Learning cubing heuristics for SAT from DRAT proofs

Jesse Michael Han

University of Pittsburgh
`jmh288@pitt.edu`

We learn a variable selection heuristic for the cube-and-conquer paradigm in SAT solving by training the NeuroCore architecture to predict variable occurrence counts in DRAT proofs of unsatisfiable formulas. We evaluate our models by averaging CDCL runtimes on the subproblems produced by branching on their predictions, and also by their average scores in the Prover-Adversary game against a random adversary. As a baseline, we compare with Z3's implementation of the `march_cu` variable selection heuristic. Our results indicate that training to predict DRAT variable counts usually outperforms training to predict occurrence of a variable in an unsat core. On all three evaluation datasets, our best models outperform `march_cu` on solution time, and on two, they achieve superior performance on the game-based metric.[1]

Introduction *Cube-and-conquer* [?] is a relatively new SAT solving paradigm wherein a lookahead solver makes expensive, globally-informed decisions on how to partition (cube) a SAT problem into subproblems, which are then solved (conquered) in parallel by CDCL solvers. It has been used to prove the unsatisfiability of relatively small (but hard for CDCL) combinatorial SAT problems [?, ?, ?]. While previous approaches [?, ?, ?, ?] to improving SAT solvers with neural networks have tried integrating variable and literal selection directly into the run of a CDCL solver, we propose targeting cubing heuristics, which allow for fewer but more expensive and impactful decisions.

A cubing heuristic comprises a variable selection heuristic and a cutoff heuristic. After each variable selection, two new leaves are added to the search tree, corresponding to either assignment of the variable. After propagating the assignments, the cutoff heuristic examines the resulting formulas at the leaves and decides whether or not they are easy for CDCL. If it deems a leaf to be easy (or if it has exceeded a budget of cubes), the cutoff heuristic freezes it. This process is repeated on the hardest unfrozen leaf. This produces a truncated search tree whose leaves are the cubes. In our present work, we target only the variable selection heuristic by querying our models for the top K variables and producing 2^K cubes. In practice, this leads to poor parallelization [?] as K scales due to a few disproportionately hard problems near the root, so we use $K = 1$ and $K = 3$, and evaluate our models by averaged runtime on the leaves.

As observed in unpublished work by Selsam [?], the job of the cutoff heuristic is essentially to estimate the size of the DPLL search tree beneath a leaf. Knuth [?] showed that the size of a backtracking search tree can be estimated by the lengths of randomly sampled paths through the tree. We recast this in terms of playouts in a two-player zero-sum game, known in the literature as the *Prover-Adversary game* [?]. At each round, player 1 (Prover) picks an unassigned variable, and player 2 (Adversary) assigns it. The game ends when either all clauses are satisfied or some clause is unsatisfied by the trail of assignments. The terminal value of the game is the number of rounds divided by the number of variables; for an unsat formula, player 1 seeks to minimize this, and player 2 seeks to maximize this. We modify the game so that unit propagation occurs after every round; then every playout is a path through the DPLL search tree. Urquhart [?] proved that player 1 has a winning strategy in fewer than K rounds iff there is a resolution proof of unsat of depth $\leq K$. A good policy for player 1 will thus guide the

[1] https://www.github.com/jesse-michael-han/neurocuber-public/

game towards shallower parts of the search tree where conflicts occur relatively quickly. This is exactly the behavior we desire from the variable selection heuristic of a cuber. We additionally evaluate our models with the average terminal value of playouts against a random adversary, where our models queried at every round for Player 1's policy. This metric provides a more robust evaluation of our models' decisions, as they are queried dozens to hundreds of times per formula versus only once; also, unlike the timing-based metric, this is unaffected by resource contention.

DRAT proofs Resolution proofs emitted by SAT solvers quickly become enormous as problems scale. A DRAT proof [?] emitted by a modern CDCL solver is essentially an extremely compressed resolution proof: each line in the proof will typically be a learned conflict clause abbreviating dozens or hundreds of unit propagation steps. DRAT proofs can thus be roughly thought of as the SAT analogue of a tactic proof script for interactive theorem provers in higher logics: a list of high-level non-deterministic steps which can be formally linked together by more primitive automation (unit propagation). The intuition behind our approach is that DRAT proofs are a readily-available high-quality representation of resolution trees, and if a variable occurs frequently in a resolution tree, branching on it will correspondingly minimize the average size of the resolution trees (and proportionally the solving times) for the leaves.

Network architecture, data, and training Our implementation is based on the simplified NeuroSAT [?] architecture used by Selsam and Bjørner [?] to guide CDCL solvers through periodic refocusing of EVSIDS scores [?]. They trained a variable scoring head and a clause scoring head to predict the variables and clauses in a labelled unsat core. Besides minor modifications to the GNN embedding network, we add another variable scoring head, and train it to predict occurrence counts of variables in DRAT proofs. The loss function is calculated by softmaxing the true occurrence counts and logits and taking the forward KL-divergence. We train to minimize the sum of all three losses . All models are implemented in TensorFlow 2. We trained on a synthetic dataset `src` of 250000 problems, based on the problem distributions **SR** and **SRC** described in [?] as follows: first, we extract an unsat core C of size ≥ 20 and ≤ 100 from a problem in $\mathbf{SR}(20)$, modified to exclude binary clauses to increase the difficulty of the core, and then sample a formula from $\mathbf{SRC}(100, C)$ which is between 5 to 20 times larger than C. As a baseline, we also trained a separate model on another dataset `sr` of 250000 unsatisfiable problems drawn from $\mathbf{SR}(\mathbf{U}(10, 40))$. We obtain variable occurrence counts from DRAT proofs (excluding deletion clauses) emitted by the state-of-the-art CDCL solver `cadical` [?], and extract unsat cores by verifying the proofs with `drat-trim` [?].

Evaluations We evaluate on three datasets `ramsey`, `schur`, and `vdw` of 1000 random subproblems each (randomly assigning 5, 35, and 3 variables) of the hard combinatorial problems $\mathsf{Ramsey}(4, 4, 18)$, $\mathsf{Schur}(4, 45)$, and $\mathsf{vanderWaerden}(2, 5, 179)$. The problems range in size from ~ 3000 to ~ 7800 clauses. For timing evaluation, we query each variable selection head of each model once on the first 250 problems in each dataset, picking the top $K = 1, 3$ scored variables, then recording the solving time of the cubes in parallel with as many cores as cubes; we used `cadical` as the conqueror. We only queried `march_cu` for $K = 1$. The runs were performed serially on a 16-core machine with no other compute-intensive tasks. For random playout evaluation, we play 50 matches on all formulas on all datasets and record the average terminal values and average number of unit propagations after every round. We used the distribution framework `ray` to parallelize up to 16 playouts at once per run; all runs were done in parallel on the PSC Bridges cluster.

	src_core	src_drat	random	march_cu
avg terminal value	0.144	0.14	0.192	0.146
avg unit props	1.415	1.471	1.064	1.873

Table 1: Average terminal values and unit propagations for all variable selection heuristics after 50000 playouts vs. a random Adversary on `ramsey`. Lower terminal values are better.

	src_core	src_drat	random	march_cu
ramsey	4.345	4.025	5.248	4.83
schur	1.504	1.517	2.392	1.903
vdw	1.843	1.803	2.215	2.07

Table 2: Averaged wall clock runtimes (in seconds) for top-1 cubing of all variable selection heuristics. On average, our best heuristics produce an 18% speedup over `march_cu`.

Results We use the naming convention `neurocuber-<train_dataset>_<head>` to refer to our learned variable selection heuristics. On all test datasets, both `neurocuber-src_drat` and `neurocuber-sr_drat` outperform `march_cu` on top-1 timing evaluation (Table 2). Table 1 shows the result of random playout evaluation on 1000 subproblems of $\mathsf{Ramsey}(4, 4, 18)$. Remarkably, even though `march_cu` finds significantly more unit propagations than our models, it is still outperformed by `neurocuber-src`. The same phenonenon occurs on the `schur` dataset. Only on `vdw` does `march_cu` achieve the best terminal value. Table 3 and Table 4 show the average percent change in performance of the DRAT-variable over core-variable scoring heads for both models. On `ramsey` and `vdw`, DRAT-variable heads outperformed core-variable heads across all models and metrics, with significant improvement for `neurocuber-sr` across the board.

Conclusions and future work The comparative timing performance of our models was tightly correlated with their terminal scores in the Prover-Adversary game, providing empirical evidence that good policies for player 1 translate to good variable selection heuristics for cube-and-conquer. Our experiments show that training to predict DRAT variable counts consistently yields better variable selections than training to predict the binary occurrence of variables in unsat cores. Remarkably, on some hard combinatorial problems squarely in the domain of cube-and-conquer, our strongest models outperform domain-specific heuristics without optimizing as much for unit propagation. While maximizing the number of expected unit propagations is an obvious short-range policy in the Prover-Adversary game (and far more sophisticated versions of this are currently state-of-the-art for cube-and-conquer), our experiments suggest that better policies can be learned, even through just supervised training on proxy targets. The natural next step is direct reinforcement learning of the policy and value functions. We will discuss steps in this direction during our talk.

	ramsey	schur	vdw
top1 timing	24.06	6.3	29.19
top3 timing	55.24	30.31	59.42
random playout	16.38	8.77	5.78

Table 3: Percent improvement of DRAT over core-var heads for `neurocuber-sr`.

	ramsey	schur	vdw
top1 timing	7.3	-1.4	2.06
top3 timing	17.42	0.1	9.96
random playout	2.91	-0.54	5.14

Table 4: Percent improvement of DRAT over core-var heads for `neurocuber-src`.

Acknowledgements We thank Daniel Selsam and Nikolaj Bjørner for initially suggesting the approach taken in this paper. This work benefited from conversations with Tom Hales, Emre Yolcu, Daniel Abolafia, and Ruben Martins. This work was supported by XSEDE start up grant TG-DMS190028 and grant G-2018-10067 from the Sloan Foundation.

References

[1] Armin Biere. CaDiCaL simplified satisfiability solver. `http://fmv.jku.at/cadical/`.

[2] Marijn J. H. Heule. The DRAT format and DRAT-trim checker. *CoRR*, abs/1610.06229, 2016.

[3] Marijn J. H. Heule. Avoiding triples in arithmetic progression. *Journal of Combinatorics*, 8(3):391–422, 2017.

[4] Marijn J. H. Heule. Schur number five. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6598–6606, 2018.

[5] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 228–245, 2016.

[6] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving very hard problems: Cube-and-conquer, a hybrid SAT solving method. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4864–4868, 2017.

[7] Sebastian Jaszczur, Michał Łuszczyk, and Henryk Michalewski. Neural heuristics for SAT solving. In *Representation Learning on Graphs and Manifolds Workshop at ICLR 2019*, 2019.

[8] Donald E Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29(129):122–136, 1975.

[9] Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Improving SAT solver heuristics with graph networks and reinforcement learning. *arXiv preprint arXiv:1909.11830*, 2019.

[10] Gil Lederman, Markus N Rabe, Edward A Lee, and Sanjit A Seshia. Learning heuristics for automated reasoning through deep reinforcement learning. *arXiv preprint arXiv:1807.08058*, 2018.

[11] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535, 2001.

[12] Pavel Pudlák. Proofs as games. *The American Mathematical Monthly*, 107(6):541–550, 2000.

[13] Daniel Selsam. Neurocuber: training NeuroSAT to make cubing decisions for hard SAT problems.

[14] Daniel Selsam and Nikolaj Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, pages 336–353, 2019.

[15] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[16] Alasdair Urquhart. The depth of resolution proofs. *Studia Logica*, 99(1-3):349, 2011.

[17] Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 422–429, 2014.