

# Perspectives on Practice and Theory of SAT Solving

**Vijay Ganesh**  
**University of Waterloo, Canada**

**SAT Semester @ Simons Institute,  
Berkeley  
Feb 10<sup>th</sup>, 2021**

# Talk Outline

- The central question in SAT/SMT solving – “Unreasonable Effectiveness of SAT/SMT Solvers” for many classes of industrial instances. At the same time, slow for others (e.g., crypto)
- Structural view of industrial instances. Current-best understanding and pitfalls
- Solvers = proof systems + machine learning view
- Separation results for solver configurations (e.g., CDCL with restarts and without restarts)
- From practice to theory – leveraging solvers for theory
- Open problems and directions
- Workshop talks



## PART I



Context and Motivation  
The Central Question in  
Solving

# The Central Question in Solver Research

## Why are Solvers Efficient?

- Why are SAT solvers efficient? After all, the Boolean SAT problem is NP-complete, believed to be intractable.
- Clearly there is a gap between the map (complexity theory) and the territory (algorithm). Can we close this gap between theory and practice in a way that not only deepens our theoretical understanding, but also paves way for better solver design? (We can leverage already well-developed complexity-theoretic tools such as parameterized complexity to close this gap.)
- Finally, we know of classes of instances obtained from applications that are hard for solvers (e.g., Crypto). How do we explain the difficulty of solving such instances?
- Many of these questions have been open for over two decades.

## The Generality of the Central Question

This question  
also applies to  
SMT, CP,...

- Answering this question for SAT would also impact more general settings, since this phenomena applies to many formal reasoning systems, e.g., CP, QBF, model checkers, SMT solvers, first-order provers,..., where problems range from NP-complete, PSPACE-complete, NEXPTIME-complete,... (Also applies to optimization cousins ILP, MAXSAT,...)
- This phenomenon extends to many other settings, where “practical” algorithms seem to defy theoretical expectations set by worst-case analysis
- Richard Karp and many other researchers have suggested that this gap between the success of “practical” algorithms and the worst-case complexity-theoretic view of algorithms is one of the most important questions in the context of algorithm design and complexity

## Sub- questions Why are Solvers Efficient?

- As has been discussed already during the bootcamp (see talks by Sam Buss and others), CDCL solvers are best modelled as proof search systems
- Key result: CDCL solvers are polynomially equivalent to general resolution [PD'11, AFT'11]
- While this equivalence provides a framework for analysis of CDCL solvers, it doesn't answer the question of why solvers are efficient on some classes of instances
- Hence, the question of “why solvers are efficient” can be pared down to:
  - What formula structure/parameters (and range of values for them) result in short proofs (for UNSAT formulas)?
  - Why is it possible for CDCL solvers to find such short proofs efficiently (i.e., automatable under appropriate parameterization)
  - Recent interesting result: Resolution is not automatable, unless P=NP [AM'20]

## Target Conjectures and Results Why are Solvers Efficient?

- Parameterized upper bounds for CDCL algorithm
  - One possibility is  $2^{f(p_1, p_2)} \cdot \text{poly}(n)$  for appropriate values of “real-world” parameters  $p_1, p_2$
  - Alternatively, similar upper-bound for CDCL as PPSZ, but with “real-world” parameters
  - Parameterized understanding of proof search and proof size
- “Better” practically-inspired lower bounds for Res (e.g., why are crypto instances hard for solvers?)
- Better understanding of solver configurations (e.g., CDCL - restarts < CDCL + restarts,...)
- Better solver design (e.g., stronger proof systems integrated into solvers, better ML-based proof rule sequencing and selecting methods,...)



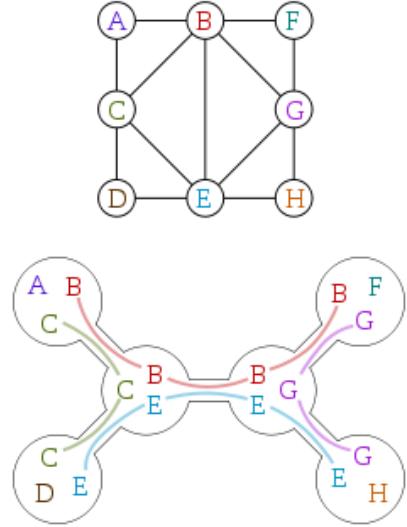
## PART II



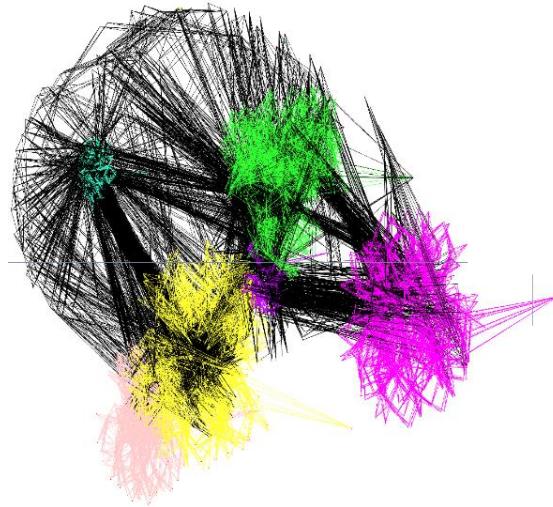
In Search of Empirical  
Understanding of Solvers  
via Parameters

# Search for Complexity-theoretic Parameters

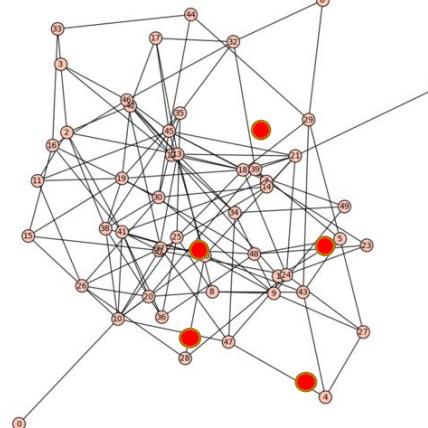
## Key Question



Treewidth



Community Structure



Backdoors

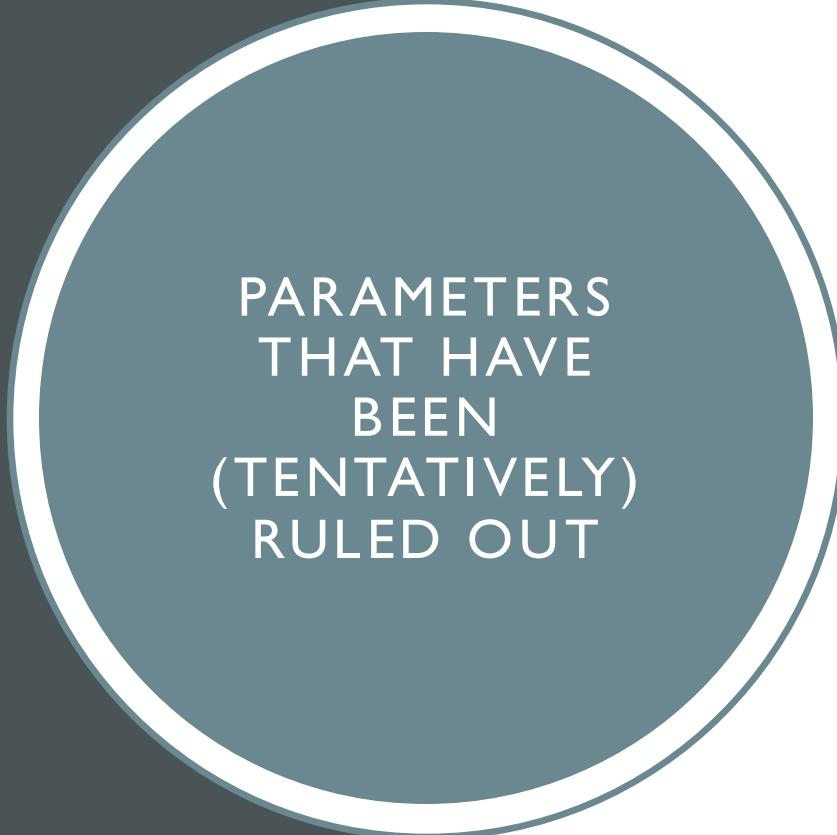
**Q:** Industrial instances have structure that is exploited by solvers. The question then is which of the proposed structural parameters have the best explanatory power, across a wide variety of instances?

The goal of empirical work is to either rule-out or rule-in parameters for further theoretical analysis



## COMPLEXITY- THEORETIC UNDERSTANDING OF SOLVERS EMPIRICAL APPROACH TO IDENTIFY GOOD PARAMETERS

- Final goal: Discover parameters that withstand a battery of well-designed empirical tests and are primed for further theoretical analysis. (it goes without saying that empirical analysis can never be conclusive or complete, because experiments are inherently contingent on a many factors such as sample size, solver settings, tools used, experimental design, etc. Also, the best we can hope for is correlations)
- Parameter must enable us to classify different kinds of SAT formulas into categories such as random, crafted, crypto, verification etc. (i.e., there must be evidence that real-world instance possess the appropriate structure)
- For appropriate range of values, the parameters must correlate well with solver running time (i.e., parameters useful as features in empirical hardness models)
- Instance-generator based scaling studies must validate our hypotheses



## PARAMETERS THAT HAVE BEEN (TENTATIVELY) RULED OUT

- CVR – Lots of excitement in the 1990's when the phase transition phenomenon was discovered [CKT'91], followed by the easy-hard-easy pattern [MSL'92]. However, it was later shown that this does not hold up under scaling studies [CDASV'03]
- Treewidth – Unfortunately, empirically treewidth does not correlate with solver runtime [Mateescu'11]
- Pathwidth/branchwidth – These are polynomially related to treewidth and hence unlikely to serve as good parameters (for the same reason treewidth fails)
- Backdoor and many variants – Similar problem as treewidth. Instances with small backdoor hard to solve, large backdoors easy to solve [Zulkoski'18]
- Community structure – While promising empirically, it is easy to show theoretically that formulas with good community structure can be hard for solvers [Zulkoski'18]
- Many of these parameters are very intuitive and easy to work with theoretically. However, they fall apart upon contact with reality.



## PARAMETERS THAT HAVE NOT YET BEEN RULED OUT

- Merge
  - A generalization of subsumption
- Hierarchical community structure
  - Variant of community structure, but does not have obvious counterexamples (e.g., formulas with “good” hierarchical community structure cannot be expanders)
- Power law
  - Degree of variable occurrences is “highly unbalanced”. A few variables occur very often, but most others occur only a few times

# Proof Systems Proof- complexity of Solvers

General resolution      The rule is form of modus ponens. Proof is a directed acyclic graph (DAG).

$$\frac{(x_1 \vee \dots \vee x_n) \quad (\neg x_n \vee y_1 \dots \vee y_m)}{(x_1 \vee \dots \vee x_{n-1} \vee y_1 \dots \vee y_m)}$$

Merge resolution      Biased towards deriving clauses that share literals.  
(In the clauses below,  $\alpha$  is a disjunction of literals.)

$$\frac{(\alpha \vee \dots \vee x_n) \quad (\neg x_n \vee \dots \alpha)}{(\alpha \vee \dots \vee x_{n-1})}$$

Unit resolution      One clause must be unit. Proof is a DAG.

$$\frac{(x_n) \quad (\neg x_n \vee y_1 \dots y_m)}{(y_1 \vee \dots \vee y_m)}$$

Tree resolution      Same rules as general resolution. Proof is a tree. Not allowed to reuse lemmas unlike DAG proofs.

# Resolution and Merges

## Mergeability and CDCL SAT Solvers

$$\frac{(\alpha \vee x_n) \ (\neg x_n \vee \beta)}{(\alpha \vee \beta)}$$

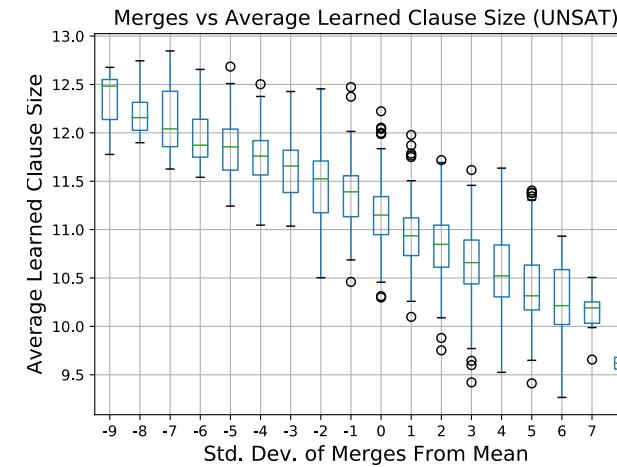
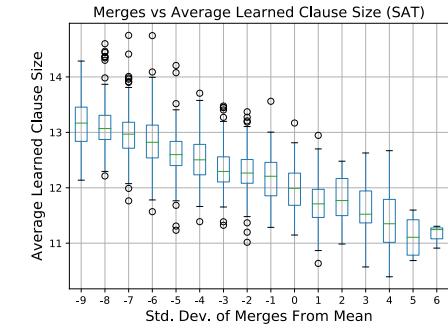
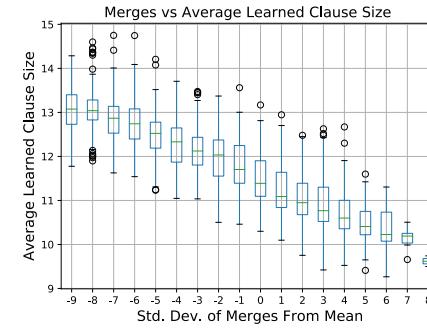
$$\frac{(\gamma \vee y_n) \ (\neg y_n \vee \delta)}{(\gamma \vee \delta)}$$

...

$$\frac{\begin{array}{c} (x_i \vee x_j) \ (\neg x_j \vee x_i) \quad (\neg x_i \vee x_k) \ (\neg x_k \vee \neg x_i) \\ \hline (x_i) \end{array}}{(\neg x_i)}$$
$$\frac{(x_i) \ (\neg x_i)}{\perp}$$

# MERGEABILITY

## NUMBER OF MERGES VS. LEARNT CLAUSE SIZE

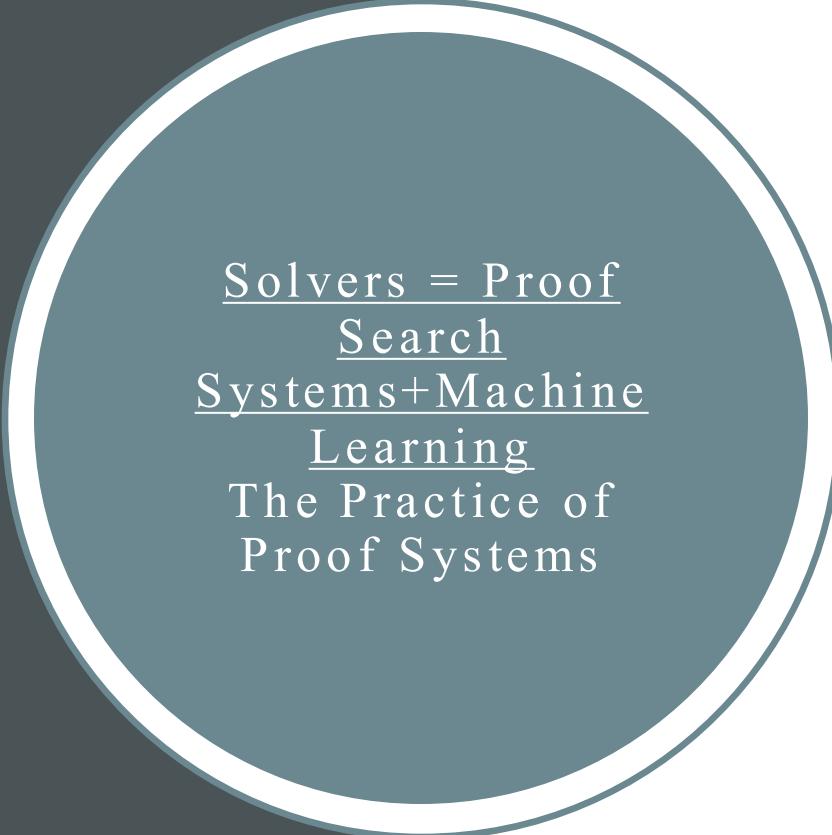




## PART III



Solvers = Proof Systems +  
Machine Learning

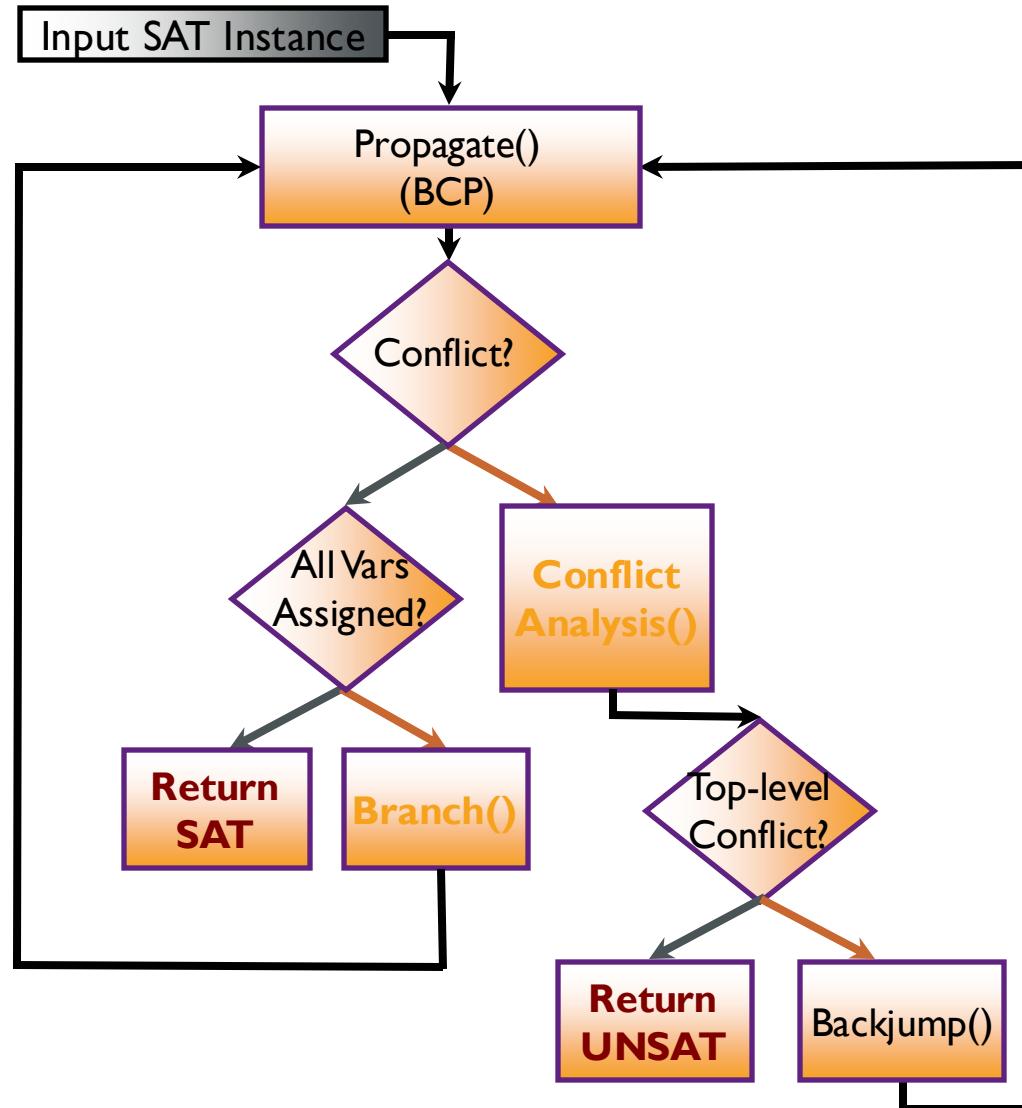


# Solvers = Proof Search Systems+Machine Learning The Practice of Proof Systems

- “Solvers = Proof Search Systems” is a well-known abstraction of solvers and provers. This is a powerful way to design solvers, although non-symbolic DNN-based SAT solvers do exist (e.g., NeuroSAT)
- How can we leverage this abstraction to design better SAT/SMT solvers?
- One way is to strengthen the reasoning engine inside a solver, e.g., CDCL(T) or programmatic SAT
- Another is to design optimization heuristics inside solvers with machine learning (ML) methods. This led to the design of award-winning MapleSAT (and variants). 5 different ML-based heuristics (branching, splitting, restarts, initialization, and algorithm selection). Our work has been widely adopted, adapted, extended [HVC15, SAT16, AAAI16, SAT17, IJCAI18, CP20, ICML20]

# Conflict-Driven Clause-Learning SAT Solver

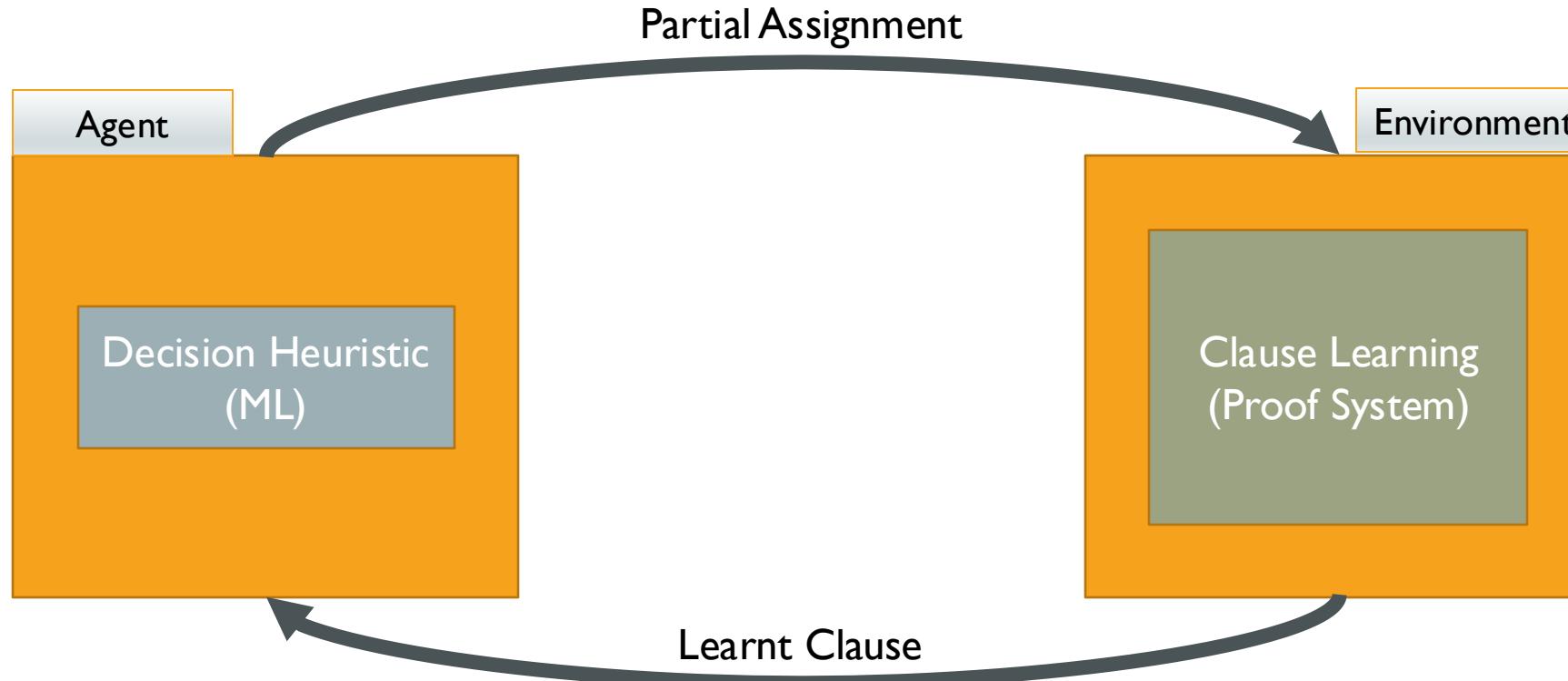
## Solver = Proof Search System + ML



GRASP Solver: [MS96]  
ZChaff Solver: [MMZZM01]

# CDCL with Deductive Feedback Loop

## Reinforcement Learning





## PART IV



Understanding CDCL  
Solver Configurations  
Restarts



## UNDERSTANDING THE POWER OF RESTARTS

- Big open question – We know that CDCL + restarts is polynomially equivalent to Res. Does this equivalence also hold for CDCL w/o restarts?
- Theorem 1: The first CDCL configuration (with restarts) below is exponentially faster than the second CDCL configuration (w/o restarts) on a class of satisfiable instances we refer to as Ladder formulas
  - CDCL + backtracking + non-deterministic variable selection + randomized value selection + restarts
  - CDCL + backtracking + non-deterministic variable selection + randomized value selection - restarts
- Theorem 2: The first CDCL configuration (with restarts) below is exponentially faster than the second CDCL configuration (w/o restarts) on a class of unsatisfiable instances we refer to as pitfall formulas
  - CDCL + backjumping + VSIDS variable selection + phase-saving value selection + restarts
  - CDCL + backjumping + VSIDS variable selection + phase-saving value selection - restarts



## PART V



From Practice to Theory



Using Solvers for Solving  
Complexity Questions



## FROM PRACTICE TO THEORY

- Solving concrete circuit complexity questions using solver
  - For example, minimum number of multiplications needed for the  $3 \times 3$  Boolean matrix multiplication problem (Ryan Williams posted this as a polymath problem.)
- Study of novel proof systems inspired by practice (e.g., DRAT or interference-based proof systems)
- Using solvers to test conjectured classes of instances as lower bounds for corresponding proof systems
- Solving combinatorics problems via SAT (e.g., find Ramsey(5))



## PART V



## Open Problems

## OPEN PROBLEMS (THEORY AND PRACTICE)

- Parameterized upper bound for CDCL – explaining the power of CDCL solvers via parameters
- Crypto hardness – why are crypto problems (e.g., SHA-1 collision/inversion, breaking DES) hard for CDCL SAT solvers? Ideally, construct instances that are “morally” like crypto, and show they are hard for Res?
- Understanding solver configurations – is CDCL w/o restarts polynomially equivalent to Res? Do restart help primarily with proof search and not proof size? How about other configurations such as CDCL with TUIP vs. CDCL without?
- Tight upper bound on BCP algorithm – not aware of any work on complexity analysis of 2-watched scheme
- ML-based solver heuristics (combining deduction with ML) – can we do better than TUIP?
- Feedback on encodings – can we design a method to give feedback to the user about how their encodings can be improved?
- Faster solvers for multiplication – multiplication circuits can be hard for SAT. How can we do better?



## PART VI



Workshop Talks

# SUMMARY OF SCHEDULED WORKSHOP TALKS

Date	Theme	Speakers
Feb 10 <sup>th</sup>	Theoretical Foundation of Solvers: Context, Directions and Open Problems	Vijay Ganesh, David Mitchell, Laurent Simon
Feb 17 <sup>th</sup>	Theory and Practice of Structure of SAT Instances	Stefan Szeider, Jordi Levy, Ralf Rothenberger
Feb 24 <sup>th</sup>	Theory and Practice of Encodings – (encoding showcase)	Oliver Kullmann, Ciaran McCreesh, Maria Bonet
Mar 3 <sup>rd</sup>	Proof Complexity Toolbox	Robert Robere, Susanna de Rezende Joanna Ochremiak,..
Mar 10 <sup>th</sup>	Recent Advances in Proof Complexity of Solvers	Marc Vinyals, Noah Fleming, Ian Li,...
Mar 17 <sup>th</sup>	Non-CDCL Solvers	Marijn Heule, Jakob Nordstrom, Zhiwei Zhang
Mar 24 <sup>th</sup>	Theorem Proving Strategies and Decision Procedures for Satisfiability	Maria Paola Bonacina, Daniel Selsam, ...
Mar 31 <sup>st</sup>	Richard M. Karp Public Lecture	TBD
Apr 7 <sup>th</sup>	Machine Learning for Solvers	TBD

# REFERENCES

- 
- [DPLL62] Davis, M., Putnam, H., Logemann, G. and Loveland, D. A Machine Program for Theorem Proving. CACM 1962
- 
- [MS96] Marques-Silva, J.P. and Sakallah, K.A. GRASP: A New Search Algorithm for Satisfiability. ICCAD 1996
- 
- [MMZZM01] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L. and Malik, S. Chaff: Engineering an efficient SAT solver. DAC 2001
- 
- [BKS03] Beame, P., Kautz, H., Sabharwal, A. Understanding the Power of Clause Learning. IJCAI 2003
- 
- [WGS03] Williams, R., Gomes, C.P. and Selman, B. Backdoors to typical case complexity. IJCAI 2003
- 
- [BSG09] Dilkina, B., Gomes, C.P. and Sabharwal, A. Backdoors in the context of learning. SAT 2009
- 
- [KSM11] Katebi, H., Sakallah, K.A. and Marques-Silva, J.P. Empirical study of the anatomy of modern SAT solvers. SAT 2011
- 
- [AFT11] Atserias, A., Fichte, J., and Thurley, M. Clause Learning Algorithms with Many Restarts and Bounded-width Resolution. JAIR 2011
- 
- [PDI11] Pipatsrisawat, K., and Darwiche, A. On the Power of Clause-learning SAT Solvers as Resolution Engines. AI 2011
- 
- [NGFAS14] Newsham, Z., Ganesh, V., Audemard, G. and Simon, L. Impact of community structure on SAT solver performance. SAT 2014
- 
- [LGZC15] Liang, J.H., Ganesh, V., Zulkoski, E., and Czarnecki, K. Understanding VSIDS branching heuristics in SAT solvers. HVC 2015.
- 
- [LGP16] Liang, J.H., Ganesh, V., Poupart, P., and Czarnecki, K. Learning Rate Based Branching Heuristic for SAT Solvers. SAT 2016
- 
- [LGPC+16] Liang, J.H., Ganesh, V., Poupart, P., and Czarnecki, K. Conflict-history Based Branching Heuristic for SAT Solvers. AAAI 2016
- 
- [RKG18] Robere, R., Kolkolova, A., Ganesh, V. Proof Complexity of SMT Solvers. CAV 2018
- 
- [SPG20] Scott, J., Panju, M., and Ganesh, V. Logic Guided Machine Learning AAAI 2020
- 
- [BBCGV13] Babka et al. Complexity Issues related to Propagation Completeness. JAIR 2013