Machine Learning and Logic: Fast and Slow Thinking

Moshe Y. Vardi

Rice University

Is Computer Science Fundamentally Changing?

Formal Science vs Data Science

- We are at peak hype about machine learning and big data!
- *Common perception*: A Kuhnian paradigm shift!

– "Throw out the old, bring in the new!"

- *In reality*: new scientific theories *refine* old ones.

– After all, we went to the moon with Newtonian Mechanics!

- *My Thesis*: Data science *refines* formal science!

Logic vs. Machine Learning

Daniel Kahneman, *Thinking, Fast and Slow*, 2011:

- **Machine Learning**: fast thinking, e.g., "Is this a stop sign?"
- **Logic**: slow thinking, e.g., "Do you stop at a stop sign?"

Example–Autonomous Vehicles: how to establish safety? [Shashua, '17]

- *Data Driven*: Drive 1B miles!
- *Data+Model Driven*: Combine data (1M miles) with reasoning.

Grand Challenge: Combine logic with machine learning!

Boolean Satisfiability

Boolean Satisfiability (SAT); Given a Boolean expression, using "and" ($\wedge$) "or", ($\vee$) and "not" ($\neg$), *is there a satisfying solution* (an assignment of 0's and 1's to the variables that makes the expression equal 1)?

Example:

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_1 \vee x_4)$$

Solution: $x_1 = 0$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$

Boolean Logic Today: assembly language for reasoning!

Algorithmic Boolean Reasoning: Early History

- Newell, Shaw, and Simon, 1955: "Logic Theorist"
- Davis and Putnam, 1958: "Computational Methods in The Propositional calculus", unpublished report to the NSA
- Davis and Putnam, JACM 1960: "A Computing procedure for quantification theory"
- Davis, Logemman, and Loveland, CACM 1962: "A machine program for theorem proving"
- Cook, 1971, Levin, 1973: Boolean Satisfiability is NP-complete.

DPLL Method: Propositional Satisfiability Test

- Convert formula to conjunctive normal form (CNF)
- Backtracking search for satisfying truth assignment
- Unit-clause preference

Modern SAT Solving

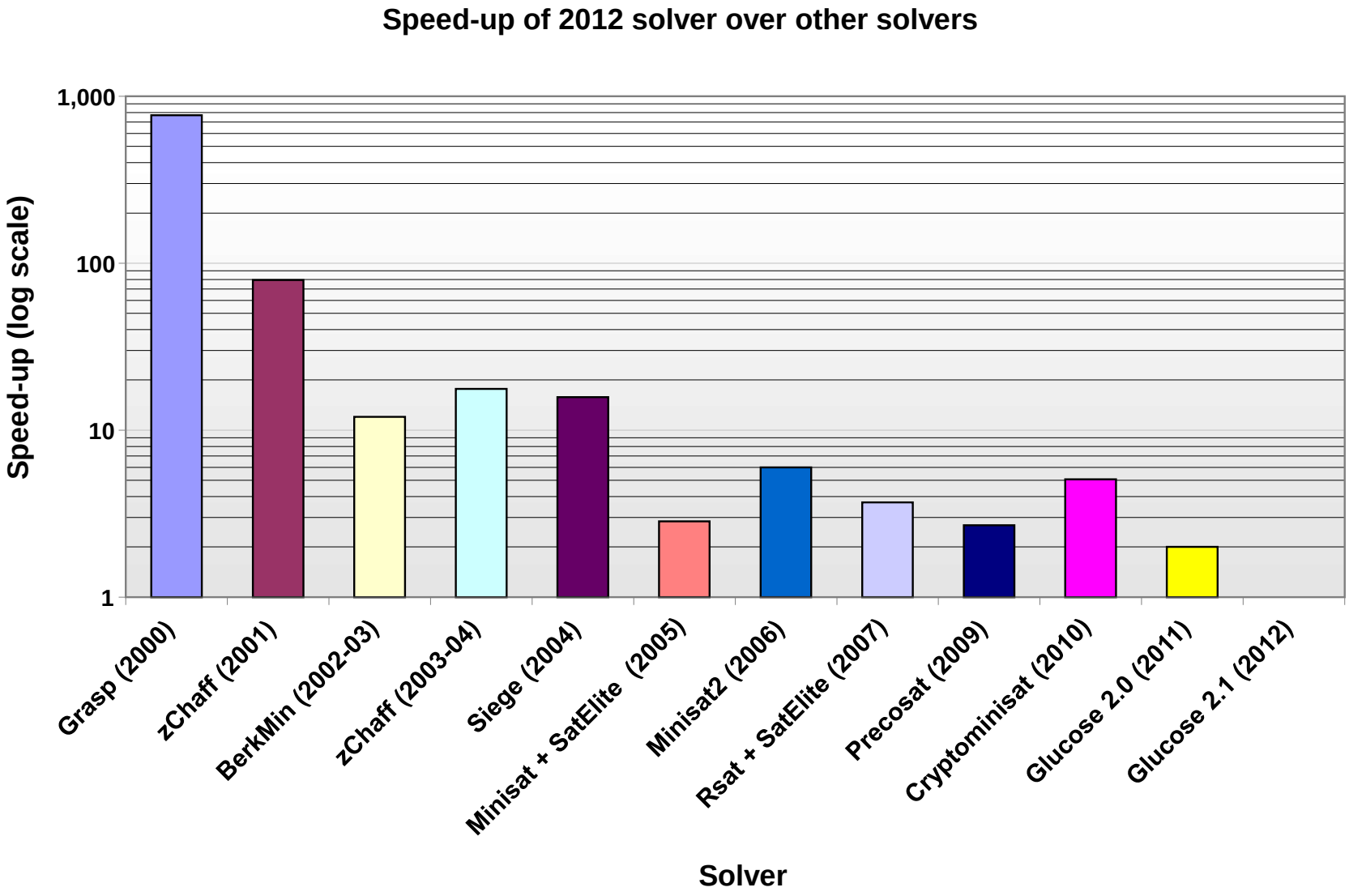
CDCL = conflict-driven clause learning

- Backjumping
- Smart unit-clause preference
- Conflict-driven clause learning
- Smart choice heuristic (brainiac vs speed demon)
- Restarts

Key Tools: GRASP, 1996; Chaff, 2001

Current capacity: *millions* of variables

Some Experience with SAT Solving



Applications of SAT Solving in SW Engineering

Leonardo De Moura+Nikolaj Björner, 2012: applications of Z3 at Microsoft

- Symbolic execution
- Model checking
- Static analysis
- Model-based design
- ...

Verification of HW/SW systems

HW/SW Industry: \$0.75T per year!

Major Industrial Problem: *Functional Verification* – ensuring that computing systems satisfy their intended functionality

- Verification consumes the majority of the development effort!

Two Major Approaches:

- *Formal Verification*: Constructing mathematical models of systems under verification and analzying them mathematically: $\leq 10\%$ of verification effort

- *Dynamic Verification*: simulating systems under different testing scenarios and checking the results: $\geq 90\%$ of verification effort

Dynamic Verification

- Dominant approach!
- Design is simulated with input test vectors.
- Test vectors represent different verification scenarios.
- Results compared to intended results.
- **Challenge**: Exceedingly large test space!

Motivating Example: HW FP Divider

$z = x/y$: x, y, z are 128-bit floating-point numbers

Question How do we verify that circuit works correctly?

- Try for all values of x and y?
- 2^{256} possibilities
- Sun will go nova before done! *Not scalable!*

Test Generation

Classical Approach: *manual* test generation - capture intuition about problematic input areas

- Verifier can write about 20 test cases per day: *not scalable!*

Modern Approach: *random-constrained* test generation

- Verifier writes *constraints* describing problematic inputs areas (based on designer intuition, past bug reports, etc.)

- Uses *constraint solver* to solve constraints, and uses solutions as test inputs – rely on industrial-strength constraint solvers!

- Proposed by Lichtenstein+Malka+Aharon, 1994: de-facto industry standard today!

Random Solutions

Major Question: How do we generate solutions *randomly* and *uniformly?*

- *Randomly*: We should not rely on solver internals to chose input vectors; we do not know where the errors are!

- *Uniformly*: We should not prefer one area of the solution space to another; we do not know where the errors are!

Uniform Generation of SAT Solutions: Given a SAT formula, generate solutions uniformly at random, while scaling to industrial-size problems.

Constrained Sampling: Applications

Many Applications:

- Constrained-random Test Generation: discussed above
- Personalized Learning: automated problem generation
- Search-Based Optimization: generate random points of the candidate space
- *Probabilistic Inference*: Sample after conditioning
- ...

Constrained Sampling – Prior Approaches, I

Theory:

- Jerrum+Valiant+Vazirani: *Random generation of combinatorial structures from a uniform distribution*, TCS 1986 – uniform generation in $BPP^{\Sigma_2^p}$

- Bellare+Goldreich+Petrank: *Uniform generation of NP-witnesses using an NP-oracle*, 2000 – uniform generation in BPP^{NP}.

But: We *implemented* the BPG Algorithm: did not scale above 16 variables!

Constrained Sampling – Prior Work, II

Practice:

- *BDD-based*: Yuan, Aziz, Pixley, Albin: *Simplifying Boolean constraint solving for random simulation-vector generation*, 2004 – poor scalability

- *Heuristics approaches*: MCMC-based, randomized solvers, etc. – good scalability, poor uniformity

Almost Uniform Generation of Solutions

New Algorithm – UniGen: Chakraborty, Fremont, Meel, Seshia, V, 2013-15:

- almost uniform generation in BPP^{NP} (randomized polynomial time algorithms with a SAT oracle)
- Based on *universal hashing*.
- Uses an *SMT solver*.
- Scales to 100,000s of variables.

Uniformity vs Almost-Uniformity

- Input formula: φ; Solution space: $Sol(\varphi)$
- Solution-space size: $\kappa = |Sol(\varphi)|$
- Uniform generation: for every assignment y: Prob$[Output = y]$=$1/\kappa$
- AImost-Uniform Generation: for every assignment y:
  $\frac{(1/\kappa)}{(1+\varepsilon)} \leq Prob[Output = y] \leq (1/\kappa) \times (1+\varepsilon)$

The Basic Idea

1. Partition $Sol(\varphi)$ into "roughly" equal small cells of appropriate size.

2. Choose a random cell.

3. Choose at random a solution in that cell.

You got random solution almost uniformly!

Question: How can we partition $Sol(\varphi)$ into "roughly" equal small cells without knowing the distribution of solutions?

Answer: *Universal Hashing* [Carter-Wegman 1979, Sipser 1983]

Universal Hashing

Hash function: maps $\{0,1\}^n$ to $\{0,1\}^m$

- Random inputs: All cells are roughly equal (in expectation)

Universal family of hash functions: Choose hash function *randomly* from family

- For *arbitrary* distribution on inputs: All cells are roughly equal (in expectation)

XOR-Based Universal Hashing

- Partition $\{0,1\}^n$ into 2^m cells.
- *Variables*: $X_1, X_2, \ldots X_n$
- Pick every variable with probability $1/2$, XOR them, and equate to 0/1 with probability $1/2$.
  - E.g.: $X_1 + X_7 + \ldots + X_{117} = 0$ (splits solution space in half)
- m XOR equations $\Rightarrow 2^m$ cells
- *Cell constraint*: a conjunction of CNF and XOR clauses

SMT: Satisfiability Modulo Theory

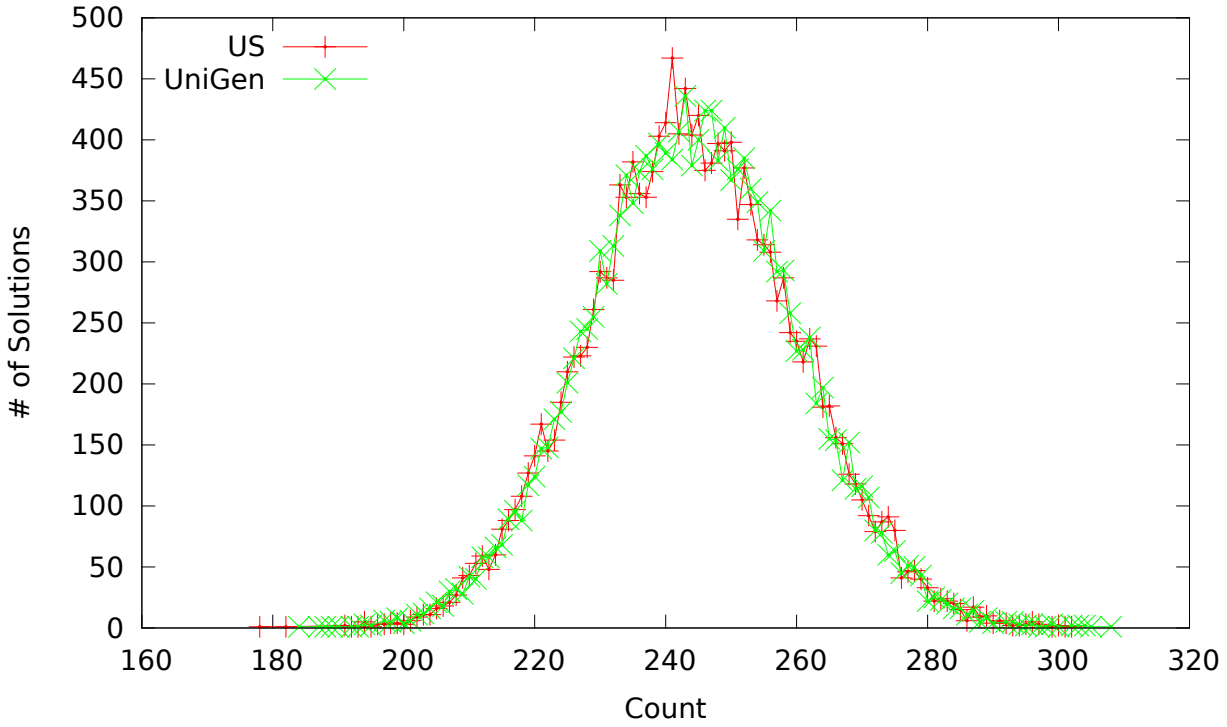
SMT Solving: Solve Boolean combinations of constraints in an underlying theory, e.g., linear constraints, combining SAT techniques and domain-specific techniques.

- Tremendous progress since 2000!

CryptoMiniSAT: M. Soos, 2009

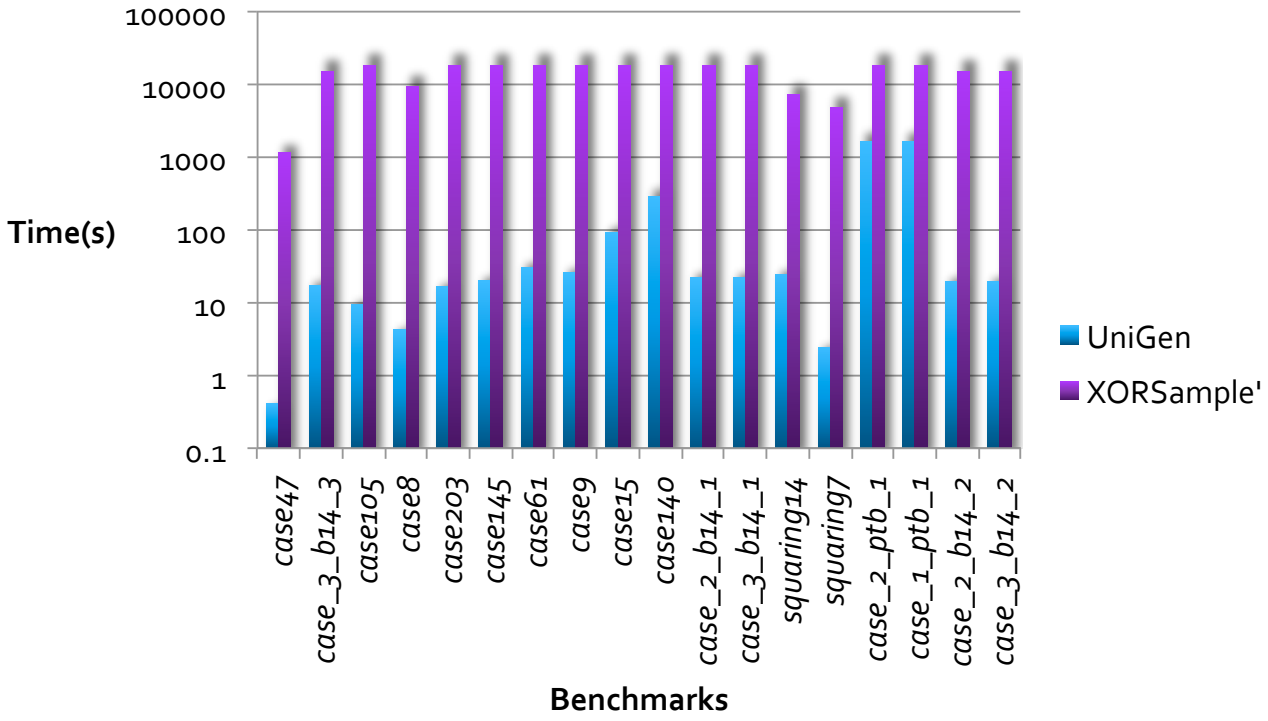
- Specialized for combinations of CNF and XORs
- Combine SAT solving with Gaussian elimination

UniGen Performance: Uniformity



Uniformity Comparison: UniGen vs Uniform Sampler

UniGen Performance: Runtime



Runtime Comparison: UniGen vs XORSample'

Are NP-Complete Problems Really Hard?

- When I was a graduate student, SAT was a "scary" problem, not to be touched with a 10-foot pole.
- Indeed, there are SAT instances with a few hundred variables that cannot be solved by any extant SAT solver.
- But today's SAT solvers, which enjoy wide industrial usage, routinely solve real-life SAT instances with millions of variables!

Conclusion We need a richer and broader complexity theory, a theory that would explain both the difficulty and the easiness of problems like SAT.

Question: Now that SAT is "easy" in practice, how can we leverage that?

- If not worst-case complexity, then what?

From Model-Driven Computer Science to Data-Driven Computer Science and Back

In Summary:

- It is a *paradigm glide*, not *paradigm shift*.
- Data-driven CS *refines* model-driven CS, it does *not* replace it.
- Physicists still teach Mechanics, Electromagnetism, and Optics.
- We should still teach Algorithms, Logic, and Formal Languages.
- But we must "marry" logic and probability.