Comparing Efficiency of SAT-solvers and Investigating Characteristics of Specific SAT Instances

Teofil Sidoruk

a joint work with Artur Niewiadomski, Wojciech Penczek and Piotr Switalski

ICS PAS, Warsaw, 08.10.2018

Outline I

1. Introduction
   - The SAT problem
   - Practical applications of SAT
   - Importance of SAT
2. SAT-solvers
   - SAT-solving algorithms
   - SAT-solvers selected for comparison
   - Selected problems in P, NP, PSPACE and EXPTIME
   - Encodings to SAT
3. Experimental Results
   - NP-complete problems
   - Chess problems
   - EXPTIME and PSPACE problems
   - Conclusions

Outline II

4. Investigating specific instances
   - Introduction
   - Existing research: order parameters and phase transition
   - Analysing formula composition
   - Conclusions

Boolean Satisfiability Problem (SAT)

- SAT - decision problem if there exists a satisfying assignment of variables for a Boolean formula
- First known **NP-complete problem**, proved independently by Cooke and Levin in the early 1970s
- Initially a subject of mainly academic discourse
- Dramatically gained importance having found numerous practical applications

(Some) Practical Applications of SAT

- Automated verification
- Model checking
- Planning
- Composition of web services
- Theorem proving
- Optimization
- Artificial intelligence
- Security and cryptography
- And others...

Importance of the SAT Problem

- Scientific interest further reinforced by practical applications
- **SAT competitions** organized annually since 2002
- Convenient 'common denominator' for hard computational problems
- Remains at the centre of the **P vs. NP hypothesis**

SAT-solving algorithms

DP - 1960 [Davis, Putnam]

- Based upon the **resolution rule**
- Quickly abandoned in favour of DPLL
  - potential memory explosion in worst case scenarios

DPLL - 1962 [Davis, Logemann, Loveland]

- Resolution-based inference replaced with the splitting rule
- Nature of the algorithm changed into a **backtracking scheme**
- The core idea remains the foundation of modern SAT-solvers

Recent developments

CDCL - 1996-now [Marques-Silva et al.]

- Evolution of the core idea of DPLL
- Non-chronological backtracking

Directions of further developments

- Continued improvements in
  - decision-making
  - efficiency of search and data storage
- Parallel processing

SAT-solvers selected for comparison

- **Lingeling** [A. Biere, JKU Linz, Austria]
- **Glucose** [L. Simon and G. Audemard]
- **Clasp** [University of Potsdam]
- **Minisat** [MIT]
- **ManySAT** [Y. Hamadi et al., based on Minisat]
- **Z3 Theorem Prover** [Microsoft Research]
- **zChaff** [Princeton University]

Selected problems in P, NP, PSPACE and EXPTIME

Classic NP-complete problems

- Three classic NP-complete problems from graph theory: graph k-colouring, vertex k-cover, Hamiltonian path.
- Random graph generation with a given number of vertices n
- Multiple (about 100) files generated for each test instance

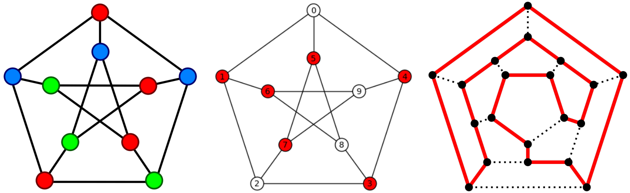


Figure: Examples of graph colouring (left), vertex cover (middle) and Hamiltonian path (right).

Selected problems in P, NP, PSPACE and EXPTIME

Post correspondence problem

- Undecidable in general
- NP-complete when solution length k is bounded

1	2	3	4
bab	bba	bab	b
ab	a	aa	bb

4	4	2	4	1
b	b	bba	b	bab
bb	bb	a	bb	ab

Figure: An example instance of PCP, for $n = 4$ and $\Sigma = \{a, b\}$, $W = (bab, bba, bab, b)$, $V = (ab, a, aa, bb)$. The solution $(4, 4, 2, 4, 1)$ corresponds to the word $\overline{w} = \overline{v} = bbbbabbab$.

Selected problems in P, NP, PSPACE and EXPTIME

Extended string-to-string correction problem (ESCP)

- Can string A be transformed into B in at most k steps?
- NP-complete when only *delete* (single character deletion) and *swap* (swapping of adjacent characters) are allowed operations

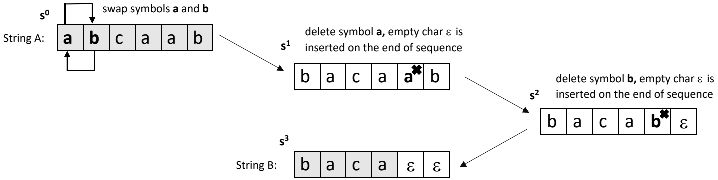


Figure: An example instance of ESCP with inputs: A = abcaab, B = baca, and the parameter $k = 3$.

Selected problems in P, NP, PSPACE and EXPTIME

P-Complete chess problems

- N-Queens
- Knight's Tour

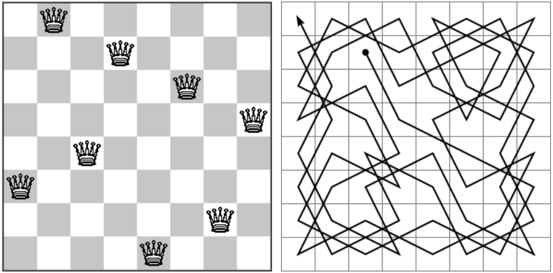


Figure: Examples of valid solutions for N-Queens (left) and Knight's Tour (right) on the standard 8×8 chessboard.

Selected problems in P, NP, PSPACE and EXPTIME

EXPTIME and PSPACE problems

- Towers of Hanoi (exponential in the size of the input)
- Model checking of UML systems

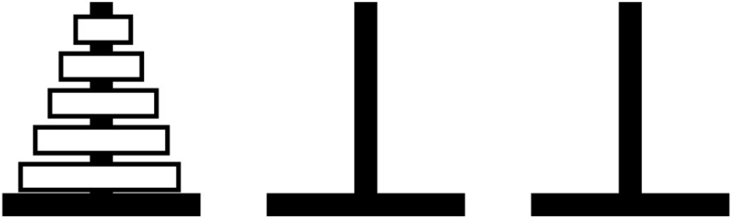


Figure: Initial state for the ToH puzzle with 5 discs.

Encodings to SAT for NP-complete problems

Graph k-colouring

- Variable $p_{i,j}$ denotes that the i-th vertex has the j-th colour
- $i = 1..n, j = 1..k$, encoding of size $O(n^2 \cdot k)$

Vertex k-cover

- Variable $p_{i,j}$ denotes that the i-th vertex is at j-th 'position' in the covering subset
- $i = 1..n, j = 1..k$, encoding of size $O(n^2 \cdot k)$

Hamiltonian path

- Variable $p_{i,j}$ denotes that the i-th vertex is at j-th position in the Hamiltonian path
- $i, j \in \{1, 2, ..., n\}$, encoding of size $O(n^2)$

Encodings to SAT for NP-complete problems

Bounded Post correspondence problem

- $\mathbf{w}$ and $\mathbf{v}$ are vectors of $k * m$ symbolic variables to represent numbers corresponding to the alphabet symbols,
- Vectors $\mathbf{p}^w$ and $\mathbf{p}^v$ (both of size $k+1$) encode positions in words $\overline{w}$ and $\overline{v}$, respectively,
- $\mathbf{id}$ is a vector of k symbolic variables encoding a solution
- Overall, we use $O(km * log(km))$ propositional variables to encode BPCP
- Encoding of size $O(k^4 m^4)$ (assuming $km > n$)

Encodings to SAT for NP-complete problems

Extended string-to-string correction problem

- We need to encode an initial string A and its k copies.
- $k+1$ vectors of symbolic variables, s^i for $i = 0..k$, corresponding to possible states of the string A after applying i edit operations.
- The strings are of length n, and so the vectors s^i consist of n symbolic variables, s^i_j, for $j = 1..n$.
- Each symbolic variable s^i_j is a sequence of propositions $s^i_{j,m} \in PV$, where $m = 0.. \lceil log_2(|\Sigma + 1|) \rceil$.
- Overall, we use $(k+1) * n * \lceil log_2(|\Sigma + 1|) \rceil$ propositional variables to encode ESCP.

Encodings to SAT for chess problems

N-Queens

- Variable $p_{i,j}$ denotes the placement of a queen in the i-th row and the j-th column
- $i, j \in \{1, 2, ..., n\}$, encoding of size $O(n^4)$

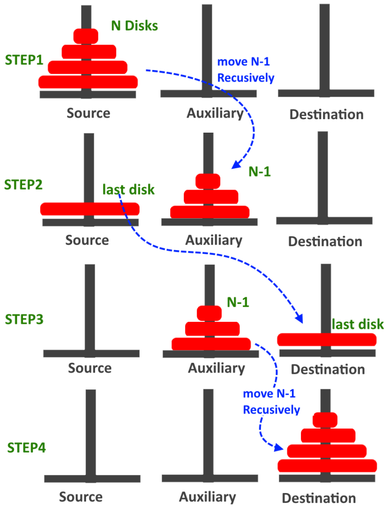
Knight's Tour

- Variable $p_{i,j,k}$ denotes that the k-th move is to the i-th row and j-th column
- $i, j \in \{1, 2, ..., n\}$, $k \in \{1, 2, ..., n^2\}$, encoding of size $O(n^4)$

Encoding to SAT for Towers of Hanoi

Towers of Hanoi

- The solution constitutes a sequence of valid moves
- Initial state as seen in step 1
- Final state as seen in step 2
- **Why stop there?**
- Moves: $max = (2^{n-1} - 1)$
- Discs: $j = 1..n$
- Towers: $t \in \{0, 1, 2\}$



Encoding to SAT for Towers of Hanoi

Towers of Hanoi

- By $D(j, i, t)$ we denote j-th disc on tower t in the i-th state
- Standard binary encoding with two propositional variables
- Initial state $\mathcal{I} = \bigwedge_{j=1}^{n} D(j, 0, 0)$
- $\mathcal{T}(i)$ represents all possible moves in the i-th state
- Final state $\mathcal{F} = \bigwedge_{j=1}^{n-1} \Big(D(j, max, 1) \Big) \wedge D(n, max, 0)$
- The problem encoded as $ToH(n) = \mathcal{I} \wedge \mathcal{F} \wedge \bigwedge_{i=0}^{max-1} \mathcal{T}(i)$
- Encoding of size $O(2^{n-1})$

Graph k-colouring

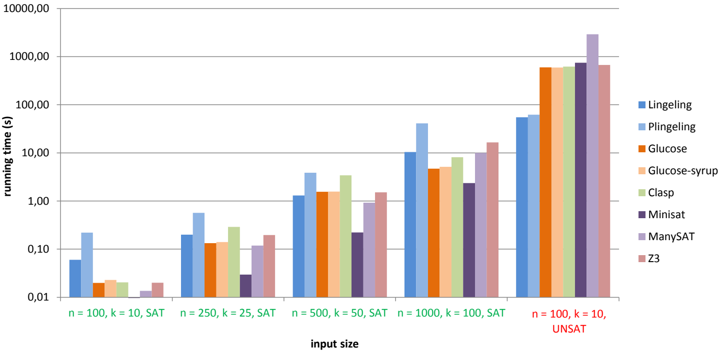


Figure: Modern solvers are lagging behind in SAT instances of the graph k-colouring test, possibly due to preprocessing time overhead. The situation changes in the UNSAT instance, where Lingeling outclasses the competition.

Vertex k-cover

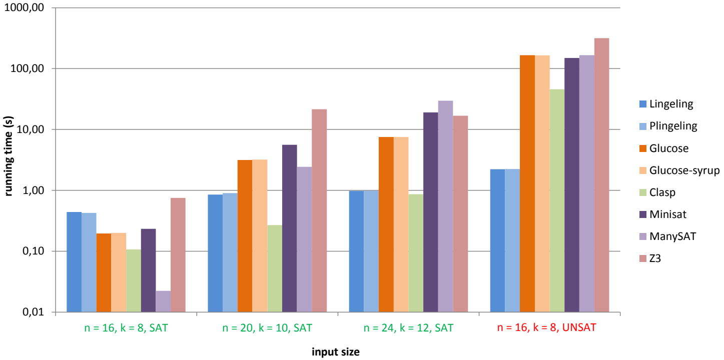


Figure: The vertex k-cover test proved to be more challenging, with older solvers unable to keep up the pace.

Hamiltonian path

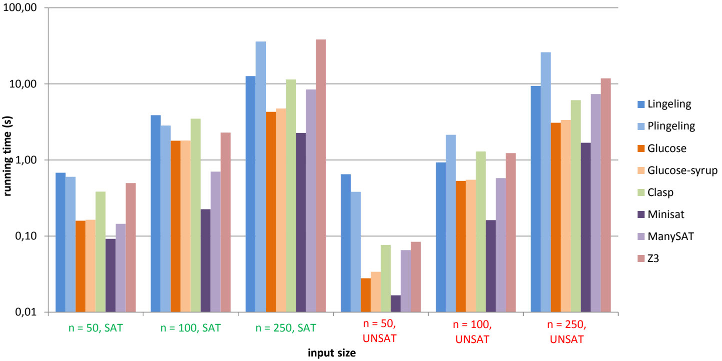


Figure: An anomaly can be observed in the Hamiltonian path test, where UNSAT instances were actually verified slightly faster.

Bounded Post correspondence problem

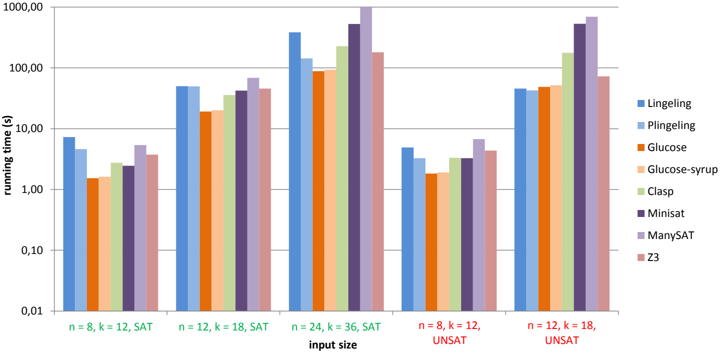


Figure: Glucose handled BPCP the fastest, with the exception of the largest UNSAT instance, where it finished marginally behind Lingeling.

Extended string-to-string correction problem

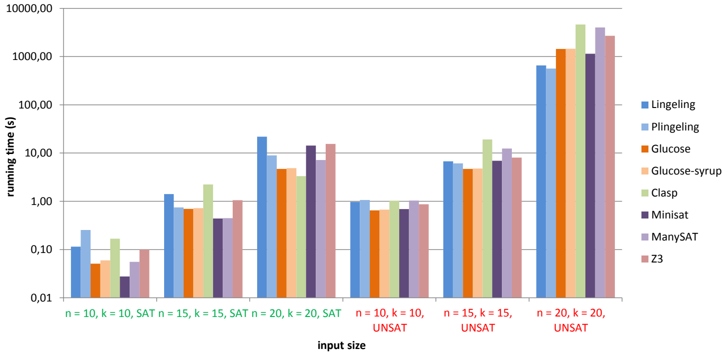


Figure: The difference in verification time of satisfiable and unsatisfiable instances was particularly large for ESCP, though individual solvers were rather closely matched.

N-Queens

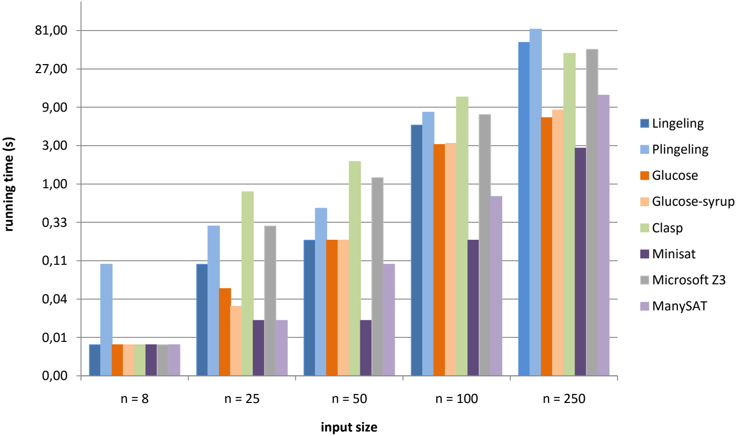


Figure: As expected, employing a SAT-solver for N-Queens is not the optimal solution.

Knight's Tour

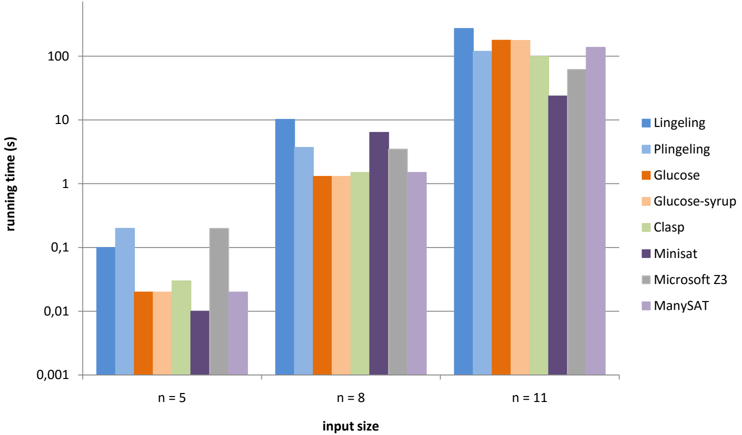


Figure: Non-viability of SAT-solvers for chess problems not in NP is even more apparent with Knight's Tour. Running time scales very poorly.

Towers of Hanoi

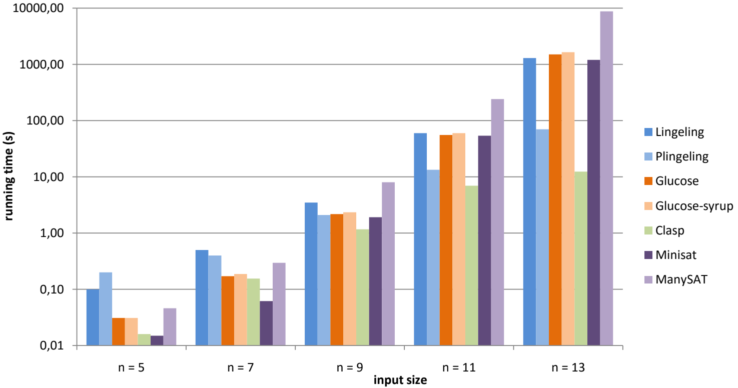


Figure: The ToH problem is exponential in the size of the input, resulting in running time increasing very quickly. The unquestioned winner is Clasp, especially for largest instances.

Model checking of UML systems

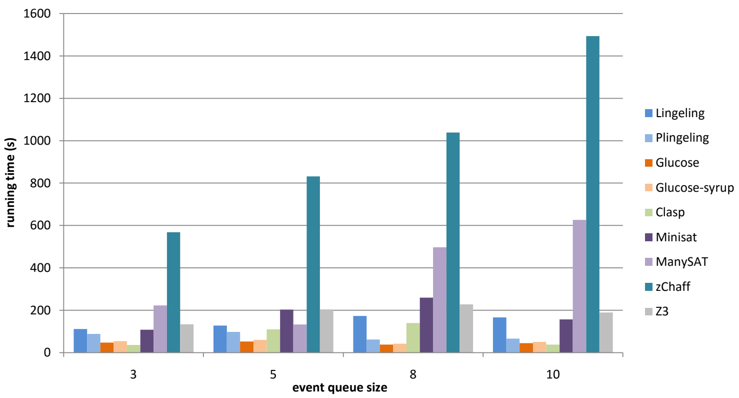


Figure: Modern SAT-solvers easily outperform the competition in verification of UML systems (a PSPACE problem).

Conclusions

- Superior (but not always!) performance of modern SAT-solvers
- Despite significant developments, **solving SAT remains a challenge**
- Continuing **need for further improvements**
- SAT-solvers quite **efficient for NP-complete and harder problems**, but far **inferior to tailored algorithms for P-complete problems**
- Implementation of parallel SAT-solving still in its early stages

What do we know already?

- Many problems, though NP-complete, have relatively easy **typical instances**
  - Graph k-coloring is *almost* always verifiable in LOGTIME
- Not a contradiction with complexity, proved to be in NP
- Thus, hard instances are bound to occur eventually
- What are the differences between easy and hard instances?
- Can we predict where the hardest instances will occur?

Order parameters and phase transition

- Several papers on the subject since early 1990s
- Instances of NP-complete problems can be described using **order parameters** [Cheeseman et al. 1991]
- **Clauses-to-variables ratio** often used to arrange sets of random SAT instances
- **Phase transition** observed as the order parameter is varied
- Distinct regions of likely satisfiable and unsatisfiable instances, both relatively easy to solve
- Hardest instances occur around the boundary

Example: phase transition for 3-SAT

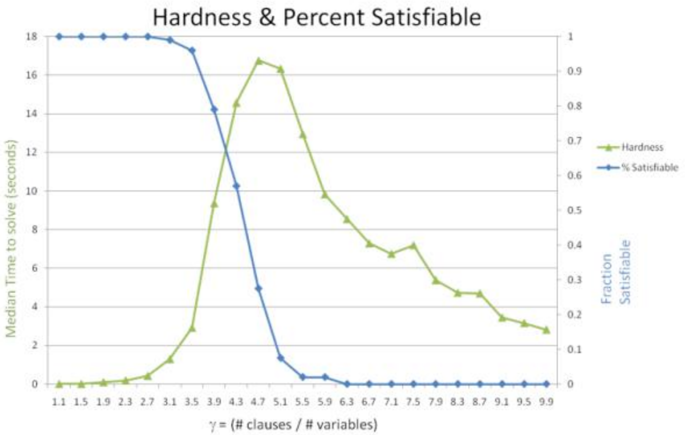


Figure: The phase transition for 3-SAT occurs at around $\gamma = 4.3$.

Example: phase transition for 3-SAT

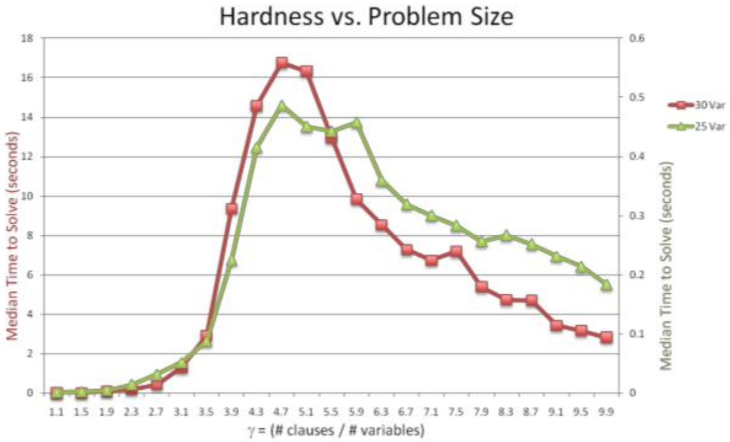


Figure: With larger formulas, the transition is more pronounced.

Example: phase transition for k-SAT

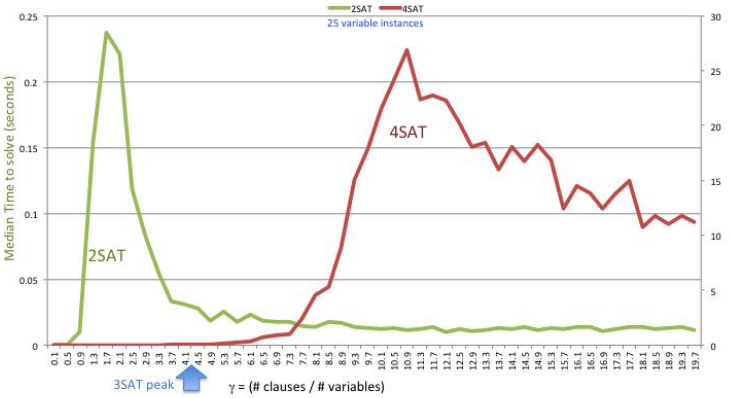


Figure: For k-SAT, the crossover point shifts to the right as k increases. Source: S. C. Kambhampati and T. Liu, *Phase Transition and Network Structure in Realistic SAT Problems*, 2013. (all three examples).

Constraint gap

- Experimental results confirm the expected easy-hard-easy pattern in principle
- Distribution of difficulty actually more complex
- **'Constraint gap'** postulated to be the reason behind unexpectedly hard instances [Gent, Walsh 1993]
- Hardest instances critically constrained, i.e., only have just enough constraints to be satisfiable
- DPLL forced to employ **splitting rule** extensively as a result

DPLL: a short recap

- **Backtracking scheme** whose core idea remains foundation of modern SAT-solvers
- Three rules: unit propagation, pure literal elimination, splitting
- The first two never branch out the search
- Only the splitting rule introduces exponential behaviour
  - heuristic-based choice of **branching literal**
- Hence, hardest instances tend to have a high ratio of splits to the other two rules of DPLL

Example: constraint gap

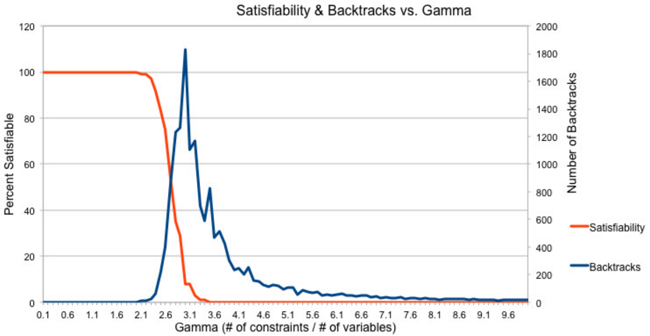


Figure: DPLL backtracks frequently in critically constrained instances. Source: S. C. Kambhampati and T. Liu, *Phase Transition and Network Structure in Realistic SAT Problems*, 2013.

But what about formula composition?

- Not attempting to experimentally prove previous findings on difficulty distribution yet again
- Analysing formula composition in search of patterns instead
- Benchmarks from previous paper used (comparison of SAT-solvers)
- Average running times from tested solvers (except zChaff) taken into account

Results for vertex k-colouring

	Easier instances	Harder instances
Avg running time	0.018 s	364.759 s
Avg number of variables	1000	1000
Avg number of clauses	24507	36007
Avg number of literals	49813	72813
Avg clause size	2.033	2.022
Longest clause	10	10
Negative literals	97.96%	98.63%
Horn clauses	0%	0%
Clauses-to-vars ratio (γ)	24.51	36.01

Table: Comparison of characteristics between easier and harder instances of the vertex k-colouring problem.

Results for vertex k-cover

	Easier instances	Harder instances
Avg running time	0.108 s	38.941 s
Avg number of variables	1500	1500
Avg number of clauses	36774	36994
Avg number of literals	74967	88127
Avg clause size	2.039	2.382
Longest clause	60	60
Negative literals	98.04%	83.41%
Horn clauses	99.03%	99.34%
Clauses-to-vars ratio (γ)	24.52	24.67

Table: Comparison of characteristics between easier and harder instances of the vertex k-cover problem.

Results for Hamiltonian path

	Easier instances	Harder instances
Avg running time	4.810 s	34.101 s
Avg number of variables	40000	40000
Avg number of clauses	8000200	8000200
Avg number of literals	20135132	17115771
Avg clause size	2.517	2.139
Longest clause	200	200
Negative literals	80.66%	93.25%
Horn clauses	0%	0%
Clauses-to-vars ratio (γ)	200.01	200.01

Table: Comparison of characteristics between easier and harder instances of the Hamiltonian path problem.

Results for string-to-string correction

	Easier instances	Harder instances
Avg running time	0.538 s	42.968 s
Avg number of variables	3995	3995
Avg number of clauses	49808	49808
Avg number of literals	318476	318476
Avg clause size	6.394	6.394
Longest clause	66	66
Negative literals	19.67%	19.67%
Horn clauses	0%	0.01%
Clauses-to-vars ratio (γ)	12.47	12.47

Table: Comparison of characteristics between easier and harder instances of the string-to-string correction problem (ESCP).

Conclusions

- No noticeable, prevalent pattern for easier or harder instances of problems
- Characteristics of formulas dependent on specifics of the problem and its SAT encoding
- No easy answers to be expected when dealing with NP-complete problems
- Just as no single solver is always better, there are no particular characteristics always contributing to a particular instance being more difficult

THANK YOU