Learning cubing heuristics for SAT from DRAT proofs

Jesse Michael Han

AITP 2020

University of Pittsburgh

Outline

The Boolean satisfiability problem (SAT)

Can we assign variables to $\{\text{true}, \text{false}\}$ and satisfy all clauses?

$$
\begin{aligned}
&(x_5 \lor x_8 \lor \bar{x}_2) \land (x_2 \lor \bar{x}_1 \lor \bar{x}_3) \land (\bar{x}_8 \lor \bar{x}_3 \lor \bar{x}_7) \land (\bar{x}_5 \lor x_3 \lor x_8) \land \\
&(\bar{x}_6 \lor \bar{x}_1 \lor \bar{x}_5) \land (x_8 \lor \bar{x}_9 \lor x_3) \land (x_2 \lor x_1 \lor x_3) \land (\bar{x}_1 \lor x_8 \lor x_4) \land \\
&(\bar{x}_9 \lor \bar{x}_6 \lor x_8) \land (x_8 \lor x_3 \lor \bar{x}_9) \land (x_9 \lor \bar{x}_3 \lor x_8) \land (x_6 \lor \bar{x}_9 \lor x_5) \land \\
&(x_2 \lor \bar{x}_3 \lor \bar{x}_8) \land (x_8 \lor \bar{x}_6 \lor \bar{x}_3) \land (x_8 \lor \bar{x}_3 \lor \bar{x}_1) \land (\bar{x}_8 \lor x_6 \lor \bar{x}_2) \land \\
&(x_7 \lor x_9 \lor \bar{x}_2) \land (x_8 \lor \bar{x}_9 \lor x_2) \land (\bar{x}_1 \lor \bar{x}_9 \lor x_4) \land (x_8 \lor x_1 \lor \bar{x}_2) \land \\
&(x_3 \lor \bar{x}_4 \lor \bar{x}_6) \land (\bar{x}_1 \lor \bar{x}_7 \lor x_5) \land (\bar{x}_7 \lor x_1 \lor x_6) \land (\bar{x}_5 \lor x_4 \lor \bar{x}_6) \land \\
&(\bar{x}_4 \lor x_9 \lor \bar{x}_8) \land (x_2 \lor x_9 \lor x_1) \land (x_5 \lor \bar{x}_7 \lor x_1) \land (\bar{x}_7 \lor \bar{x}_9 \lor \bar{x}_6) \land \\
&(x_2 \lor x_5 \lor x_4) \land (x_8 \lor \bar{x}_4 \lor x_5) \land (x_5 \lor x_9 \lor x_3) \land (\bar{x}_5 \lor \bar{x}_7 \lor x_9) \land \\
&(x_2 \lor \bar{x}_8 \lor x_1) \land (\bar{x}_7 \lor x_1 \lor x_5) \land (x_1 \lor x_4 \lor x_3) \land (x_1 \lor \bar{x}_9 \lor \bar{x}_4) \land \\
&(x_3 \lor x_5 \lor x_6) \land (\bar{x}_6 \lor x_3 \lor \bar{x}_9) \land (\bar{x}_7 \lor x_5 \lor x_9) \land (x_7 \lor \bar{x}_5 \lor \bar{x}_2) \land \\
&(x_4 \lor x_7 \lor x_3) \land (x_4 \lor \bar{x}_9 \lor \bar{x}_7) \land (x_5 \lor \bar{x}_1 \lor x_7) \land (x_5 \lor \bar{x}_1 \lor x_7) \land \\
&(x_6 \lor x_7 \lor \bar{x}_3) \land (\bar{x}_8 \lor \bar{x}_6 \lor \bar{x}_7) \land (x_6 \lor x_2 \lor x_3) \land (\bar{x}_8 \lor x_2 \lor x_5)
\end{aligned}
$$

The Boolean satisfiability problem (SAT)

- Prototypical NP-complete problem
- Modern SAT solvers are:
  - highly engineered and capable of handling problems with millions of variables and clauses
  - dominated by the conflict-driven clause learning (CDCL) paradigm: backtracking tree search + learning *conflict clauses* to continuously prune the search space
  - mostly use cheap *branching heuristics*: which variable to assign next?

These branching heuristics are appealing targets for machine learning methods (e.g. neural networks).

Branching heuristics are appealing targets

- Simple enough:
  - find some way to embed the formula $\mathcal{F}$
  - obtain embeddings for each variable, get logits by applying a FFN
  - select $v_{\text{next}} \leftarrow \pi(\mathcal{F})$; assign; propagate; repeat
  - simple MDP formulation with a binary terminal reward
    - Note that reward is sparse, need some curriculum/reward engineering
- Important problem:
  - SAT solvers routinely operate on astronomically-sized problems
  - Even on problems with merely $\approx$1M clauses, querying a neural network for every branching decision is impractical
- Solution:
  - query less frequently
  - but make the network's decision more impactful.

Cube-and-conquer

- Early branching decisions are especially important
- They determine the rest of the search space
  - CDCL solvers use restarts to erase poor early decisions from the assignment stack
- Cube-and-conquer: use an expensive, globally-informed heuristic (a cuber) to make early branching decisions
  - grows a partial search tree, uses CDCL solvers to finish the leaves
  - the cuber has traditionally been a *look-ahead* solver
- Used to great effect by Marijn Heule (Pythagorean triples problem, Schur number five...) on hard unsatisfiable instances

- Idea: replace the cuber with a neural branching heuristic

Learning from DRAT proofs

- The job of the cuber is to minimize solver runtime at the leaves
- SAT solvers can be thought of as resolution proof engines
- Solver runtime directly correlated with size of resolution proof
- So: we want to select variables which minimize the expected size of the resolution proofs for either branch.
- full resolution proofs are huge, but they admit a compact representation (DRAT format) as a sequence of learned clauses

- Idea: prioritize variables that occur frequently in DRAT proofs
  - selecting these variables leads to parts of the search space where conflicts useful for proving unsat will occur more often

Network architecture and training

- For these experiments, we used the NeuroCore architecture, a simplified version of the NeuroSAT graph neural network
- 4 rounds of message passing on the bipartite clause-literal graph
- Train on synthetic random problems only
  - **SR**(n): incrementally sample clauses of varying length until the problem becomes unsat. A single literal in the final clause can be flipped to make the problem sat.
  - **SRC**(n, C): C is an unsatisfiable problem that can be made sat by flipping a single literal. Make C sat, then sample clauses as with SR until the problem is unsat. Then make C unsat again, guaranteeing an unsat core of a certain size.

- Training problems: 250K problems sampled from **SRC**$(100,$ **SR**$(20))$.

Network architecture and training

- A datapoint in our training set is a pair $(\mathbf{G}, c)$ where $\mathbf{G}$ is a sparse clause-literal adjacency matrix and c is a vector of occurrence counts for each variable index.
- Minimize the KL divergence $\mathbf{D}_{\mathrm{KL}}(\mathrm{softmax}(c)||\widehat{\pi})$, where $\widehat{\pi}$ is the probability distribution over variables predicted by the network.
- We also simultaneously trained heads for predicting whether clauses/variables occured in the unsat core.

Evaluation

- Evaluation dataset:
  - `ramsey`: 1000 subproblems, generated by randomly assigning 5 variables, of Ramsey(4,4,18)
  - `schur`: 1000 subproblems, generated by randomly assigning 35 variables, of Schur(4,45)
  - `vdW`: 1000 subproblems, generated by randomly assigning 3 variables, of vanderWaerden(2,5,179)
- Evaluation scheme:
  - Query network once on the entire problem.
  - Split on the top-rated variable, producing two cubes.
  - Primary metric: average solver runtime on the cubes.
  - As a baseline, we compare against Z3's implementation of the `march_cu` cubing heuristic, which performs lookahead with handcrafted features.

Evaluation

	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

Averaged wall-clock runtimes (in seconds) for all variable selection heuristics across all datasets. Our models consistently choose better branching variables than `march_cu`, with lower average runtime on the cubes averaged across all 1000 problems on all three datasets.

Takeaways:

- neural cubers pick better variables
- DRAT provides better signal than unsat cores

Conflict-driven learning

- It's hard to prove `unsat` (exponential-time lower bounds on DPLL)
- Verifying/minimizing a DRAT proof can take even longer than running the solver that produced it
- Supervised learning with DRAT proofs:
  - only labels unsatisfiable problems
  - biases data towards problems easy enough to solve
  - doesn't scale efficiently

- Solution: move beyond DRAT.
- Prioritize variables that optimize conflict clause learning instead.

Conflict-driven learning

- In highly symmetric hard combinatorial problems, there is less sharp preference for specific variables in the final DRAT proofs.
- Better to treat the SAT solver, in these kinds of situations, as a resolution forward chainer. Accelerate clause learning and it might find a path to the empty clause sooner.
- SAT solvers don't produce sophisticated proofs.
  - they chain together a *lot* of simple lemmas from conflict analysis
- Focus on learning useful lemmas sooner. This accelerates performance on `sat` instances also.

The Prover-Adversary game

- A path through the DPLL tree can be phrased in terms of a two-player zero-sum game. In each round,
  - Player 1 picks a variable
  - Player 2 assigns it a value, and it propagates.
  - The terminal value is 1/#rounds. Player 1 wants to minimize, Player 2 wants to maximize.
- Urquhart showed that winning strategies for Player 1 correspond to short resolution proofs of `unsat`.
- In this way, we see that the *conflict efficiency* of our branching heuristics is directly related to the size of the resolution proof.

The Prover-Adversary game

We can empirically validate this with the previous models. We use our variable branching heuristics as the policy for Player 1, and a uniform random policy for Player 2. For all 1000 problems in the `ramsey` dataset, we generate 50 playouts and average the terminal values.

	`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

- Takeaways:
  - our models are more conflict efficient, even while being less propagation-efficient
  - variable heuristic conflict efficiency correlates with solver runtime

Reinforcement learning of glue level minimization

- Glue level is a well-studied proxy metric for the quality of a learned clause. It measures the number of assignments involved in the clause. Clauses with low glue level tend to propagate more frequently, accelerating learning.
- We modify the Prover-Adversary game so that the terminal reward is $1/g^2$, where g is the glue level of the clause that would be learned from conflict analysis.
- View as a reinforcement learning task in a finite episodic MDP and apply policy gradient methods.
- Allows us to ignore the sparse terminal reward and view every path through the DPLL tree as an episode.

It works, at scale

Enhancing SAT solvers with glue variable predictions (arXiv:2007.02559)

- Use both RL formulation and supervised learning formulation — predict variables which show up in learned clauses of minimal glue level, called *glue clauses*
- Use a smaller, lightweight network architecture based on the RL for QBF paper (remember to see Markus' talk!), CPU-only inference
- Target the state-of-the-art solver CaDiCaL with *periodic refocusing* — only periodically update the EVSIDS activity score branching heuristic with network predictions
- Improvements on SATCOMP 2018, SATRACE 2019, and a dataset of SHA-1 preimage attacks
- Work done under the supervision of John Harrison during an intership at the Automated Reasoning Group at AWS.

Thank you for your attention!