Ten Challenges *Redux*: Recent Progress in Propositional Reasoning and Search

Henry Kautz[1] and Bart Selman[2]

[1] Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195 USA
kautz@cs.washington.edu
[2] Department of Computer Science
Cornell University
Ithaca, NY 14853 USA
selman@cs.cornell.edu

Abstract. In 1997 we presented ten challenges for research on satisfiability testing [1]. In this paper we review recent progress towards each of these challenges, including our own work on the power of clause learning and randomized restart policies.

1 Introduction

The past few years have seen enormous progress in the performance of Boolean satisfiability (SAT) solvers. Despite the worst-case exponential run time of all known algorithms, SAT solvers are now in routine use for applications such as hardware verification [2] that involve solving hard structured problems with up to a million variables [3, 4]. Each year the International Conference on Theory and Applications of Satisfiability Testing hosts a SAT competition that highlights a new group of "world's fastest" SAT solvers, and presents detailed performance results on a wide range of solvers [5, 6]. In the the 2003 competition, over 30 solvers competed on instances selected from thousands of benchmark problems.

In 1997, we presented ten challenges for research on satisfiability testing [1], on topics that at the time appeared to be ripe for progress. In this paper we revisit these challenges, review progess, and offer some suggestions for future research.

A full review of the literature related to the original challenges, let alone satisfiability testing as a whole, is beyond the scope of this paper. We do highlight several of the main recent developments, but the discussion below is biased towards topics from our own research program in recent years. We welcome pointers to any key papers we may have missed. We plan to keep this document up-to-date with regular revisions posted on the SAT Challenge web page [7].

2 Challenging SAT Instances

Empirical evaluation of sat solvers on benchmark problems (such as those from [8]) has been a effective driving force for progress on both fundamental algo-

rithms and theoretical understanding of the nature of satisfiability. The first two challenges were specific open SAT problems, one random and the other highly structured.

CHALLENGE 1: *Prove that a hard 700 variable random 3-SAT formula is unsatisfiable.*

When we formulated in this challenge in 1997, complete SAT procedures based on DPLL [9] could handle around 300 to 400 variable hard random 3-SAT problems. Progress in recent years had slowed and it was not clear DPLL could be much improved upon for random 3-SAT. In particular, the there was the possibility that the best DPLL methods were obtaining search trees that were close to minimal in terms of the number of backtrack points [10]. Dubois and Dequen [11], however, showed that there was still room for improvement. They introduced a new branching heuristic that exploits so-called "backbone" variables in a SAT problem. A backbone variable of a formula is a variable that is assigned the same truth value in all assignments that satisfy the maximum number of clauses. (For satisfiable formulas, these are simply the satisfying assignments of the formula.) The notion of a backbone variable came out of work on k-SAT using tools from statistical physics, which has provided significant insights into the solution structure of random instances. In particular, it can be shown that a relatively large set of backbone variables suddenly emerges when one passes though the phase transition point for k-SAT ($k \geq 3$) [12]. Using a backbone-guided search heuristic, Dubois and Dequen can solve a 700 variable unsatisfiable, hard random 3-SAT instance in around 25 days of CPU time, thereby approaching practical feasibility.

In the context of this challenge, it should be noted that significant progress has been made in the last decade in terms of our general understanding of the properties of random 3-SAT problems and the associated phase transition phenomenon. A full review of this area would require a separate paper. (See e.g. [13–22].) Many of the developments in the area have been obtained by using tools from statistical physics. This work has recently culminated in a new algorithm for solving satisfiable k-SAT instances near the phase transition point [23]. The method is called survey propagation and involves, in a sense, a sophisticated probabilistic analysis of the problem instance under consideration. An efficient implementation enables the solution of hard random 3-SAT phase transition instances of up to a million variables in about 2 hours of CPU time. For comparsion, the previously most effective procedure for random 3-SAT, WalkSAT [24], can handle instances with around 100,000 variables within this timeframe. The exact scaling properties of survey propagation — and WalksSAT for that matter — are still unknown.

In conclusion, even though we have seen many exciting new results in terms of solving hard random instances, the gap between our ability to handle satisfiable and unsatisfiable instances has actually grown. An interesting question is whether a procedure dramatically different from DPLL can be found for handling unsatisfiable instances.

CHALLENGE 2: *Develop an algorithm that finds a model for the DIMACS 32-bit parity problem.*

The second challenge problem derives from the problem of learning a parity function from examples. This problem is NP-complete and it is argued in [25] that any particular instance is likely to be hard to solve (although average-case NP-completeness has not been formally shown). However, this challenge was solved in 1998 by preprocessing the formula to detect chains of literals that are equivalent considering binary clauses alone, and then applying DPLL after simplification [26].[3] Later [27] showed similar performance by performance equivalency detection at every node in the search tree.

Parity problems are particularly hard for local search methods because such algorithms tend to become trapped at a near-solution such that a small subset of clauses is never satisfied simultaneously. Clause re-weighting schemes [28, 29] try to smooth out the search space by giving higher weight to clauses that are often unsatisfied. A clause weighting scheme based on Langrange multipliers [30] was able to solve the 16-bit versions of the parity learning problems.

3 Challenges for Systematic Search

At the time of our original challenge paper nearly all the best systematic methods for propositional reasoning on clausal formulas were based on creating a resolution proof tree.[4] This includes the depth-first search Davis-Putnam-Loveland-Logemann procedure (DPLL) [33, 34], where the proof tree can be recovered from the trace of the algorithm's execution, but is not explicitly represented in a data structure (the algorithm only maintains a single branch of the proof tree in memory at any one time). Most work on systematic search concentrates on heuristics for variable-ordering and value selection, all in order to the reduce size of the tree.

However, there are known fundamental limitations on the size of the shortest resolution proofs that can be obtained in this manner, even with ideal branching strategies. The study of proof complexity [35] compares inference systems in terms of the sizes of the shortest proofs they sanction. For example, two proof systems are linearly related if there is a linear function $f(n)$ such that for any proof of length n in one system there is a proof of length at most $f(n)$ in the other system. A family of formulas C provides an *exponential separation* between systems S_1 and S_2 if the shortest proofs of formulas in C in system S_1 are exponentially smaller than the corresponding shortest proofs in S_2.

A basic result in proof complexity is that general resolution is exponentially stronger than the DPLL procedure [36, 37]. This is because the trace of DPLL running on an unsatisfiable formula can be converted to a tree-like resolution

[3] [26] also described a general preprocessor for identifying conjunctions of nested equivalencies subformulas using linear programming.

[4] Much work in verification has involved non-clausal representations, in particular Boolean Decision Diagrams [31, 32]; but the large body of work on BDD's will not be further discussed here.

proof of the same size, and tree-like proofs must sometimes be exponentially larger than the DAG-like proofs generated by general resolution. Furthermore, it is known that even general resolution requires exponentially long proofs for for certain "intuitively easy" problems [38–40]. The classic example are "pigeon hole" problems that represent the fact that n pigeons cannot fit in $n-1$ holes. Shorter proofs do exist in more powerful proof systems. Examples of proof systems more powerful than resolution include extended resolution, which allows one to introduce new defined variables, and resolution with symmetry-detection, which uses symmetries to eliminate parts of the tree without search. Assuming $NP \neq co-NP$, even the most powerful propositional proof systems would require exponential long proofs worst case — nonetheless, such systems provably dominate resolution in terms of minimum proof size.

Early attempts to mechanize proof systems more powerful than tree-like resolution gave no computational savings, because it is *harder* to find the small proof tree in the new system than to simply crank out a large resolution proof. In essence, the overhead in dealing with the more powerful rules of inference consumes all the potential savings. Our third challenge was to present a practical proof system more powerful than resolution. In reviewing progress in this area we first consider systems more powerful than tree-like (DPLL) resolution, and next ones more powerful than general resolution.

3.1 Beyond DPLL

CHALLENGE 3A: *Demonstrate that a propositional proof system more powerful than tree-like resolution can be made practical for satisfiability testing.*

Two new satisfiability testing algorithms were introduced in 1997, the same year as our challenge paper: rel-sat [41] and SATO [42]. Both were versions of DPLL augmented with "conflict clause learning", a technique that grew out of research in AI on explanation-based approaches to speed-up learning [43–45]. The idea in clause learning is that at each backtrack point the system derives a reason for the inconsistency in the form of a new clause added to the original formula. Rel-sat and SATO were suprisingly powerful, and even able to solve open problems in finite mathematics. Clause learning was further developed for the solvers GRASP [46], Chaff [47, 48] and BerkMin [49], and is currently a key technique in backtracking SAT solvers for applications such as verification.

Marquis-Silva [50] observed that clause learning can be viewed as adding resolvents to a tree-like proof, and Zhang [48] showed how different clause learning schemes could be categorized according to way clauses were derived from cuts in a data structure called a *conflict graph*. The conflict graph records the pattern of *unit propagations* that have been performed at any point in the execution of the algorithm. Each node in the graph is a literal that is currently assumed to be true. The leaves are branch literals and the inner nodes are literals derived by unit propagation. A conflict literal is one that appears both negatively and positively in the graph.

Consider the implication graph at a stage where there is a conflict and fix a conflict graph contained in that implication graph. Pick any cut in the conflict

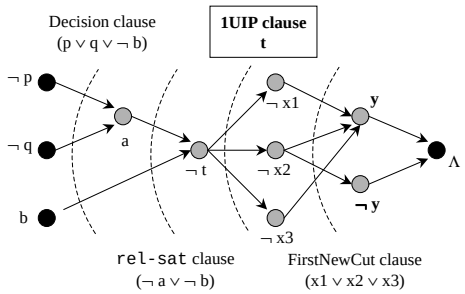

Fig. 1. A conflict graph depicting various learning schemes.

graph that has all decision variables on one side, called the *reason side*, and *false* as well as at least one conflict literal on the other side, called the *conflict side*. All nodes on the reason side that have at least one edge going to the conflict side form a *cause* of the conflict. The negations of the corresponding literals forms the *conflict clause associated with this cut*: that is, a clause that is implied by the original formula. Figure 1 illustrates different possible cuts through a conflict graph, corresponding to different clause learning algorithms.

Although the empirical power of clause learning had been clear for several years, Beame *et al.* [51] provided the first proof of an exponential separation between clause learning and ordinary DPPL. The result was, in fact, even stronger: they showed that there are formulas with short clause learning proofs that require exponentially large *regular resolution* proofs. Regular resolution proofs are DAGS, as in general resolution, but are restricted so that no variable is resolved upon more than once in any path from the root to a leaf. It is easy to see that all tree-like proofs are regular but not vice-versa. They further showed that combining clause learning with *restarts* [52, 53] (where learned clauses are saved between restarts) is equivalent to general resolution. However, the questions of whether clause learning is *strictly stronger* than regular resolution — that is, whether or not there are also formulas with short regular proofs but long clause proofs – and whether clause learning *without* restarts is equivalent to general resolution are open.

Making clause learning work well in practice requires efficient strategies for mananging the large number of learned clauses. The first technique developed for this management problem was relevance-bounded learning [41, 42]. The idea is to discard a learned clause once it is unlikely to be useful later on in the proof. A simple but effective strategy is to throw out clauses of length greater than some fixed k when the search backtracks above the point at which any of the literals in the clause are assigned a value [41]. A second important management technique, called "watched literals", was most fully exploited in Chaff [47]. Watched literals is actually a generic technique for reducing the time needed to tell which clauses have been shortened to length one during the DPPL's unit propagation step. Two literals are arbitrarily chosen in each clause to be "watched". When a

literal is set, rather than scanning through all clauses containing the negation of the literal, the algorithm only scans clauses contained *watched* negations of the literal. It is easy to see that this technique still finds all unit clauses, because such a clause is guaranteed to be scanned once it becomes a binary clause. Watched literals allows modern solvers to handle millions of learned clauses with small time overhead (although space can then become problematic).

Clause learning strategies and variable branching strategies have traditionally been studied separately. However, [54] shows that there is great promise in developing branching strategies that explicitly take into account the order in which clauses are learned. They considered a class of formulas known as pebbling formulas [36, 55–57], which can be thought of as representing precedence graphs in dependent task systems and scheduling scenarios. Such formulas require exponential-sized proofs for tree-like resolution, but have polynomial clause-learning proofs. However, it remains difficult to *find* such proofs. [54] preprocesses the formula to extract a *domain-specific* branching sequence — that is, a branching order that can be formally shown to yield small clause learning proofs for formulas encoding pebbling graphs. While ordinary DPLL (with a good branching order) scales to problems with about 60 variables on the pebbling formulas, and clause learning alone scales to 4,000 variables, clause learning with the domain specific ordering handles over 2,000,000 variables. To make this work of practical use we need to develop domain-specific strategies for other common structures that arise in applications such as verification or planning, and automated or semi-automated techniques for recognizing the structures.

3.2 Beyond General Resolution

CHALLENGE 3B: *Demonstrate that a propositional proof system more powerful than general resolution can be made practical for satisfiability testing.*

Currently the most practical extension of general resolution is symmetry detection. The pigeon hole problem is intuitively easy because we immediately see that different pigeons and holes are indistinguishable, so we do not need to actually consider all possible matchings — without loss of generality, attempting to find a particular (say, lexigraphically ordered) matching suffices. [58] showed how to determine if there existed a renaming (permutation) ψ of the variables in a formula that resulted in the same set of clauses, which justified a new rule of inference: from any clause $(a \vee b \vee ...)$, infer $(\psi(a) \vee \psi(b) \vee ...)$. [59] introduced a different way of using symmetries, by strengthening the formula through the addition of clauses that ruled out all but one of the symmetric cases. The drawback of this approach appeared to be the large (quadratic) number of symmetry breaking clauses needed; but [60] showed that a linear sized set of symmetry-breaking predicates was logically equivalent, and led to dramatic speedup on certain structured benchmark problems. Symmetry detection is not, however, a cure-all; [61] showed that any formula that was exponential for resolution could be transformed into one that was still exponential for resolution plus symmetry detection, by adding new literals and clauses that "hid" the symmetry.

As we have noted clause learning alone does not exceed the power of general resolution. However, if instead of cacheing conflicts, one modifies DPLL so that the entire residual formula at each node in the search tree is cached, then the proof complexity of the resulting system can exceed resolution [62] (if the test for a cached formula includes subsumption checking). Furthermore, [63] argues that formula caching is the fastest practical algorithm for *counting* the number of solutions of formula.

CHALLENGE 4: *Demonstrate that integer programming can be made practical for satisfiability testing.*

Over the years, there has been a significant amount of work on the close connection between 0/1 integer programming and SAT (e.g., [64, 65]). A key question is whether techniques developed for integer programming can be of use in SAT solvers. So far, it has been difficult to obtain a concrete computational advantage of integer programming methods on practical SAT instances. The recent work by Warners and van Maaren provides two promising examples of where integer programming and related techniques may have an impact. First, as discussed above, linear programming can be used in a two-phase algorithm for the 32-bit parity formulas [26]. Secondly, by using a semi-definite programming formulation, pigeon hole formulas can be solved efficiently [66]. The challenge remains to incorporate these approaches in more general, practical SAT solvers.

In recent years, we have also seen an interesting development in the opposite direction: use SAT techniques in the design of more efficient solvers for 0/1 integer programming problems. More specifically, one considers pseudo-Boolean encodings, which use Boolean variables and linear inequalities over such variables with integer coefficients. Most interestingly, some of the best solvers for pseudo-Boolean problems are extensions of the best SAT solvers [67–69].

4 Challenges for Stochastic Search

CHALLENGE 5: *Design a practical stochastic local search procedure for proving unsatisfiability.*

Given the success of local search style procedures on satisfiable problem instances, it would be interesting to use a local search strategy for finding "proof objects", *i.e.*, objects that demonstrate the unsatisfiability of an instance. This challenge remains wide open. A key issue is the need to find smaller proof objects. Work on strong backdoor sets, which are small sets of variables that, together with a polytime propagation method, can demonstrate unsatisfiability may lead to some new opportunities in this area [70].

CHALLENGE 6: *Improve stochastic local search on structured problems by efficiently handling variable dependencies.*

DPLL procedures handle variable dependency quite effectively through unit propagation. Local search methods, such as Walksat, handle dependencies through a random walk process, which may require on the order of N^2 flips to travel a dependency chain of N variables [71]. Given the large number of dependent variables in structured instances, the local search methods therefore are often less

effective than local search style methods. Note that this is not always the case. For example, in runs on verification benchmarks, Velev [3] showed how the performance of DLM [72] and Walksat [24, 73] is comparable to many of the best DPLL style methods. A series of papers, such as [74–79] among others, has also led to a much improved understanding of local search methods for SAT.

Hirsch [80] introduces a local search procedure, UnitWalk, where variable dependencies are propagated explicitly as part of the search process. The propagation strategy is closed related to the one studied in [81]. UnitWalk is quite effective on certain classes of structured problems but there is still room for improvement. Comparisons with WalkSat shows that neither strategy dominates. This led to QingTing [82], which is a local search solver that dynamically switches between a UnitWalk and a Walksat strategy, depending on the underlying structure of of the problem.

In a different approach to handling dependencies, in [71], redundant clauses are added to the SAT problem instances in a preprocessing phase. The redundant clauses capture long range dependencies between variables. It can be shown, both theoretically and empirically, that such redundant clauses speed up a local search style solver.

Although the challenge problem was formulated specifically in the context of local search methods, techniques for discovering and exploiting various forms of variable dependencies have also been shown to be effective for DPLL style procedures. See, for example, [83–85].

5 Randomized Systematic Search

CHALLENGE 7: *Demonstrate the succesful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.*

[86, 87] present hybrid approaches, integrating a local search and a DPLL solver. This work provides a promising step towards hybrid solvers, but it remains a challenge to have such solvers outperform non-hybrids on a wide range of benchmark problems.

We implicitly assumed in this challenge, as was common at the time, that stochastic search refers to some form of local search. Systematic, complete methods, such as DPLL, were generally deterministic. A major recent change during the last five years came out of the insight that adding randomization to a complete search method, combined with a restart strategy, can provide a significant computational advance [52]. (Note that explicit randomization is not required. For example, clauses learning between restarts of a DPLL solver, such as used in Chaff, also forces explorations of different parts of the search space on different restarts.)

Randomization and restarts take advantage of the large variations that have been observed between different runs of backtrack search procedures on a given problem instance. In fact, it has been shown that randomized DPLL run time distributions are often — but not always — "heavy-tailed" [88–91]. This means

that one observes a mixtures of run times on dramatically different scales. By using rapid restarts, one can take advantage of the occasionally short, successful run [52]. In a recent paper [70], it was shown that such short runs can be explained by the existence of a small set of *backdoor* variables in the problem instance. Once backdoor variables are assigned a value, the polytime propagation and simplication mechanism of the solver under considaration sets the remaining variables without further backtracking. (In case of a unsatisfiable instances, the propagation mechanisms discovers an inconsistancy after propagation.) Practical problem instances can have surprisingly small sets of backdoor variables. We have observed structured instances with tens of thousands of variables with backdoor sets of around a dozen variables. Randomization and restarts, in conjunction with the variable selection heuristics, help the solver discover the backdoor sets. Work on backdoor variables and clause learning, as discussed above, is providing us with a better understanding as to why structured SAT instances with up to a million variables, from, e.g., verification applications, can be solved with current state-of-the-art solvers.

An important related issue is how to decide on a good restart policy. Luby *et al.* [92] described restart policies for general randomized algorithms for two scenarios where runtime itself is the only observable: (i) when each run is a random sample from a known distribution, one can calculate a fixed optimal cutoff; (ii) when there is no knowledge of the distribution, a *universal schedule* mixing short and longer cutoffs comes within a log factor of the minimal run time.

Horvitz *et al.* [93] showed that it is possible to do better than Luby's fixed optimal policy by making observations of a variety of features related to the nature and progress of problem solving during an early portion of the run (referred to as the *observation horizon*) and learning, and then using, a Bayesian model to predict the length of each run. Examples of features of a running SAT solver (satz) included the minimum, maximum, final, and average values of (1) The number of backtracks; (2) The number of unit propagations; (3) Domain-specific measures of the current subproblem (for example, for a coloring problem, the number of nodes that have been colored), as well as the derivatives of such values. Under the assumption that each run is an independent random sample of one runtime distribution (RTD), [94] used observations to discriminate the potentially short runs from the long ones and then adopted different restart cutoffs for the two types of runs.

Ruan *et al.* [95] considered the case where there are k known distributions, and each run is a sample from *one* of the distributions—but the solver is not told *which* distribution. The paper showed how offline dynamic programming can be used to generate the optimal restart policy, and how the policy can be coupled with real-time observations to control restarting. In recent work the same authors [96] generalize this to the case where the k distributions are not specified in advance: instead, the solver first infers how a problem ensemble can be decomposed into a set of sub-ensembles such that each sub-ensemble clusters instances with similar runtime distributions.

The following example from [96] illustrates this approach where instances are clustered by their median runtime. Suppose that the RTD of each instance is a scaled Pareto distribution controlled by a parameter b:

$$P(t) = \begin{cases} b/t^2 \text{ if } t \geq b \\ 0 \quad \text{ if } t < b \end{cases}$$

This is a canonical example of a heavy-tailed distribution. Furthermore, suppose that b is an integer that is uniformly distributed in the range $[11, 100]$ across the problem distribution. The median run time of any particular instance is $2b$, so we expect that median run times of the sampled instances would fall uniformly in the range $[22, 200]$. A binary clustering by the median run times of the samples should give one cluster where the instance medians are in the range $[22, 110]$ (equivalently, $b \in [11, 55]$) and another cluster where the instance medians are in the range $[111, 200]$ (equivalently, $b \in [56, 100]$). Each cluster, or sub-ensemble, yields an *ensemble* run time distribution. The ensemble distribution RTD is the normalized sum of the RTD's of the instances it contains.

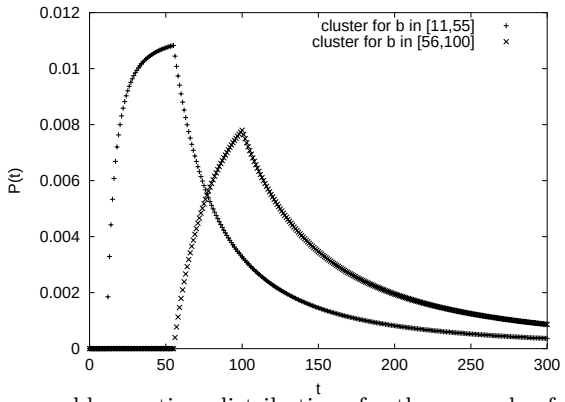

Fig. 2. The sub-ensemble run-time distributions for the example of a family of scaled Pareto distributions.

Fig. 2 shows the sub-ensemble RTD's for this example. The ensemble RTD's are not simple scaled Pareto distributions, because there is a non-zero probability density to the left of the maximum points. One can show (analytically or by computer simulation) that the optimal cutoffs for the two clusters are at 98 and 244 respectively. The dynamic programming procedure mentioned above can then be used to calculate a complete policy—in the case of the example where there are no run-time feature observations, this is a sequence of cutoff values to try on any given instance until solution is reached. In this example, the series of changing cutoff values are 201, 222, 234, 239, 242, 244, 244, ...

On experiments with hard quasigroup completion problems and SAT encodings of planning problems the approach showed a speedup ranging from 57% to 72% over Luby's universal policy. Interestingly, the policy of using fixed cutoff

that is optimal under the (false) assumption that all instances in the ensemble are equally difficult fails catastrophically, because some instances are never solved.

6 Challenges for Problem Encodings

CHALLENGE 8: *Characterize the computational properties of different encodings of a real-world problem domain, and/or give general principles that hold over a range of domains.*

There has been a good amount of work on comparing different SAT encodings. For example, [97, 98] consider different translations of constraint satisfaction problems (CSP) into SAT. A central issue in this work is what kinds of encodings preserve local CSP consistency checking in the SAT encoding, where local processing consists mainly of unit-propagation. By exploiting some key ideas from CSPs, such as m-loosenes [99], one can in fact optimize the SAT encodings [100]. Examples of other work in the area are on encoding planning problems [101, 102] and quasi-group completion problems (a multi-coloring task) [103].

This work shows clearly that encodings have a significant impact on the practical solvability of the underlying problems. Some general lessons have been obtained, but there is still a need for more unifying, domain-independent principles.

CHALLENGE 9: *Find encodings of real-world domains which are robust in the sense that "near models" are actually "near solutions".*

In our work on planning [104], we noticed that assignments that satisfy all but a few of the clauses encoding our planning problems often represented action sequences that were very different from valid plans. This means that there can be a significant practical mismatch between a solver that tries to maximize the number of satisfied clauses (which is the standard approach is SAT solvers) and the search for valid plans. In particular, maximizing the number of satisfied clauses does not lead to nearly valid plans. It would seem that it should be possible to design better SAT encodings. This challenge remains open. For some related work, dealing with the robustness of encodings in general, see [105].

CHALLENGE 10: *Develop a generator for problem instances that have computational properties that are more similar to real-world instances.*

The final challenge is in response to the concern that the random k-SAT formulas that dominated benchmarks in 1997 might begin to drive research in the wrong direction [106]. [107] introduced a generation model based on the quasi-group (or Latin square) completion problem (QCP). The task is to determine if a partially colored square can be completed so that no color is repeated in any row or any column. QCP is an NP-complete problem, and random instances exhibit a peak in problem hardness in the area of the phase transition in the percentage of satisfiable instances generated as the ratio of the number of uncolored cells to the total number of cells is varied. The structure implicit in a QCP problem

is similar to that found in real-world domains, such as scheduling, bandwidth assignment, and experimental design.

In order to measure the performance of incomplete solvers, it is necessary to have benchmark instances that are known to be satisfiable. This requirement is problematic in domains where incomplete methods can solve larger instances than complete methods: it is not possible to use a complete method to filter out the unsatisfiable instances. [103] described a generation model for quasigroup completion problems that are always guaranteed to be satisfiable. Another interesting approach for generating satisfiable instances is based on a translation of problems from cryptography [108].

Structured problem generators have also been created by linking a random generator for some particular domain to a SAT translator. For example, the Blackbox planning system [109] can be used to convert STRIPS planning problems into CNF formulas. The Blackbox distribution included a simple generator for random logistics planning problems, making it easy to generate random SAT problems that have the underlying structure of a planning problem.

Many SAT benchmarks today are encodings of bounded-model checking verification problems [2, 110]. While hundreds of specific problems are available, it would be useful to be able to randomly generate similar problems by the thousands for testing purposes: we hope to encourage the creation of such a tool.

7 Conclusion

The challenges from our original paper provide a useful framework for discussing some of the exciting progress in satisfiability testing in recent years. We expect further developments on extensions to DPLL and randomized systematic search to continue. Much remains to be done, however, towards the challenges on problem encodings, local search for proofs of unsatisfiability, and hybrid methods.

References

1. Bart Selman, Henry A. Kautz, and David A. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 50–54, 1997.
2. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207, Amsterdam, The Netherlands, March 1999.
3. M. N. Velev and R. E. Bryant. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. In *Proc. 38th Design Automation Conference (DAC '01)*, pages 226–231, 2001.
4. P. Bjesse, T. Leonard, and A. Mokkedem. Finding bugs in an alpha microprocessor using satisfiability solvers. In *Proc. 13th Int. Conf. on Computer Aided Verification*, 2001.
5. L. Simon, D. Le Berre, and E. Hirsch. The sat2002 competition, 2002. http://www.satlive.org/SATCompetition/onlinereport.pdf.

6. D. Le Berre and L. Simon. The essentials of the sat'03 competition, 2003. Under review. Draft available at http://www.lri.fr/ simon/contest03/results/.
7. `http://www.cs.washington.edu/homes/kautz/challenge/`.
8. David S. Johnson and Michael A. Trick, editors. *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Disc. Math. and Theor. Computer Science*. AMS, 1996.
9. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1979.
10. C.M. Li and S. Gerard. On the limit of branching rules for hard random unsatisfiable 3-sat. In *Proc. ECAI*, 2000.
11. O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-sat formulae. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI '01)*, 2001.
12. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic phase transitions. *Nature*, 400(8):133–137, 1999.
13. O. Martin, R. Monasson, and R. Zecchina. Statistical mechanics methods and phase transitions in optimization problems. *Theor. Computer Science*, 265, 2001.
14. Phase transitions and algorithmic complexity ipam, July 2002. www.ipam.ucla.edu/programs/ptac2002/ptac2002-schedule.html.
15. Dimitris Achlioptas, Paul Beame, and Michael Molloy. A sharp threshold in proof complexity. In *Proc., 33st Annual ACM Symp. on Theory of Computing*, pages 337–346, Crete, Greece, July 2001.
16. B. Bollobas, C. Borgs, J. T. Chayes, J. Han Kim, and D.B. Wilson. The scaling window of the 2sat transition. *Rand. Struct. Alg.*, 18:301, 2001.
17. Dimitris Achlioptas, Lefteris, M. Kirousis, Evangelos Kranakis, and Danny Krizanc. Rigorous results for (2+p)-sat. *Theor. Comp. Sci.*, 265:109–129, 2001.
18. Olivier Dubois, Rimi Monasson, Bart Selman, and Riccardo Zecchina. Phase transitions in combinatorial problems (special issue). *Theor. Computer Science*, 265, 2001.
19. E. Friedgut. Sharp thresholds of graph properties, and the k-sat problem. *Journal of the American Mathematical Society*, 12:1017–1054, 1999.
20. C. Gomes and B. Selman. Satisfied with physics. (perspective article.). *Science*, 297:784–785, 2002.
21. T. Hogg, B. Huberman, and C. Williams (Eds.). Phase Transitions and Complexity (Special Issue). *Artificial Intelligence*, 81(1–2), 1996.
22. David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions for SAT problems. In *Proc. 10th Natl. Conf. on Artificial Intelligence*, pages 459–465, 1992.
23. Marc Mézard, Giorgio Parisi, and Riccardo Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297:812, 2002.
24. B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Dimacs Ser. in Discr. Math. and Theor. Comp. Science*, volume 26, pages 521–532. AMS, 1996.
25. J.M. Crawford, M.J. Kearns, and R.E. Schapire. The minimal disagreement parity problem as a hard satisfiability problem, 1995. unpublished manuscript.
26. J. Warners and H. van Maaren. A two phase algorithm for solving a class of hard satisability problems. *Operations Research Letters*, 23:81–88, 1999.
27. Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proceedings of the National Conference on Articial Intelligence*, pages 291–296, 2000.

28. P. Morris. The breakout method for escaping from local minima. In *Proc. AAAI-93*, pages 40–45, 1993.
29. Bart Selman and Henry Kautz. An empirical study of greedy local search for satisfiability testing. In *Proc. AAAI-93*, pages 46–51, 1993.
30. Z. Wu and B.W. Wah. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proc.AAAI-99*, pages 673–678, 1999.
31. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.
32. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
33. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7, 1960.
34. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.
35. Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1977.
36. Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johansen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM J. Comput.*, 30(5):1462–1484, 2000.
37. Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near-optimal separation of treelike and general resolution. Technical Report TR00-005, Elec. Colloq. in Comput. Compl., http://www.eccc.uni-trier.de/eccc/, 2000.
38. A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.
39. V. Chvatal and E. Szemeredi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–208., 1988.
40. S. Cook and D. Mitchell. Finding hard instances of the satisfiability problem: a survey. In *DIMACS Series in Discr. Math. and Theoretical Comp. Sci.*, volume 35, pages 1–17. 1997.
41. Roberto J. Bayardo Jr. and Robert C. Schrag. Using CST look-back techniques to solve real-world SAT instances. In *Proceedings, AAAI-97: 14th Natl. Conf. on Art. Intel.*, pages 203–208, 1997.
42. Hantao Zhang. SATO: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction, LNAI*, volume 1249, pages 272–275, July 1997.
43. R. Stallman and G. Sussman. and forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2), 1977.
44. J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.
45. R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.
46. João P. Marques-Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In *Proc. of the International Conference on Computer Aided Design*, pages 220–227, San Jose, CA, November 1996. ACM/IEEE.
47. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of the 38th Design Automation Conference*, pages 530–535, Las Vegas, NV, June 2001. ACM/IEEE.

48. Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proc. of the International Conference on Computer Aided Design*, pages 279–285, San Jose, CA, November 2001. ACM/IEEE.
49. E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solver. In *Proceedings of Design Automation and Test in Europe (DATE) 2002*, pages 142–149, 2002.
50. J.P. Marques-Silva. An overview of backtrack search satisfiability algorithms. In *5th International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 1998.
51. Paul Beame, Henry Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In *Proc. of the 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August 2003.
52. Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proc. 15th Natl. Conf. on Artificial Intelligence (AAAI-98)*, pages 431–438, 1998.
53. Luis Baptista and Joao P. Marques Silva. Using randomization and learning to solve hard real-world instances of satisfiability. In *Prin. and Prac. of Const. Prog.*, pages 489–494, 2000.
54. Paul Beame, Ashish Agarwal, and Henry Kautz. Using problem structure for efficient clause learning. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing*, 2003.
55. Maria Luisa Bonet and Nicola Galesi. A study of proof search algorithms for resolution and polynomial calculus. In *Proc., 40th Annual Symp. on Found. of Comput. Sci.*, pages 422–432, New York, NY, October 1999. IEEE.
56. E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near-optimal separation of treelike and general resolution. Technical Report TR00-005, Electronic Colloquium in Computation Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2000.
57. Paul Beame, Russell Impagliazzo, Toniann Pitassi, and Nathan Segerlind. Memoization and DPLL: Formula caching proof systems. In *Proc., 18th Annual IEEE Conf. on Comput. Complexity*, Aarhus, Denmark, July 2003. To appear.
58. B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22:253–275, 1985.
59. J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proc. Intl. Conf. Principles of Knowledge Representation and Reasoning*, pages 148–159, 1996.
60. F. A. Aloul, I. L. Markov, and K. A. Sakallah. Efficient symmetry breaking for boolean satisfiability. In *Proc. Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, 2003.
61. A. Urquhart. The symmetry rule in propositional logic. *Discrete Applied Mathematics*, 96:177–193, 1999.
62. Paul Beame, Russell Impagliazzo, Toniann Pitassi, and Nathan Segerlind. Memoization and dpll: formula caching proof systems. In *IEEE Conference on Computational Complexity*, 2003.
63. F. Bacchus, S. Dalmao, and T. Pitassi. Dpll with caching: A new algorithm for #sat and bayesian inference, 2003. Technical Report TR03-003, Electronic Coooloquium in Computational Complexity, `http://www.ecc.uni-trier.de/eccc/`.
64. John Hooker. Resolution vs. cutting plane solution of inference problems: Some computational experience. *Operations Research Letter*, 7:1–7, 1988.

65. A.P. Kamath, N.K. Karmarker, K.G. Ramakrishnan, and M.G.C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. In *Proc. Integer Programming and Combinatorial Optimization*, 1990.
66. J.P. Warners E. de Klerk, H. van Maaren. Relaxations of the satisfiability problem using semidefinite programming, 1999.
67. Joachim Walser. Solving linear pseudo-boolean constraint problems with local search. In *Proc. 14th Natl. Conf. on Artificial Intelligence (AAAI-97)*, 1998.
68. F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Generic ilp versus specialized 0-1 ilp: an update. In *Intl. Conf. on Computer Aided Design (ICCAD)*, 2002.
69. Andrew Parkes. Pb-lib: The pseudo-boolean library, 2003.
70. Ryan Williams, Carla Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proc. of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.
71. Wei Wei and Bart Selman. Accelerating random walks. In *Proc. 8th Intl. Conf. on the Princ. and Practice of Constraint Programming (CP-2002)*, 2002.
72. B. W. Wah and Y. Shang. A discrete langrangian-based global- search method for solving satisfiability problems. *J. of Global Optim.*, 12, 1998.
73. B. Selman, H. Levesque, and D. Mitchell. Gsat: A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Art ificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA, 1992. AAAI Press.
74. D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. In *Proceedings of the Seventeenth National Conference on Articial Intelligence (AAAI-2000)*, pages 297–302, 2000.
75. Guilhem Semerjian and Remi Monasson. A study of pure random walk on random satisfiability problems with physical methods. In *Proc. SAT-2003*, 2003.
76. Holger Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proceedings of AAAI-99*, pages 661–666. AAAI Press, 1999.
77. H. Hoos. Stochastic local search — methods, models, applications. PhD Thesis,TU Darmstadt, 1998.
78. J.A. Boyan and A.W. Moore. Learning evaluation functions for global optimization and Boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Articial Intelligence (AAAI-98)*, pages 3–10, 1998.
79. Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.
80. E. A. Hirsch and A. Kojevnikov. Unitwalk: A new sat solver that uses local search guided by unit clause elimination. In *PDMI preprint 9/2001, Steklov Institute of Mathematics at St.Petersburg*, 2001.
81. R. Paturi, P. Pudlak, and F. Zane. Satisfiability coding lemma. In *Proc. of the 38th Annual IEEE Symposium on Foundations of Computer Science, FOCS'97*, pages 566–574, 1997.
82. Xiao Yu Li, Matthias Stallman, and Franc Brglez. Qingting: A local search sat solver using an effective switching strategy and efficient unit propagation. In *Proc. SAT-2003*, 2003.
83. R. I. Brafman. A simplifier for propositional formulas with many binary clauses. In *Proc. IJCAI-2001*, pages 515–520, 2001.
84. F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Proc. SAT-2003*, 2003.
85. R. Ostrowski, E. Grigoire, B. Mazure, and L. Sais. Recovering and exploiting structural knowledge from cnf formulas. In *Proc. of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002), LNCS, Ithaca (N.Y.)*, pages 185–199. Springer, 2002.

86. Djamal Habet, Chu Min Li, Laure Devendeville, and Michel Vasquez. A hybrid approach for sat. In *Proc. of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002), LNCS, Ithaca (N.Y.)*, pages 172–184. Springer, 2002.
87. B. Mazure, L. Sais, and E. Gregoire. Boosting complete techniques thanks to local search methods. In *Proc. Math and AI*, 1996.
88. Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1–2):67–100, 2000.
89. Hubie Chen, Carla Gomes, and Bart Selman. Formal models of heavy-tailed behavior in combinatorial search. In *In Principles and Pratices of Constraint Programming (CP-01)*, 2001.
90. I. Gent and T. Walsh. Easy Problems are Sometimes Hard. *Artificial Intelligence*, 70:335–345, 1993.
91. T. Walsh. Search in a Small World. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.
92. M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Process. Letters*, pages 173–180, 1993.
93. Eric Horvitz, Yongshao Ruan, Carla Gomes, Henry Kautz, Bart Selman, and Max Chickering. A Bayesian approach to tackling hard computational problems. In *Proc. 17th Conc. on Uncertainty in Artificial Intelligence (UAI-2001)*, 2001.
94. Henry Kautz, Eric Horvitz, Yongshao Ruan, Bart Selman, and Carla Gomes. Dynamic randomized restarts: Optimal restart policies with observation. AAAI2002, 2002.
95. Yongshao Ruan, Eric Horvitz, and Henry Kautz. Restart policies with dependence among runs: A dynamic programming approach. In *Principles and Practice of Constraint Programming - CP 2002*, 2002.
96. Yongshao Ruan, Eric Horvitz, and Henry Kautz. Hardness-aware restart policies, 2003. under review.
97. Toby Walsh. Reformulating propositional satisfiability as constraint satisfaction. *Lecture Notes in Computer Science*, 1864, 2000.
98. Steven Prestwich. Local search on sat-encoded csps. In *Proc. SAT-2003*, 2003.
99. Peter van Beek and Rina Dechter. Constraint tightness and looseness versus local and global consistency. *JACM*, 44(4):549–566, 1997.
100. Cristian Bessiere, Emmanuel Hebrard, and Toby Walsh. Local consistencies in sat. In *Proc. SAT-2003*, 2003.
101. Henry A. Kautz, David A. McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proc. of the 5th Intl. Conf. on Princ. of Knowl. Repr. and Reasoning*, pages 374–384, Boston, MA, November 1996.
102. Michael Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-compilation of planning problems. In *IJCAI*, pages 1169–1177, 1997.
103. Dimitris Achlioptas, Carla P. Gomes, Henry A. Kautz, and Bart Selman. Generating satisfiable problem instances. In *AAAI/IAAI*, pages 256–261, 2000.
104. H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the 13th AAAI*, pages 1194–2001, 1996.
105. Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy. Supermodels and robustness. In *AAAI/IAAI*, pages 334–339, 1998.
106. D. Johnson. Experimental analysis of algorithms: The good, the bad, and the ugly, 1996. Invited Lecture, *AAAI-96* , Portland, OR. See also http://www.research.att.com/ dsj/papers/exper.ps.

107. C.P. Gomes and B. Selman. Problem Structure in the Presence of Perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 221–227, New Providence, RI, 1997. AAAI Press.
108. Fabio Massacci. Using walk-SAT and rel-sat for cryptographic key search. In *Proc. IJCAI-99*, pages 290–295.
109. H.A. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proc. of the 16th International Joint Conference on Artificial Intelligence*, pages 318–325, Stockholm, Sweden, August 1999.
110. Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.