Explaining SAT Solving using Causal Reasoning

Jiong Yang[1], **Arijit Shaw**[2,3], **Teodora Baluta**[1]
Mate Soos[4], **Kuldeep S. Meel**[1,3]

[1] Georgia Tech [2] Chennai Mathematical Institute
[3] University of Toronto [4] Ethereum Foundation

SAT: BOOLEAN SATISFIABILITY
An important but hard problem

Boolean Satisfiability (SAT) asks whether there exists an assignment of truth values that satisfies a given Boolean formula. Despite the problem being **NP-complete**, modern SAT solvers solve large industrial problems in seconds.

SAT solvers are complex "black boxes" that use various heuristics to drive performance. Our work uses causal reasoning to uncover which solver features truly cause better performance.

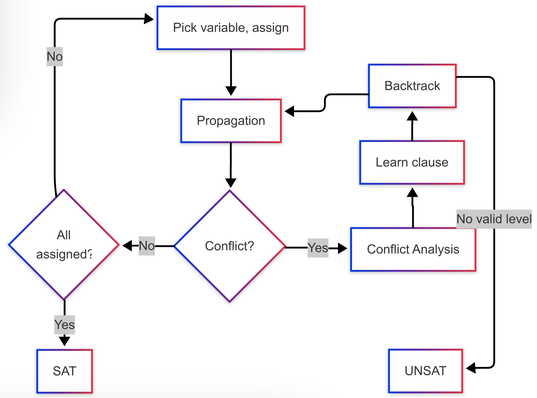
APPLICATIONS OF SAT

- **Hardware and software verification:** providing formal guarantees for bug-freeness, detecting design errors.
- **Theorem provers** have solvers as an indispensable component.
- **Optimization**: Planning, scheduling, resource allocation.
- **Bio-informatics:** haplotype inference, protein folding.
- **Cryptography:** validating encryption, detect vulnerabilities.

SAT Solvers are simple in theory, but complex in implantation

SAT SOLVER IN THEORY

Conflict Driven Clause Learning (CDCL) Algorithm



SAT SOLVER IN PRACTICE

Implements the same algorithm, but much complicated. For example, Kissat, a modern SAT solver consists of more than 40K lines of code. Relies on several complex heuristics for:

- **Variable Selection:** Deciding which variable to branch on next, and which value to assign.
- **Restart Mechanisms:** Periodically resetting the solver's state to explore different parts of the search space.
- **Clause Deletion:** Learned clauses are vital for performance, but they accumulate quickly. Removing low-value clauses is crucial to control memory usage.

THE MYSTERIOUS SAT SOLVERS

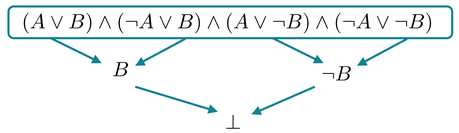
We have little understanding of why they perform well only sometimes.

- Modern SAT solvers can often handle huge problems in seconds, and get stuck on small benchmarks.
- Complexity-theoretic analyses focus on worst-case behaviour, but do not explain the surprising success.
- Widely used heuristics are often justified by empirical *rules of thumb*, but do not explain why they fail.

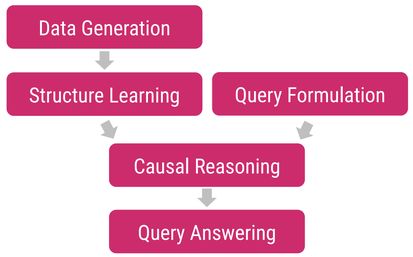
In this work, we work on solving this gap. Specifically, we ask: **what causes a learnt clause useful?**

Our framework uses causal reasoning to uncover how solver components interact and affect clause utility.

EXAMPLE BOOLEAN FORMULA

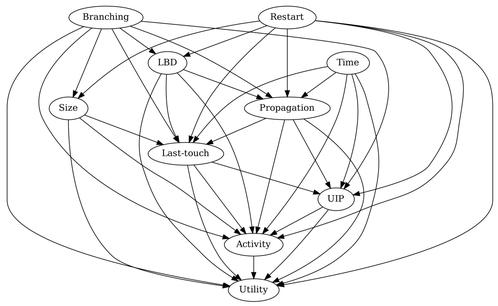


FRAMEWORK



STRUCTURE LEARNING

We build a fully directed causal graph from observational data using hill-climbing algorithm.



- Use the backdoor algorithm to identify the minimal set of controlled variables that d-separates treatment from outcome, eliminating confounding effects.
- Convert the ATE query into an *estimand* expression by summing over the controlled variables, leveraging the backdoor set.
- Apply linear regression to estimate the causal effect from the observational data.
- Validate the robustness of the estimation using refutation tests, such as adding an independent random common cause.

WHAT'S OUR DATA?

- It is not easy to intervene on SAT solvers, so we depend on observational data.
- We modify a modern SAT solver to record detailed features for every learned clause.
- Run solver on bunch of unsatisfiable formulas, multiple times with different heuristics.
- Look at the proof of unsatisfiability to determine which learnt clauses has been useful.
- Our data looks like the following.

Branching	Size	LBD	Time	Utility
Maple	4	2	1000	10
VSIDS	7	3	10000	2
Maple	3	2	100	100

Heuristic used in solver — Features about the learnt clause — How many times the clause has been used?

QUERY + CAUSAL REASONING

We encoded a few *rules of thumb,* and few interesting questions as three types of queries:

1. Average Treatment Effect (ATE)

$$\mathrm{ATE}(X, Y, a, b) = \mathbb{E}[Y|do(X = a)] - \mathbb{E}[Y|do(X = b)]$$

2. Conditional Average Treatment Effect (CATE)

$$\mathrm{CATE}(X, Y, W, a, b) = \mathbb{E}[Y|do(X = a), W] - \mathbb{E}[Y|do(X = b), W]$$

3. Average CATE

Question	Query	Conclusion
Does a low-LBD clause have greater utility?	ATE(Utility, LBD, 1) = −0.26 < 0	Low-LBD clause has greater utility.
Does a clause with high LBD experience a rapid drop in utility over time?	CATE(Utility, Time, 10000, LBD > 6) = −0.09 < 0 CATE(Utility, Time, 10000, LBD ≤ 6) = 0.38 > 0	High-LBD clause experiences a rapid drop in utility over time.
Does a small clause have greater utility? What if the LBD is fixed?	ATE(Utility, Size, 1) = −0.03 < 0 ACATE(Utility, Size, 1, LBD) = −0.02 < 0	Small clause has greater utility, which also holds when LBD is fixed.

We use causality to demystify the solver for an NP-complete problem.

OPEN QUESTIONS

- **Explain Hardness:** Can our framework be extended to reveal why certain problems are easy while others are inherently hard?
- **Improve Solvers:** How can we enhance solver performance by leveraging data-driven insights rather than relying solely on expert intuition?
- **Domain-Specific Solvers:** Given the general-purpose nature of solvers, is it possible to fine-tune heuristics for specific domains using domain-specific data?

arijit@cs.toronto.edu

arxiv.org/abs/2306.06294