Learning Clause Deletion Heuristics with Reinforcement Learning

(with a twist)

Pashootan Vaezipoor[1], Gil Lederman[2], Yuhuai Wu[1],
Roger Grosse[1] and Fahiem Bacchus[1]

[1]University of Toronto

[2]UC Berkeley

September 16, 2020

Goal: Use ML in order to build better SAT solvers to tackle *industrial-scale* problems (e.g., SATCOMP).

Highlight

1. Preamble on SAT Heuristics
2. Earlier Efforts
3. Challenges of Gaining Wall-Clock Improvement on SAT
4. Problem Statement
5. Redemption *(a.k.a. the twist)*

Preamble on SAT Heuristics

Backtracking search algorithms for SAT (DPLL) gradually extend a partial assignment by selecting a new variable to branch on at each decision level. The partial assignment is extended until it becomes a satisfying assignment, or until a *conflict* is reached.

Preamble on SAT Heuristics

Backtracking search algorithms for SAT (DPLL) gradually extend a partial assignment by selecting a new variable to branch on at each decision level. The partial assignment is extended until it becomes a satisfying assignment, or until a *conflict* is reached.

Conflict-driven clause learning (CDCL) combines backtracking search with *clause learning*. While DPLL simply backtracks out of conflicts, CDCL "analyzes" the conflict by performing a couple of resolution steps and "learns" a clause which is added to the list of clauses in order to cut off large parts of the search space and thereby speeds up the search process.

Preamble on SAT Heuristics (cont.)

1. **Clause Deletion:** Audemard and Simon in 2009 devised a new score to identify clauses that are likely to be used more frequently in the future. The *Literals Block Distance (LBD)* of a clause is the number of distinct decision levels of its literals at the time of the conflict analysis.
   Very aggressive clause deletion where half of the learnt clauses are removed every ~ 20000 conflicts, gives a significant boost in performance.
2. **Branching Heuristic:** VSIDS has been the dominant heuristic here, where an *activity* score is kept for each variable. Upon each conflict the activity of variables involved in the conflict are *bumped*, and every variable's activity is periodically *decayed* exponentially.

Earlier Efforts

QBF: Lederman, et al. *"Learning Heuristics for Quantified Boolean Formulas through Reinforcement Learning."*, ICLR 2019.

SAT:

	NeuroSAT	NeuroCore	GQSAT	"deep" WalkSAT
Training Alg.	Supervised	Supervised	DQN	REINFORCE
GNN?	✔	✔	✔	✔
Target Heuristic	–	Branching	Branching	Branching
Completeness	✖	✔	✔	✔
Problem Size	(40, –)[1]	Scheduling Problems	(300, 1000)	rand(50, 213)
CDCL-based?	✖	✔	✔	✖

[1] (# variables, # clauses)

Challenges of Gaining Wall-Clock Improvement on SAT

- Problems with offline training:
  - A solver is a dynamic process and changes to the heuristics directly affect the landscape of future observations offline training might fail to capture this non-stationary behaviour.
  - There's a need for a surrogate labeling mechanism for offline learning.

Challenges of Gaining Wall-Clock Improvement on SAT

- Problems with offline training:
  - A solver is a dynamic process and changes to the heuristics directly affect the landscape of future observations offline training might fail to capture this non-stationary behaviour.
  - There's a need for a surrogate labeling mechanism for offline learning.
  - ***[Solution] RL can 1. better capture the dynamic nature of the solver and 2. it allows for training the heuristic directly towards optimizing the desired metric (e.g., number of decisions, running time, etc.)***

Challenges of Gaining Wall-Clock Improvement on SAT

- Problems with offline training:
  - A solver is a dynamic process and changes to the heuristics directly affect the landscape of future observations offline training might fail to capture this non-stationary behaviour.
  - There's a need for a surrogate labeling mechanism for offline learning.
  - ***[Solution] Online Training (RL)***
- Running time and its effect on (online) training
  - The roll out of industrial instances takes hours

Challenges of Gaining Wall-Clock Improvement on SAT

- Problems with offline training:
  - A solver is a dynamic process and changes to the heuristics directly affect the landscape of future observations offline training might fail to capture this non-stationary behaviour.
  - There's a need for a surrogate labeling mechanism for offline learning.
  - ***[Solution] Online Training (RL)***
- Running time and its effect on (online) training
  - The roll out of industrial instances takes hours
  - Synthesizing SAT formulas that are amenable for training is challenging on two fronts:
    - Feasibility: Episode role outs should be reasonably fast
    - Generalizability: formula should preserve some "structural properties" of the real industrial instances

Challenges of Gaining Wall-Clock Improvement on SAT

- Problems with offline training:
  - A solver is a dynamic process and changes to the heuristics directly affect the landscape of future observations offline training might fail to capture this non-stationary behaviour.
  - There's a need for a surrogate labeling mechanism for offline learning.
  - ***[Solution] Online Training (RL)***
- Running time and its effect on (online) training
  - The roll out of industrial instances takes hours
  - Synthesizing SAT formulas that are amenable for training is challenging on two fronts:
    - Feasibility: Episode role outs should be reasonably fast
    - Generalizability: formula should preserve some "structural properties" of the real industrial instances
  - ***[Solution 1] - simplify original formula f by attaching a partial assignment m to it:***
    1. $(SAT?, m) \leftarrow$ ***Solve*** f ***with vanilla*** `Glucose`
    2. ***if*** $SAT?$***:*** $m|_r \leftarrow$ ***random subset of*** m
    3. ***if*** $\neg SAT?$***:*** $m|_r \leftarrow$ ***random assignment to subset of*** $vars(f)$
    4. $f_{simp} = f \wedge \bigwedge_{u \in m|_r} u$

Challenges of Gaining Wall-Clock Improvement on SAT

- Problems with offline training:
  - A solver is a dynamic process and changes to the heuristics directly effect the landscape of future observations offline training might fail to capture this non-stationary behaviour.
  - There's a need for a surrogate labeling mechanism for offline learning.
  - ***[Solution] Online Training (RL)***
- Running time and its effect on (online) training
  - The roll out of industrial instances takes hours
  - ***[Solution 1] Generate simplified instances***
  - ***[Solution 2] Use a surrogate reward (e.g., 1/glue level of conflict clause learnt, J. Han "Enhancing SAT solvers with glue variable predictions"***

Challenges of Gaining Wall-Clock Improvement on SAT

- Problems with offline training:
  - A solver is a dynamic process and changes to the heuristics directly effect the landscape of future observations offline training might fail to capture this non-stationary behaviour.
  - There's a need for a surrogate labeling mechanism for offline learning.
  - ***[Solution] Online Training (RL)***
- Running time and its effect on (online) training
  - The roll out of industrial instances takes hours
  - ***[Solution 1] Generate simplified instances***
  - ***[Solution 2] Use a surrogate reward***
- Industrial instances are huge making feasibility of GNNs challenging

Challenges of Gaining Wall-Clock Improvement on SAT

- Problems with offline training:
  - A solver is a dynamic process and changes to the heuristics directly effect the landscape of future observations offline training might fail to capture this non-stationary behaviour.
  - There's a need for a surrogate labeling mechanism for offline learning.
  - ***[Solution] Online Training (RL)***
- Running time and its effect on (online) training
  - The roll out of industrial instances takes hours
  - ***[Solution 1] Generate simplified instances***
  - ***[Solution 2] Use a surrogate reward***
- Industrial instances are huge making feasibility of GNNs challenging
  - ***[Solution] Either crop the problem graph or send delta updates or possibly partition the graph and use parallelism.***

Challenges of Gaining Wall-Clock Improvement on SAT (Cont.)

- Performance trade-offs during inference
  - Millions of decisions can be made in a few seconds by SAT solvers (branching)
  - Forward pass of the NN can be expensive

Challenges of Gaining Wall-Clock Improvement on SAT (Cont.)

- Performance trade-offs during inference
  - Millions of decisions can be made in a few seconds by SAT solvers (branching)
  - Forward pass of the NN can be expensive
  - ***[Solution] Less frequent queries to the model, e.g., 1. Periodic Refocusing; 2. Clause Reduction***

Challenges of Gaining Wall-Clock Improvement on SAT (Cont.)

- Performance trade-offs during inference
  - Millions of decisions can be made in a few seconds by SAT solvers (branching)
  - Forward pass of the NN can be expensive
  - ***[Solution] Less frequent queries to the model, e.g., 1. Periodic Refocusing; 2. Clause Reduction***
- Action space explosion (Specifically for Clause Deletion)
  - Clause Reduction receives a list of $N(\sim 2000)$ clauses and returns a binary vector in $\{0,1\}^N$
  - The naive idea of deciding keep/drop per clause makes for an exponential space
  - In policy gradient algorithms that means huge "injected" randomness and huge variance in gradient estimates
  - ***[Solution] Threshold Policy: On every step, the policy outputs a single real positive number. All clauses with LBD score greater or equal to this threshold are deleted.***

Refined Goal: Use *RL* in order to extract *versatile* heuristics for *complete* solvers to tackle *industrial-scale* problems (e.g., SATCOMP).

Problem Statement

The environment $\mathcal{E}$ is a *Markov Decision Process (MDP)* with states $\mathcal{S}$, action space $\mathcal{A}$, and rewards per time step $r_t \in \mathbb{R}$. In our setting:

- $\mathcal{E}$ is the modern SAT Solver `Glucose` along with a set of formulas

Problem Statement

The environment $\mathcal{E}$ is a *Markov Decision Process (MDP)* with states $\mathcal{S}$, action space $\mathcal{A}$, and rewards per time step $r_t \in \mathbb{R}$. In our setting:

- $\mathcal{E}$ is the modern SAT Solver `Glucose` along with a set of formulas
- Each step is equivalent to one garbage collection

Problem Statement

The environment $\mathcal{E}$ is a *Markov Decision Process (MDP)* with states $\mathcal{S}$, action space $\mathcal{A}$, and rewards per time step $r_t \in \mathbb{R}$. In our setting:

- $\mathcal{E}$ is the modern SAT Solver `Glucose` along with a set of formulas
- Each step is equivalent to one garbage collection
- An *episode* is the result of the interaction of the agent with $\mathcal{E}$ while solving a formula f

Problem Statement

The environment $\mathcal{E}$ is a *Markov Decision Process (MDP)* with states $\mathcal{S}$, action space $\mathcal{A}$, and rewards per time step $r_t \in \mathbb{R}$. In our setting:

- $\mathcal{E}$ is the modern SAT Solver `Glucose` along with a set of formulas
- Each step is equivalent to one garbage collection
- An *episode* is the result of the interaction of the agent with $\mathcal{E}$ while solving a formula f
  - *Complete*: Solver solved f successfully
  - *Incomplete*: Solver was aborted due to some *termination criteria*

Problem Statement (Cont.)

The environment $\mathcal{E}$ is a *Markov Decision Process* (MDP) with states $\mathcal{S}$, action space $\mathcal{A}$, and rewards per time step $r_t \in \mathbb{R}$. In our setting:

- The *observations* are a set of features that capture the dynamic state of the solver, such as: *average trail size*, *ratio of the learned clauses*, *mean decision level* and *LBD histogram* of the last t steps *(No GNN initially!)*

Problem Statement (Cont.)

The environment $\mathcal{E}$ is a *Markov Decision Process* (MDP) with states $\mathcal{S}$, action space $\mathcal{A}$, and rewards per time step $r_t \in \mathbb{R}$. In our setting:

- The *observations* are a set of features that capture the dynamic state of the solver, such as: *average trail size*, *ratio of the learned clauses*, *mean decision level* and *LBD histogram* of the last t steps *(No GNN initially!)*
- The *reward* is a *deterministic* operation counter acting as a surrogate for solver's performance.

(Negative) Results

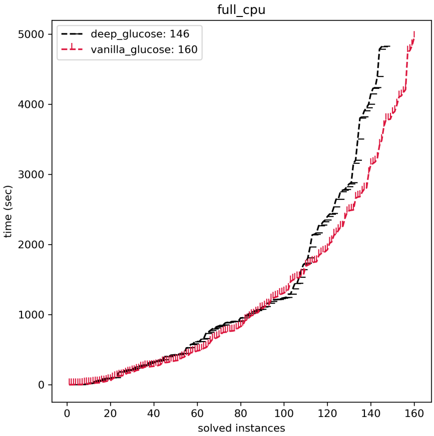


Figure 1: We trained on simplified formulas from past few SAT Competitions. The result is compared with `Glucose` on SAT competition 2018:

Culprits (conjectures)

- Clause deletion is a less direct method to change solver's behaviour.
- LBD is already too good and the ceiling is not high enough.
- The cost of querying a "better policy" is not justified by its benefits.

The Twist – `Neuro#`[2]

We used our framework to target another problem:

1. Target another domain with smaller problem sizes: #SAT
   - At each step the model only sees a subset of the entire graph (component) making GNNs more feasible.

[2] *"Learning Branching Heuristics for Propositional Model Counting"* (arxiv.org/abs/2007.03204)

The Twist – `Neuro#`[2]

We used our framework to target another problem:

1. Target another domain with smaller problem sizes: #SAT
   - At each step the model only sees a subset of the entire graph (component) making GNNs more feasible.
2. Target branching heuristic

[2] *"Learning Branching Heuristics for Propositional Model Counting"* (arxiv.org/abs/2007.03204)

The Twist – Neuro#[2]

We used our framework to target another problem:

1. Target another domain with smaller problem sizes: #SAT
   - At each step the model only sees a subset of the entire graph (component) making GNNs more feasible.
2. Target branching heuristic
3. Adopt Evolution Strategy (ES) instead of Policy Gradient
   - ES injects randomness in weight space, which for us is much smaller and most importantly, independent of number of clauses/variables. (perhaps applicable to clause deletion too)
   - No backprop (minor advantage)

[2] *"Learning Branching Heuristics for Propositional Model Counting"* (arxiv.org/abs/2007.03204)

The Twist – `Neuro#`[2]

We used our framework to target another problem:

1. Target another domain with smaller problem sizes: #SAT
   - At each step the model only sees a subset of the entire graph (component) making GNNs more feasible.
2. Target branching heuristic
3. Adopt Evolution Strategy (ES) instead of Policy Gradient
   - ES injects randomness in weight space, which for us is much smaller and most importantly, independent of number of clauses/variables. (perhaps applicable to clause deletion too)
   - No backprop (minor advantage)
4. Go distribution-specific!
   - Target a specific family of problems
   - Use a high-level generative procedure to produce problems of varying sizes (small problems for training)
   - *No an unreasonable assumption in industry.*

[2] *"Learning Branching Heuristics for Propositional Model Counting"* (arxiv.org/abs/2007.03204)

Neuro# – Results

Cellular:

GridWorld:

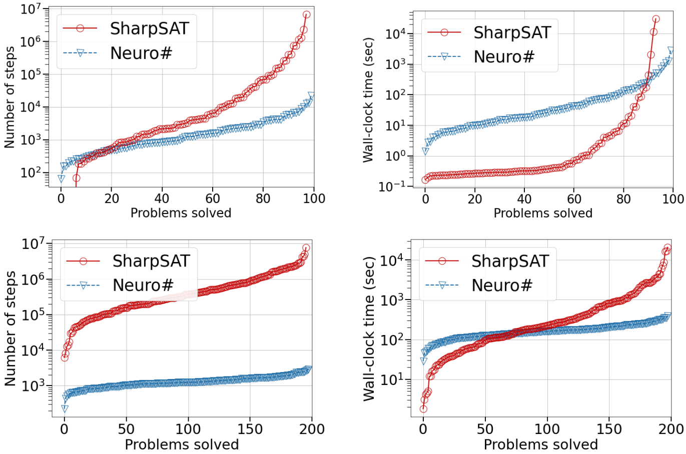
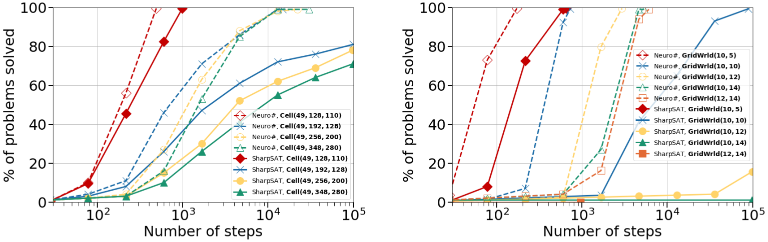


Figure 2: Trained on instances of size $6k/25k$ (Cellular) and $300/1k$ (Grid). Results are tested on instances of size $25k/102k$ (Cellular) and $2k/6k$ (Grid).

Neuro# – Results



Cellular GridWorld

Figure 3: Neuro# generalizes well to larger problems. Compare the robustness of Neuro# vs. SharpSAT as the problem sizes increase. Solid and dashed lines correspond to SharpSAT and Neuro#, respectively. All episodes are capped at 100k steps.

Q&A

Thanks!