From Deterministic Automata to Algebraic Decision Diagrams in Boolean Reasoning

Moshe Y. Vardi
Rice University

Werner Kuich

Congratulations to Werner for his 80th birthday!

Many Thanks:

- For foundational work in automata theory, and
- For early recognition of the importance of *quantitative models*!

SAT Solving

SAT: Is a given Boolean formula (typically, in CNF), satisfiable?

David-Putnam-Logemann-Loveland, 1959-62:

- Backtracking search
- Unit propagation
- Pure-literal elimination

Modern CDCL SAT Solvers:

- Backjumping search
- Fast unit propagation
- Fast splitting rules
- Conflict-driven clause learning
- Restarts

‘

The SAT Revolution

An Astonishing Technical Development

- $\sim$1960: First algorithms for SAT
- S.A. Cook, 1971: *NP-completeness* – SAT is the hardest problem in NP.
  - *Common View*: SAT is intractable!
- Late 1990s: Heuristic explosion!
- Mid 2000s: SAT solvers can handle problems with 1Ms variables!
- Late 2000s: *"NP-easy"*– SAT solvers as generic problem solvers

Raining on The Party

Critical Observation: There is basically only one algorithm that scales well to large industrial problems - *CDCL*

- Based on weakest proof system – *resolution*

Provocative Note: "Monoculture" (the practice of cultivating a single crop over a wide area) is a risky practice!

This talk: A sketch of another technical approach

Satisfiability and Formal Languages

- $Prop = \{p_1, \ldots, p_n\}$
- Truth assignment: $\alpha : Prop \rightarrow \{0, 1\}$
- Satisfying assignments: $models(\varphi) = \{\alpha \ : \ \alpha \models \varphi\} \subseteq \{0, 1\}^n$

Crux: $models(\varphi)$ is a *finite*, so *regular language*.

An Automata-Theoretic Approach to Satisfiability

Basic Approach:

1. Construct DFA A_φ such that $L(A_\varphi) = models(\varphi)$
2. Check that $L(A_\varphi) \neq \emptyset$

In more details:

1. $\varphi = c_1 \wedge \ldots \wedge c_k$ (CNF)
2. Construct DFA A_i for clause $c_i = \ell_{i_1} \vee \ell_{i_2} \vee \ldots$
3. $A_\varphi = A_1 \times \ldots \times A_k$
4. $L(A_\varphi) \neq \emptyset$ iff there is a path from an initial state to an accepting state.

Problem: $|A_\varphi| = O(n^k)$

Don't Cares

Observation:

- Typically: $|c_i| \ll n$
- So why $|A_i| = O(n)$?
- After all, most propositions are "don't cares" in c_i?

Desidetra:

- Automaton model that can "skip" don't-care letters, so $|A_i| = O(|c_i|)$

Solution: Binary Decision Diagrams!

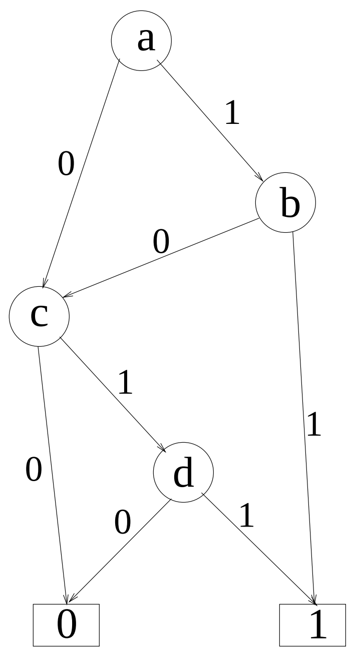


Figure 1: BDD for $(a \wedge b) \vee (c \wedge d)$

Reduced Ordered Binary Decision Diagrams

BDDs: efficient way to manipulate Boolean functions

- directed acyclic graph ("folded decision tree")
- internal nodes correspond to Boolean variables
- all paths lead to one of the two terminal vertices labeled by 0 and 1

Properties:

- canonical representation for a given variable ordering
- easy equivalence check
- polynomial Boolean operations

BDD and SAT Solving

- **Good News**: $|B_i| = O(|c_i||$
- **Bad News**: $|B_\varphi| = O(\Pi_{i=1}^k |c_i|)$

Bottom Line: "Monolithic" approach not viable.

Complexity Theory and Proofs

Fundamental Questions:

- $P = NP$? $NP = co - NP$?

Cook-Reckhow's Theorem, 1979: $NP = co - NP$ if and only if there exists a polynomially bounded propositional proof system.

Proof-Complexity Theory: Study of quantitative properties of propositional proof systems.

- Haken, 1985 – Pigeonhole Principle requires exponentially long resolution proofs.
- Cook-Reckhow, 1985 – Pigeonhole Principle does have polynomially long extended Frege proofs.

Proof Complexity and Satisfiability Solving

- Galil, 1977: A DPLL refutation can be transformed into a tree resolution refutation of the same size.
- Beame-Kautz-Sabharwal, 2003:
  - A CDCL refutation with clause learning and restarts can be transformed into a resolution refutation of the same size, and vice versa.
  - **Corollary**: CDCL is exponentially more powerful than DPLL.

Bottom line: Study of proof complexity is *practically* important to satisfiability solving.

Constraint Satisfaction Problem (CSP)

Input: $P = (V, D, C)$:

- A finite set V of *variables*
- A finite set D of *values*
- A finite set C of *constraints* restricting the values that tuples of variables can take – **Constraint**: $(\mathbf{x}, R)$
  - $\mathbf{x}$: a tuple of variables over V
  - R: a relation of arity $|\mathbf{x}|$

Solution: $h : V \to D$ such that $h(\mathbf{x}) \in R$: for all $(\mathbf{x}, R) \in C$

Decision Problem: Does (V, D, C) have a solution? I.e., is there an assignment of values to the variables such that all constraints are satisfied?

3-Colorability

3-COLOR: Given an undirected graph $A = (V, E)$, is it 3-colorable?

- The variables are the nodes in V.
- The values are the elements in $\{\mathbf{R}, \mathbf{G}, \mathbf{B}\}$.
- The constraints are $\{(\langle u, v\rangle, \rho) \quad : \quad (u, v) \in E\}$, where $\rho = \{(R, G), (R, B), (G, R), (G, B), (B, R), (B, G)\}$.

Constraint Satisfaction

Applications:

- belief maintenance
- machine vision
- natural language processing
- planning and scheduling
- temporal reasoning
- type reconstruction
- bioinformatics
- $\cdots$

Homomorphisms

Homomorphism: Let $\mathbf{A} = (A, R_1^{\mathbf{A}}, \ldots, R_m^{\mathbf{A}})$ and $\mathbf{B} = (B, R_1^{\mathbf{B}}, \ldots, R_m^{\mathbf{B}})$ be two relational structures over same vocabulary.

$h : A \to B$ is a *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ if for $i = 1, \ldots, m$ and every tuple $(a_1, \ldots, a_n) \in A^n$,

$$R_i^{\mathbf{A}}(a_1, \ldots, a_n) \implies R_i^{\mathbf{B}}(h(a_1), \ldots, h(a_n)).$$

The Homomorphism Problem: Given relational structures $\mathbf{A}$ and $\mathbf{B}$, is there a homomorphism $h : \mathbf{A} \to \mathbf{B}$?

Example: An undirected graph $\mathbf{A} = (V, E)$ is 3-colorable

$$\Longleftrightarrow$$

there is a homomorphism $h : \mathbf{A} \to K_3$, where K_3 is the *3-clique*.

Homomorphism Problems

Examples:

- k-Clique: $K_k \stackrel{h}{\to} (V, E)$?
- Hamiltonian Cycle: $(V, C_{|V|}, \neq) \stackrel{h}{\to} (V, E, \neq)$?
- Subgraph Isomorphism: $(V, E, \overline{E}) \stackrel{h}{\to} (V', E', \overline{E'})$?
- s-t Connectivity: $(V, E, \{\langle s, t\rangle\}) \stackrel{h}{\not\to} (\{0, 1\}, =, \neq)$?

Feder&V., STOC'93: CSP=Homomorphism Problem

- Trivial reductions in both directions

Introduction to Database Theory

Basic Concepts:

- *Relation Scheme*: a set of attributes
- *Tuple*: mapping from relation scheme to data values
- *Tuple Projection*: if t is a tuple on P, and $Q \subseteq P$, then $t[Q]$ is the restriction of t to Q.
- *Relation*: a set of tuples over a relation scheme
- *Relational Projection*: if R is a relation on P, and $Q \subseteq P$, then $R[Q]$ is the relation $\{t[Q] : t \in R\}$.
- *Join*: Let R_i be a relation over relation scheme S_i. Then $\bowtie_i R_i$ is a relation over the relation scheme $\cup_i S_i$ defined by $\bowtie_i R_i = \{t : t[S_i] \in R_i\}$.

CSP Proofs

[Atserias, Kolaitis, V., 2004]: **CSP Refutation**: A CSP proof that $P = (V, D, C)$ is unsatisfiable is a finite sequence of constraints $(\mathbf{x}, R)$ each of which is of one of the following forms:

1. *Axiom*: $(\mathbf{x}, R) \in C$
2. *Join*: $(\mathbf{x} \cup \mathbf{y}, R \bowtie S)$, where $(\mathbf{x}, R)$ and $(\mathbf{y}, S)$ are previous constraints,
3. *Projection*: $(\mathbf{x} - \{x\}, R[\mathbf{x} - \{x\}]$, where $(\mathbf{x}, R)$ is a previous constraint,
4. *Weakening*: $(\mathbf{x}, S)$, where $(\mathbf{x}, R)$ is a previous constraint and $R \subseteq S$,

where the last constraint has an *empty* relation.

Theorem: (V, D, C) has no solution iff it has a CSP refutation.

CSP Refutations and Constraint Propagation

Constraint propagation: technique for preprocessing and solving constraints

What is constraint propagation?

- *Join*: The constraints $(\mathbf{x}, R)$ and $(\mathbf{y}, S)$ are combined to yield $(\mathbf{x} \cup \mathbf{y}, R \bowtie S)$.
- *Projection*: The constraint $(\mathbf{x} \cup \mathbf{y}, R \bowtie S)$ is projected back on $\mathbf{x}$ to yield $(\mathbf{x}, R \bowtie S)[\mathbf{x}])$.

Dechter-van Beek, 1997: $resolve_x(c, d)$ is

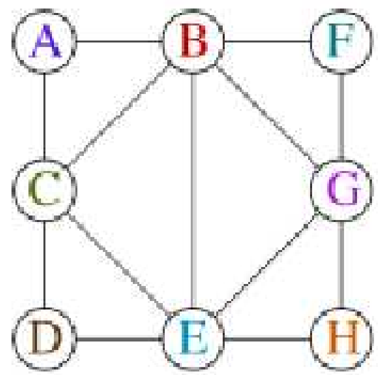
$$models(c) \bowtie models(d)[var(c \cup d) - \{x\}]$$

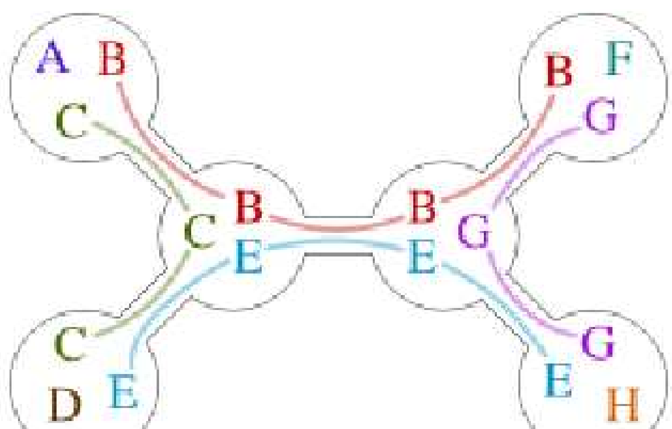
Treewidth

Definition: A *tree decomposition* of a structure $\mathbf{A} = (A, R_1, \ldots, R_m)$ is a labeled tree T such that

- Each label is a non-empty subset of A;
- For every R_i and every $(a_1, \ldots, a_n) \in R_i$, there is a node whose label contains $\{a_1, \ldots, a_n\}$.
- For every $a \in A$, the nodes whose label contain a form a subtree.

$$\mathrm{tw}(\mathbf{A}) \;=\; \min_T\{\max\{\text{label size in } T\}\} - 1$$


A B F
C G
D E H


A B C
B F G
B C E
B G E
C D E
G E H

Treewidth and Bounded-Width Proofs

Treewidth of CSP instance P: View tuples of constraints of P as tuples of a relational structure.

Width of Refutation: Maximal arity of constraints in refutation.

Atserias, Dalmau, Kolaitis and V., 2002-4: If P has treewidth at most k, then P is unsatisfiable iff P has width-k refutation.

Boolean Constraint Representation

Representing a constraint $(\mathbf{x}, R)$ over Boolean domain:

- Relations: set of tuples
- Clauses: $\bigvee_i \pm x_i$
- Linear inequalities: $\sum_i a_i x_i \leq a_0$

Desideratum: polynomial closure under join, projection, and weakening.

- Clauses are closed under resolution, but not under join.

Another possibility: representation by *BDDs* – reduced, ordered, binary decision diagrams

BDDs vs. Resolution

AKV'04: BDDs polynomially simulate resolution.

Proof:

- Resolution=join+projection
- BDDs support polynomial join and projection
- A clausal constraint can be expressed by a linear-sized BDD – exclude a single truth assignment.

AKV'04: Pigeonhole Principle has polynomial-size BDD refutations.

Conclusion: BDD refutations are exponentially more powerful than resolution.

More Power to BDDs

Gaussian calculus:

- Constraints: $x + y + z = 0/1$
- Proofs: Gaussian elimination

AKV'04: BDDs polynomially simulate the Gaussian calculus.

Cutting Planes:

- Constraints: $a_1x_1 + a_2x_2 + a_3x_3 \leq a_0$ (unary coefficients)
- Proofs: addition, scalar multiplication, integer division

AKV'04: BDDs polynomially simulate Cutting Planes.

Discussion

So far: Proof complexity theory for CSP

- A general framework of CSP proofs
- Study of bounded-width CSP proofs
- Study of BDD proofs

Big Question: Is this useful for SAT solving?

BDD Proofs for SAT Solving

A Possible Approach [Pan-V., 2004]:

- Perform constraint propagation, using joins and projections, exhaustively – empty constraint implies unsatisfiability.

Question: At what order to apply joins and projections?

Naive Approach: Leverage the connection between bounded treewidth and bounded-width refutations

- Obtain an optimal tree decomposition.
- Use decomposition to guide deduction.

Problem: Finding an optimal tree decomposition is NP-hard.

A Practical Approach: Early Quantification

Early Quantification: A basic technique in BDD-based model checking.

- A SAT instance is a formula of the form

$$(\exists v_1)(\exists v_2)\ldots(\exists v_n)(c_1 \wedge c_2 \wedge \ldots \wedge c_m)$$

- ‘ If v_j does not appear in the clauses $c_{k+1}, \ldots, c_m$, then we can rewrite the formula into an equivalent one:

$$(\exists v_1)\ldots(\exists v_{j-1})(\exists v_{j+1})\ldots(\exists v_n)((\exists v_j)(c_1 \wedge \ldots \wedge c_k) \\ \wedge c_{k+1} \wedge \ldots \wedge c_m)$$

Suggested Approach [Pan&V., 2004]: Join lazily, project eagerly.

Clause Reordering

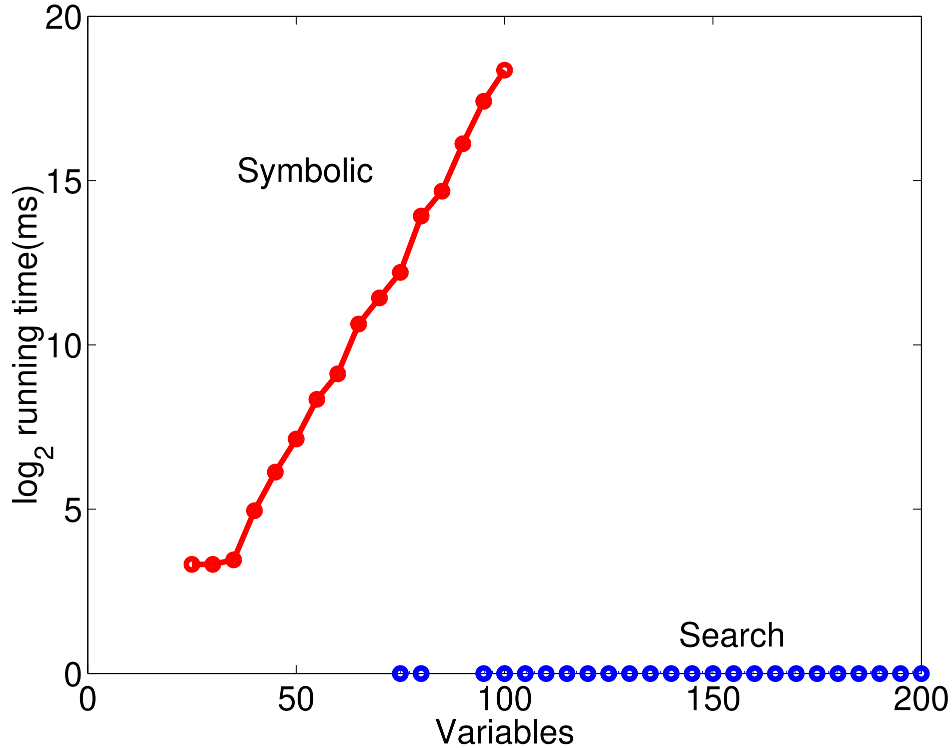
Goal: Reorder clauses to maximize early quantification, i.e., minimize size of intermediate constraints.

Difficulty: Finding optimal clause order is NP-hard– related to finding optimal tree decomposition.

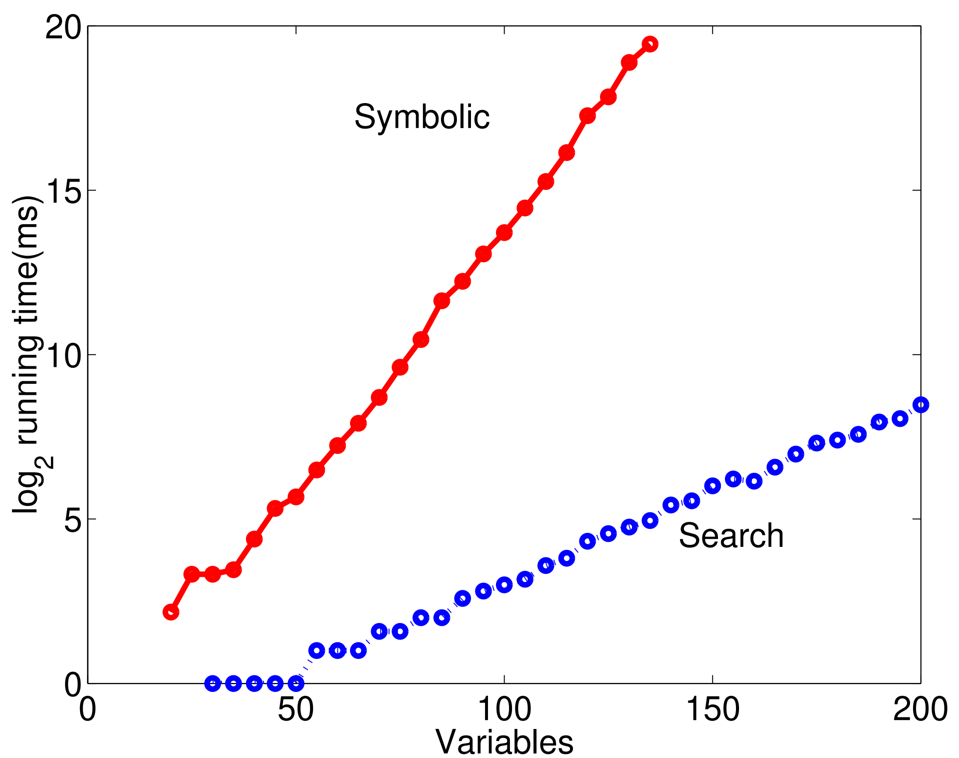
BDDSAT [Pan-V., 2004]:

- Borrow heuristics used to finding good tree decompositions.
- Borrow heuristics used in CSP
- Borrow heuristics used in symbolic model checking.

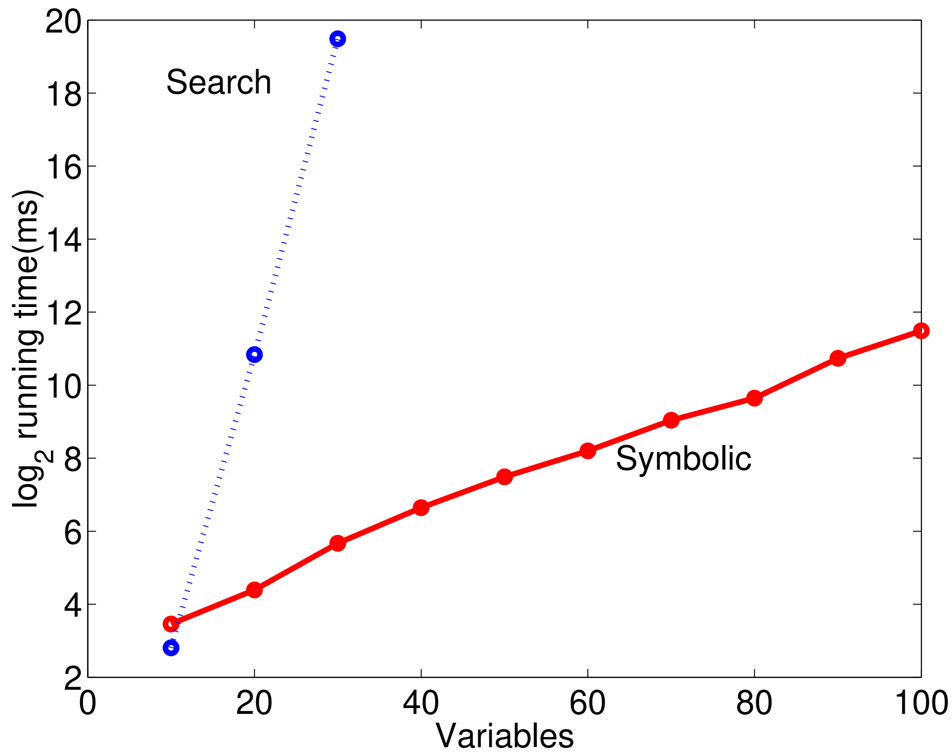
BDDSAT vs. ZChaff:: Random 3-CNF, density=1.5



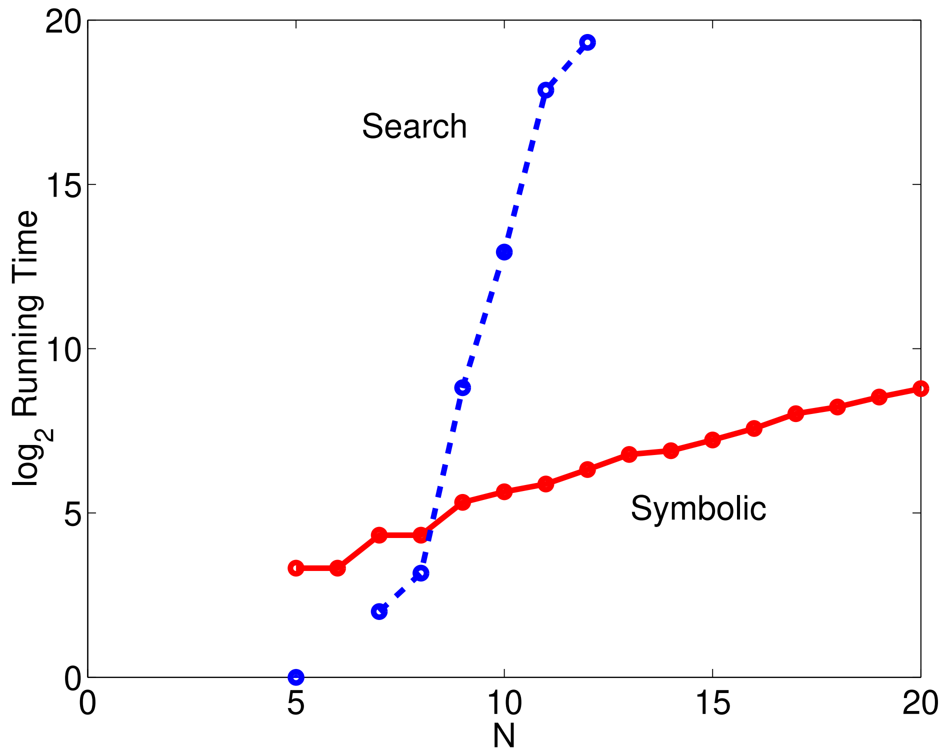
BDDSAT vs. ZChaff:: Random 3-CNF, density=6



BDDSAT vs. ZChaff:: Random Biconditionals



BDDSAT vs. ZChaff:: Mutilated Checkerboard



Symbolic Approach vs. Search

Summary as of 2005:

- Symbolic approach scales better on some problems
- Incomparable in general

Note: Search has the benefit of 40 years of engineering!

My conclusion in 2005: Symbolic approach merits further study – need to understand areas of effectiveness

Bryant-Heule, 2021: "A powerful, BDD-based SAT solver to generate proofs of unsatisfiability"

Quantitative Boolean Reasoning

- **Model counting (#SAT)**: computing number of satisfying assignments of Boolean formula
- **Complexity**: #P-complete (Valiant, 1979)
- **Numerous applications**: especially in probabilistic reasoning

Weighted Model Counting: Assignments are *weighted*, e.g., literal weighting:

- Each literal has a weight.
- Weight of assignment = product of literal weights
- **Task**: Compute sum of weights of satisfying assignments

Reduced Ordered Algebraic Decision Diagrams

ADDs: Efficient way to manipulate *pseudo-Boolean* functions

- directed acyclic graph ("folded decision tree")
- internal nodes correspond to Boolean variables
- terminal nodes labeled with real numbers

Properties:

- canonical representation for a given variable ordering
- polynomial plus/times operations

Weighted MC with ADDs

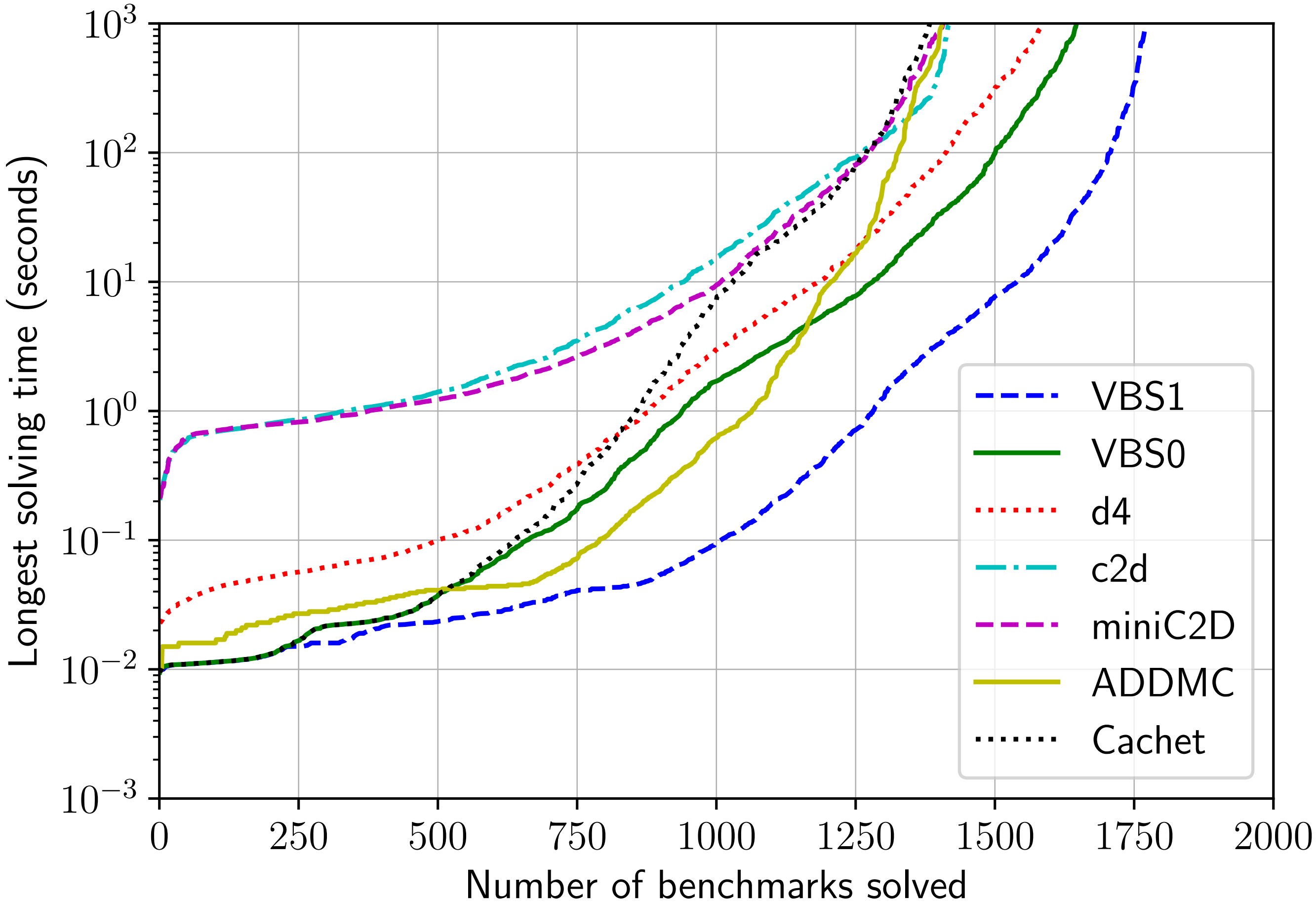
Basic Idea:

- Construct BDD B_φ for input formula φ.
- Combine with literal weights to construct ADD $A_\varphi : 2^{Prop(\varphi)} \rightarrow \mathbb{R}$
- Project all propositions using $\Sigma_p(A) = A[p \mapsto 0] + A[p \mapsto 1]$.

Problem: B_φ blows up.

ADDMC: Early quantification as in BDDSAT [Dudek, Phan, and V., 2020] – tied for 1st place of weighted track in 2020 Model Counting Competition.

ADDMC



Longest solving time (seconds)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
0
250
500
750
1000
1250
1500
1750
2000
Number of benchmarks solved
VBS1
VBS0
d4
c2d
miniC2D
ADDMC
Cachet

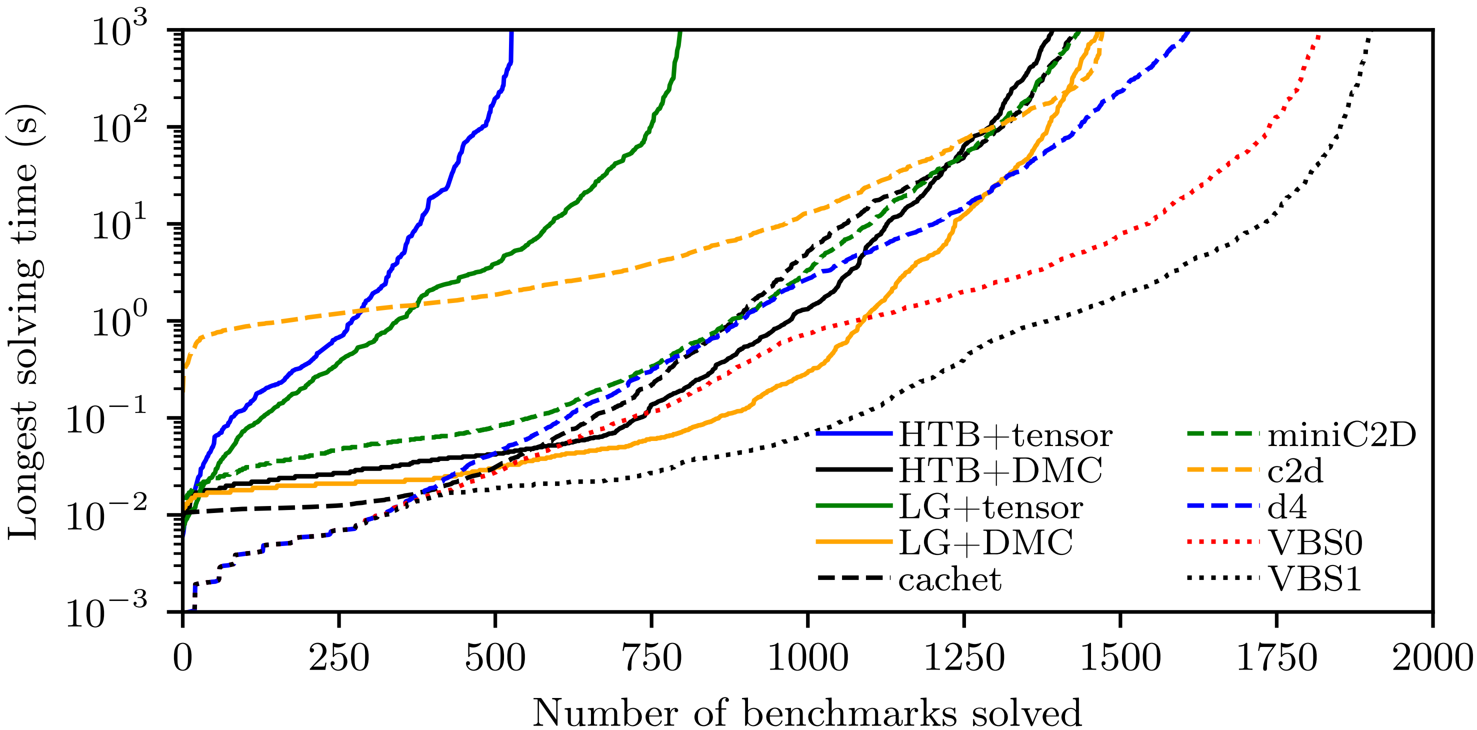
Back to Tree Decompositions

Key Observation: Computing optimal tree decompositions is NP-hard, but there has been huge progress since 2005 in computing “good” tree decomspoitions, including *anytime* solvers.

DPMC: Early quantification, but based on tree decompositions rather than CSP heuristics [Dudek, Phan, and V., 2020].

- **Empirical observation**: DMPC beats ADDMC decisively.

DPMC



Longest solving time (s)
Number of benchmarks solved
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
0
250
500
750
1000
1250
1500
1750
2000
HTB+tensor
HTB+DMC
LG+tensor
LG+DMC
cachet
miniC2D
c2d
d4
VBS0
VBS1

Discussion

My points:

- BDDs and ADDs are variants of DFAs.
- BDDs/ADDs provide a viable approach to Boolean reasoning.
- Industrial tools are often hybrid engines - *algorithmic protfolio*!
- The SAT community is ignoring this approach!

Questions:

- Are the competitions discouraging alternative approaches?
- How can we combine search and symbolic techniques?