristics in the style of alpha(go) zero. *CoRR*, abs/1802.05340, 2018. URL: http://arxiv.org/abs/1802.05340, arXiv:1802.05340.

13 Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth McMillan, and Risto Miikkulainen. Neurocomb: Improving sat solving with graph neural networks. *arXiv:2110.14053*, 2021. URL: http://www.cs.utexas.edu/users/ai-lab?wang:arxiv21.

14 Emre Yolcu and Barnabas Poczos. Learning local search heuristics for boolean satisfiability. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/12e59a33dea1bf0630f46edfe13d6ea2-Paper.pdf.