

# Symbolic Techniques in Propositional Satisfiability Solving

Moshe Y. Vardi  
Rice University

# The SAT Revolution

## An Astonishing Technical Development

- ~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: Technical breakthroughs – *clause learning and memory locality*
- Mid 2000s: SAT solvers can handle problems with 1M variables!
- Late 2000s: “*NP-easy*”– SAT solvers as generic problem solvers

## 2009 CAV Award

On July 2, 2009, the Computer-Aided Verification (CAV) award was presented to the following individuals:

- Conor F. Madigan
- Sharad Malik, Princeton University
- Joao P. Marques-Silva, University College Dublin
- Matthew W. Moskewicz, UC Berkeley
- Karem A. Sakallah, University of Michigan
- Lintao Zhang, Microsoft Research
- Ying Zhao

with the citation: “For fundamental contributions to the development of high-performance Boolean satisfiability solvers”

# Raining on The Party

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

- Backtrack search with clause learning (memoing)
- Based on a weak 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

# 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.

*Example:* Haken, 1985 – Pigeonhole Principle requires exponentially long resolution proofs.

*Example:* 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 DPLL refutation with clause learning and restarts can be transformed into a resolution refutation of the same size, and vice versa.
- **Corollary:** DPLL with clause learning and restarts is exponentially more powerful than pure DPLL.

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

# Constraint Satisfaction Problem (CSP)

**Input:**  $(V, D, C)$ :

- 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:**  $h : V \rightarrow D$

- $h(\mathbf{x}) \in R$ : for all  $(t, R) \in C$

**Question:** 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  $\{(\langle u, v \rangle, \rho) : (u, v) \in E\}$ , where  $\rho = \{(R, G), (R, B), (G, R), (G, B), (B, R), (B, G)\}$ .

# Homomorphisms

**Homomorphism:** Let  $\mathbf{A} = (A, R_1^{\mathbf{A}}, \dots, R_m^{\mathbf{A}})$  and  $\mathbf{B} = (B, R_1^{\mathbf{B}}, \dots, R_m^{\mathbf{B}})$  be two relational structures.

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

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

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

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



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

# CSP vs. Homomorphisms

## From CSP to Homomorphism:

Given:  $(V, D, \{C_1, \dots, C_m\})$ , where  $C_i = (t_i, R_i)$ .

Define **A**, **B**:

- $\mathbf{A} = (V, \{t_1\}, \dots, \{t_m\})$
- $\mathbf{B} = (D, R_1, \dots, R_m)$

**Fact:**  $(V, D, C)$  has a solution iff there is homomorphism from **A** to **B**.

# Homomorphisms vs. CSP

## From Homomorphism to CSP:

Given:  $\mathbf{A} = (A, R_1^A, \dots, R_m^A)$ ,  $\mathbf{B} = (B, R_1^B, \dots, R_m^B)$ .

Define  $(V, D, C)$ :

- $V = A$ : elements of  $\mathbf{A}$  are variables.
- $D = B$ : elements of  $\mathbf{B}$  are values.
- $C = \{(t, R_i^B) : t \in R_i^A\}$ : constraints derived from  $\mathbf{A}, \mathbf{B}$ .

**Fact:** There is homomorphism from  $\mathbf{A}$  to  $\mathbf{B}$  iff  $(V, D, C)$  has a solution.

Feder&V., 1993: CSP=Homomorphism Problem

# 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

$$\text{CSP}(\mathbf{B}) = \{\mathbf{A} \mid \exists h : \mathbf{A} \rightarrow \mathbf{B}\}$$

- $\mathbf{A} \in \text{CSP}(\mathbf{B})$ : exhibit homomorphism
- $\mathbf{A} \notin \text{CSP}(\mathbf{B})$ : prove that no such homomorphism

**CSP Proof:** A CSP proof that  $\mathbf{A} \notin \text{CSP}(\mathbf{B})$  is a finite sequence of constraints  $(\mathbf{x}, R)$  each of which is of one of the following forms:

1. *Axiom:*  $(\mathbf{x}, R^{\mathbf{B}})$ , where  $\mathbf{x} \in R^{\mathbf{A}}$ ,
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.

# CSP Proofs: Database Perspective

**Theorem:**  $A$  has a CSP( $B$ ) refutation if and only if  $A \notin \text{CSP}(B)$ . In fact, axioms and joins alone are already enough to refute an unsatisfiable instance.

Bibel'88, Gyssens-Jeavons-Cohen'96: DB vs. CSP

Given:  $(V, D, \{C_1, \dots, C_m\})$ , where  $C_i = (\mathbf{x}_i, R_i)$ .

Assume (wlog): Each  $\mathbf{x}_i$  consists of distinct elements.

## Database Perspective:

- $V$ : attributes
- $D$ : values
- $(\mathbf{x}_i, R_i)$ : relation  $R_i$  over relation scheme  $\mathbf{x}_i$

**Theorem:**  $(V, D, \{C_1, \dots, C_m\})$  has a solution iff  $\bowtie_1^m R_i$  is nonempty.

# 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

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

**Why weakening?** Technical reason.

**Why projection?** To limit size of constraints.

**Focus now:** *Bounded width* refutations (polynomially bounded constraints).

# Existential $k$ -Pebble Games

$A, B$ : structures

- **Spoiler**: *places on or removes* a pebble from an element of  $A$ .
- **Duplicator**: tries to duplicate move on  $B$ .

$A$ :  $a_1, a_2, \dots, a_l$   $l \leq k$

$B$ :  $b_1, b_2, \dots, b_l$

- *Spoiler wins*:  $h(a_i) = b_i, 1 \leq i \leq l$  is **not** a homomorphism.
- *Duplicator wins*: otherwise.

**Easy Fact**: If there is a homomorphism from  $A$  to  $B$ , then the Duplicator wins the existential  $k$ -pebble game on  $A, B$ . *The reverse need not hold.*

# Bounded-Width refutations and Games

Atserias-Kolaitis-V., 2004: The following are equivalent:

1.  $\mathbf{A}$  has a  $\text{CSP}(\mathbf{B})$  refutation of width  $k$ .
2. The Spoiler wins the existential  $k$ -pebble game on  $\mathbf{A}$  and  $\mathbf{B}$ .

To check whether  $\mathbf{A}$  has a  $\text{CSP}(\mathbf{B})$  refutation of width  $k$ , find out who wins the existential  $k$ -pebble game on  $\mathbf{A}$  and  $\mathbf{B}$ .

**Game Configuration Space:**  $(\|\mathbf{A}\| + \|\mathbf{B}\|)^k$

- Fixed  $k$ : polynomial
- Variable  $k$ : exponential

Kolaitis-Panttaja, 2003: Deciding who wins the game for variable  $k$  is *EXPTIME-complete* – harder than homomorphism testing!

**Conjecture:** Deciding if a CNF formula has a refutation of width  $k$  is EXPTIME-complete.

# Treewidth

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

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

The *treewidth*  $\text{tw}(\mathbf{A})$  of  $\mathbf{A}$  is defined by

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

**Note:** Generalizes the *treewidth* of a *graph* – measure “*treeness*”.

# Treewidth and Bounded Width

Dechter-Pearl, 1987, Freuder, 1990: Treewidth vs. induced width in *adaptive consistency*

Dalmau, Kolaitis and V., 2002: If  $\mathbf{A}$  has treewidth less than  $k$ , then the following are equivalent:

1. There is a homomorphism from  $\mathbf{A}$  to  $\mathbf{B}$ .
2. The Duplicator wins the existential  $k$ -pebble game on  $\mathbf{A}$  and  $\mathbf{B}$ .

AKV, 2004: If  $\mathbf{A}$  has treewidth less than  $k$ , then the following are equivalent:

1.  $\mathbf{A}$  has a CSP( $\mathbf{B}$ ) refutation of width  $k$ .
2. There is a homomorphism from  $\mathbf{A}$  to  $\mathbf{B}$ .

# 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 possibilities:** representation by *BDDs* – reduced, ordered, binary decision diagrams

# Example

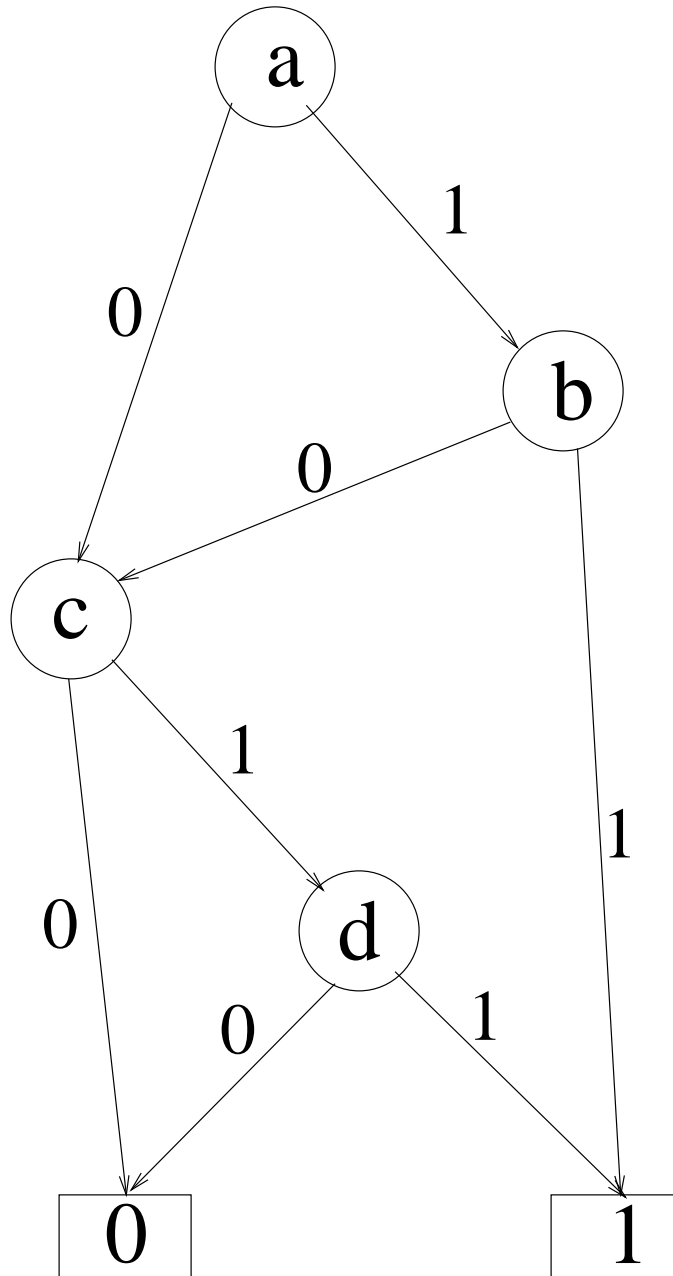


Figure 1: BDD for  $(a \wedge b) \vee (c \wedge d)$

# 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 the values 0 and 1

**Properties:**

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

BDD refutations constitute a propositional proof system (Cook-Reckhow).

# 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 does have polynomially long 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]:

- Apply constraint propagation (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

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

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

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

**Suggested Approach:** Join lazily, projecting variables sequentially.

**Comment:** Related to *bucket elimination*, used in CSP (Dechter, 1997).

# 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.

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

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

# BDDs vs. ZChaff: Round I

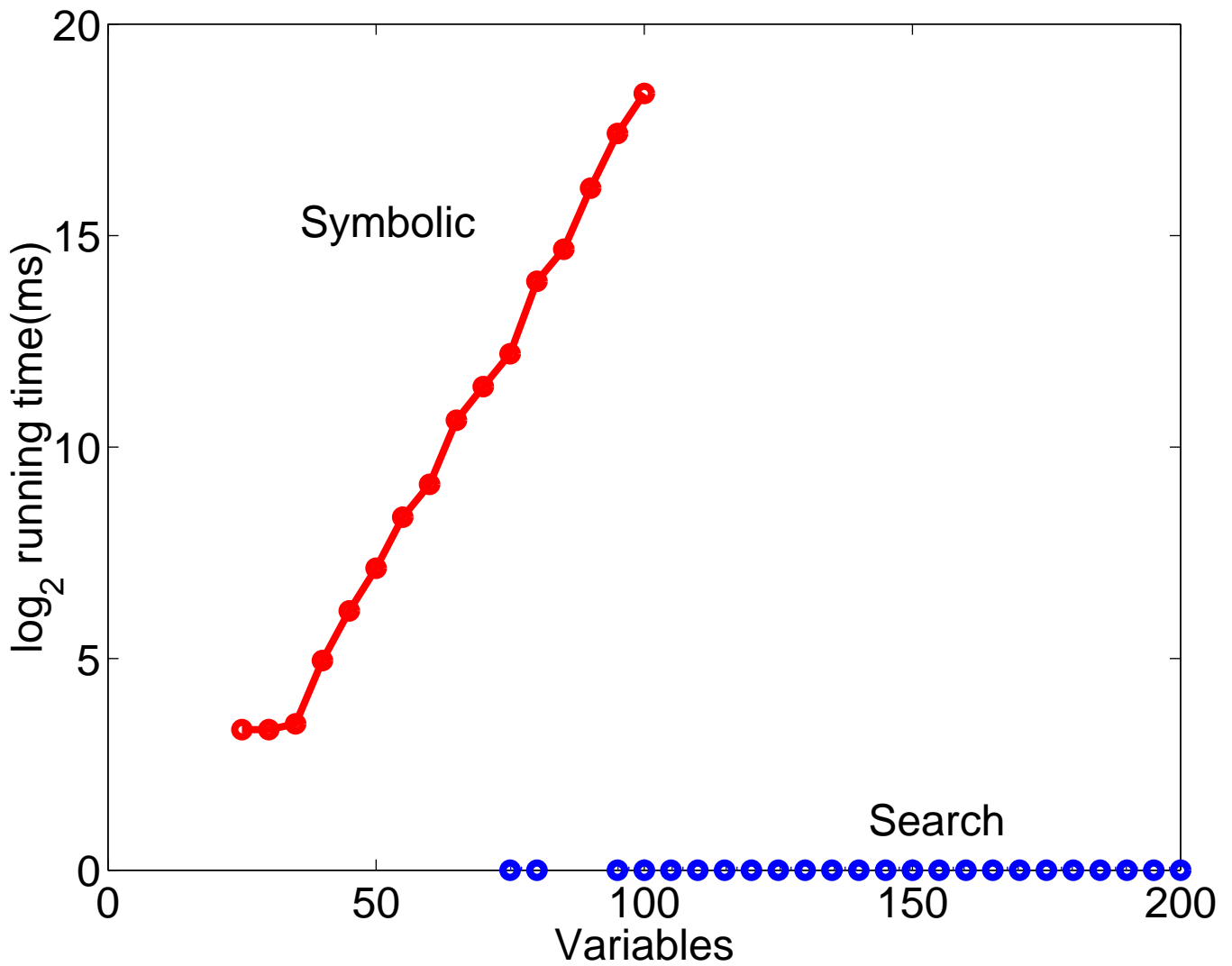


Figure 2: Random 3-CNF, density=1.5

# BDDs vs. ZChaff: Round II

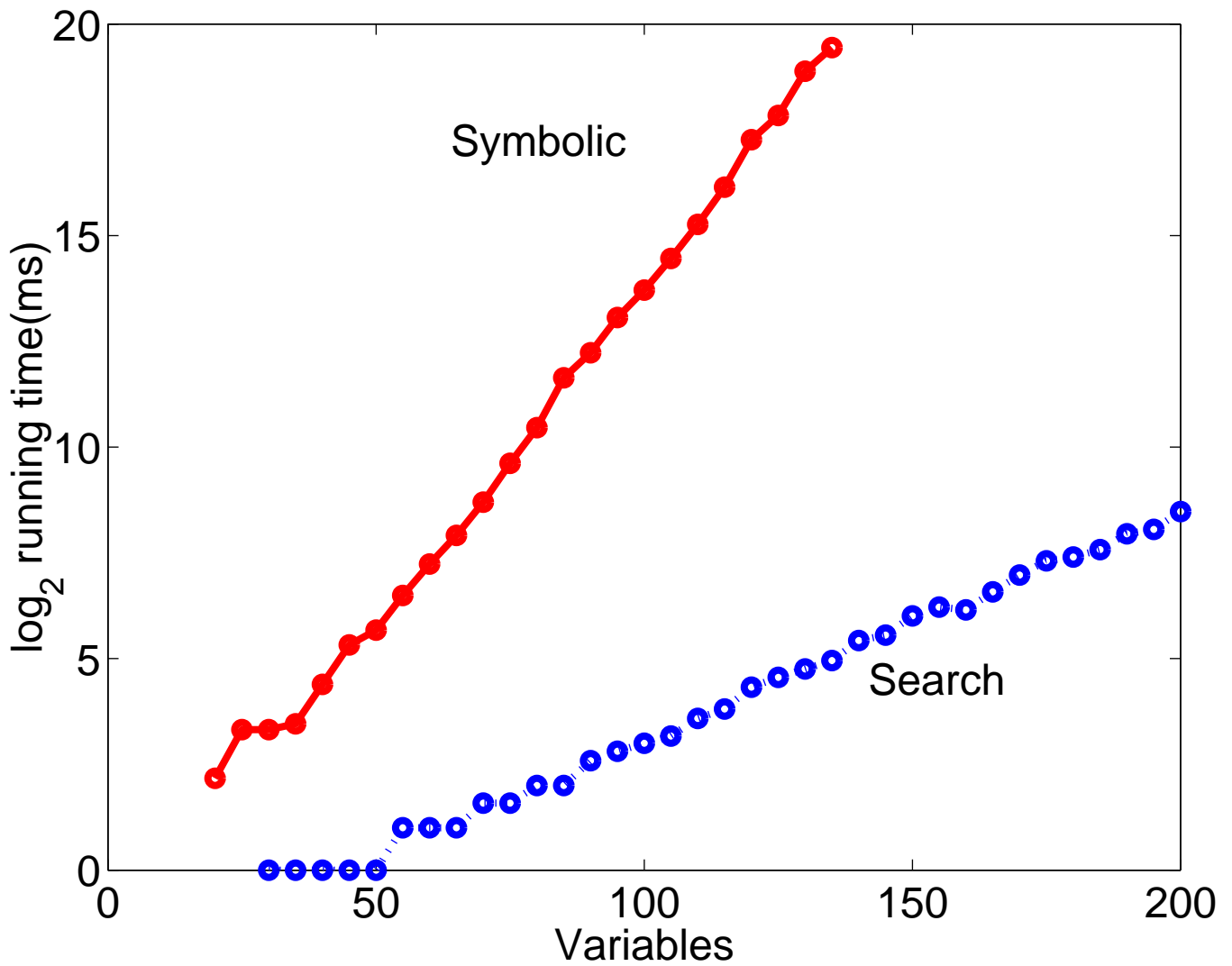


Figure 3: Random 3-CNF, density=6

# BDDs vs. ZChaff: Round III

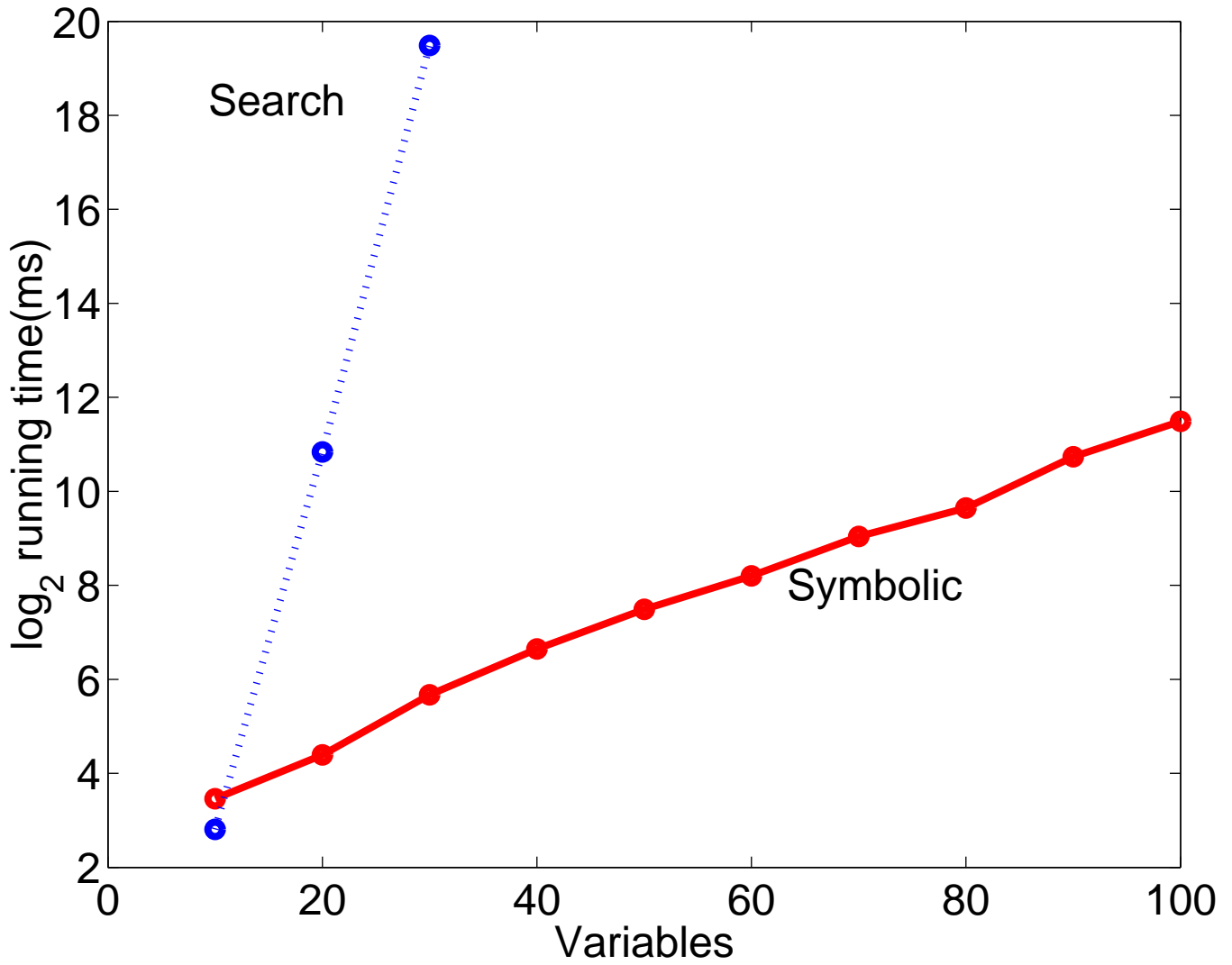


Figure 4: Random Biconditionals

# BDDs vs. ZChaff: Round IV

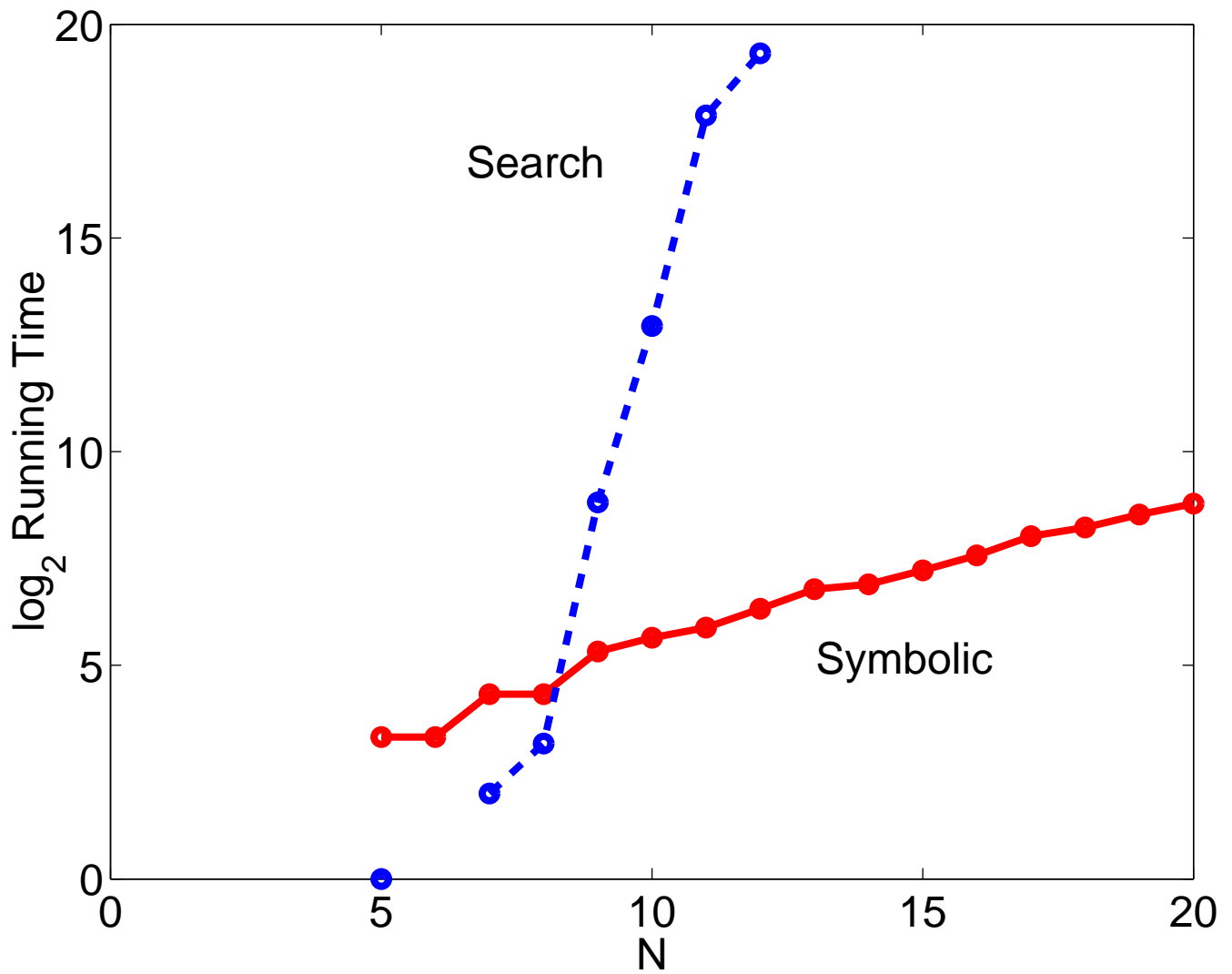


Figure 5: Mutilated Checkerboard

# Symbolic Approach vs. Search

## Summary So Far:

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

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

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

**Related Work:** Symbolic representation of clause sets [Chatalic-Simon, 2001] – not competitive with BDDs.

# Another Arena for Comparison: QBF

**QBF:**  $(\exists v_1)(\forall v_2) \dots (\exists v_n)(c_1 \wedge c_2 \wedge \dots \wedge c_m)$

- Canonical PSPACE problem
- Many problems can be easily reduced to QBF
  - Conformant planning
  - Two-player games
  - Modal Logic
  - Model checking
- Quite harder than SAT in practice

# Symbolic Approach to QBF

**QBDD** [Pan+V., 2004]:

- Eliminate quantified variables inside out
  - $\exists v_i$ : projection
  - $\forall v_i$ : dualized projection
- Join lazily, only when required for quantifier elimination

**Note:** Clause reordering plays minimal role

**Result:** Beautiful algorithm, weak performance

# Back to Clauses

Chatalic-Simon, 2001: Symbolic implementation of [Davis-Putnam, 1960]

- Solve SAT by quantifier elimination
- Represent constraints as clauses
- Implement quantifier elimination by resolution

## Symbolic Implementation:

- Represent a clause as bit vector, with two BDD variables per proposition (10 - positive occurrence, 01 - negative occurrence, 00 - no occurrence, 11 - not allowed).
- Represents sets of sparse bit vectors by ZDDs (Zero-Suppressed Decision Diagrams)
- Implement resolution symbolically

# Symbolic Solving of QBF

**QMRES** [Pan+V., 2004]:

- Solve QBF by quantifier elimination
- Represent constraints as clauses
- Implement quantifier elimination by resolution and projection
- Represent clauses as bit vectors
- Represents sets of clauses by ZDDs
- Implement quantifier elimination symbolically

# Performance Benchmarking

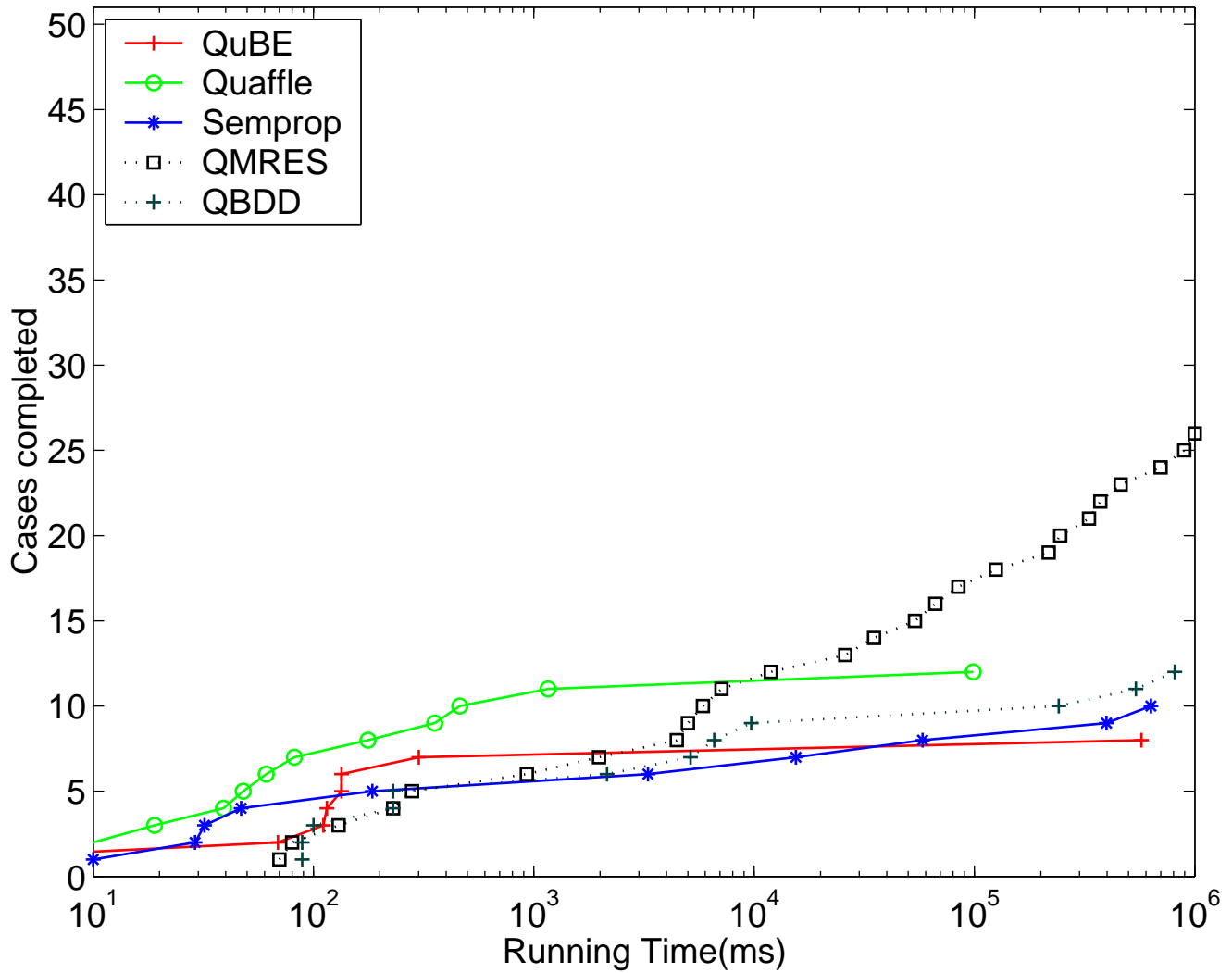


Figure 6: Ayari's Benchmarks

# Performance Benchmarking

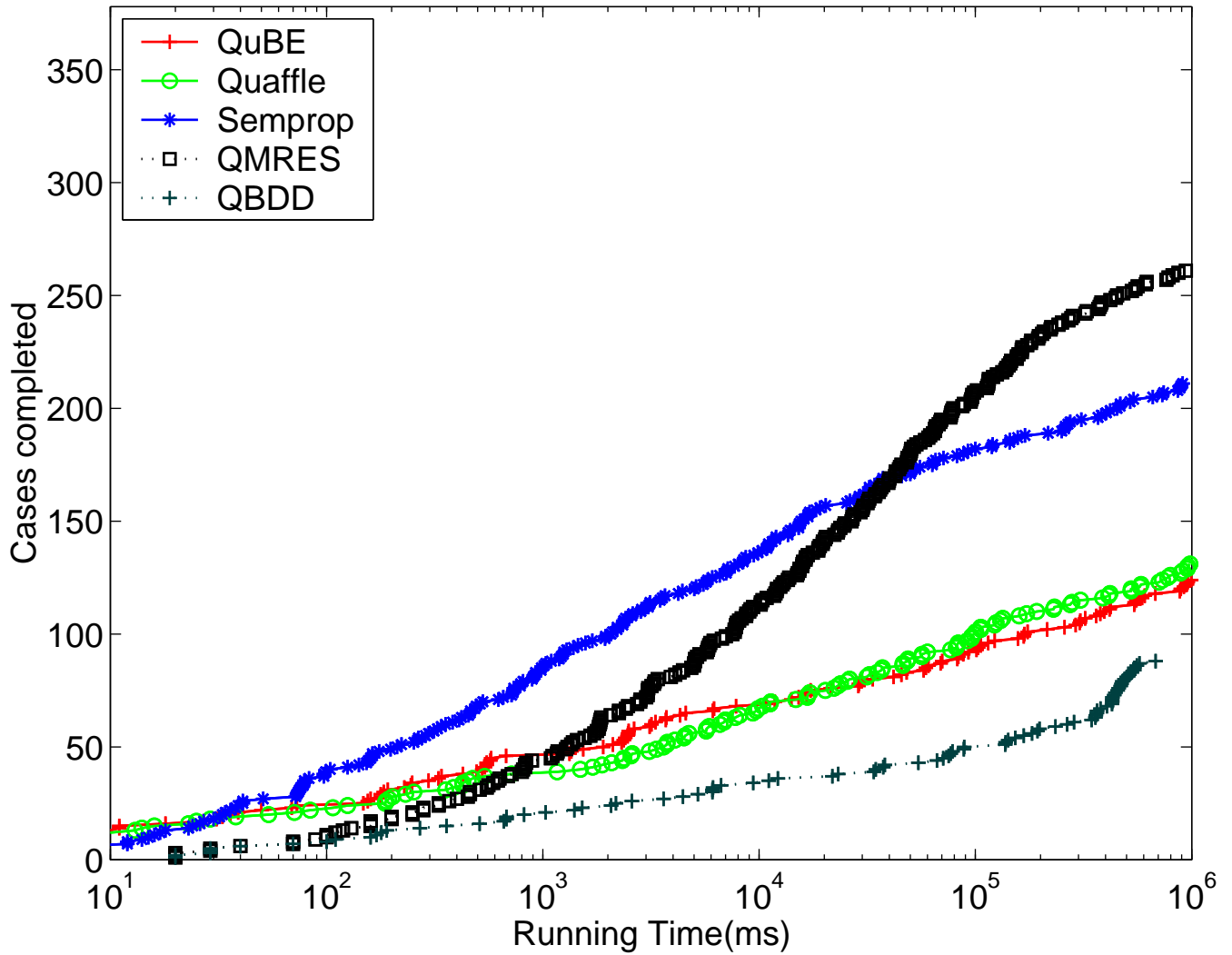


Figure 7: Modal-Logic Benchmarks

# Discussion

## My points:

- Symbolic techniques provide a viable approach to Boolean reasoning.
- Industrial tools are often hybrid engines.
- The SAT community is ignoring this approach.

## Questions:

- Are the competitions discouraging alternative approaches?
- How can we combine search and symbolic techniques?
- Do BDDs fit in the SMT framework?