Annals of Mathematics and Artificial Intelligence (2005) 43: 1–34

On SAT instance classes and a method for reliable performance experiments with SAT solvers

Franc Brglez, Xiao Yu Li and Matthias F. Stallmann

Department of Computer Science, NC State University, Raleigh, NC 27695, USA

A recent series of experiments with a group of state-of-the-art SAT solvers and several well-defined classes of problem instances reports statistically significant performance variability for the solvers. A systematic analysis of the observed performance data, all openly archived on the Web, reveals distributions which we classify into three broad categories: (1) readily characterized with a simple χ^2-test, (2) requiring more in-depth analysis by a statistician, (3) incomplete, due to time-out limit reached by specific solvers. The first category includes two well-known distributions: normal and exponential; we use simple first-order criteria to decide the second category and label the distributions as near-normal, near-exponential and heavy-tail. We expect that good models for some if not most of these may be found with parameters that fit either generalized gamma, Weibull, or Pareto distributions. Our experiments show that most SAT solvers exhibit either normal or exponential distribution of execution time (*runtime*) on many equivalence classes of problem instances. This finding suggests that the basic mathematical framework for these experiments may well be the same as the one used to test the reliability or *lifetime* of hardware components such as lightbulbs, A/C units, etc. A batch of N *replicated* hardware components represents an equivalence class of N problem instances in SAT, a *controlled* operating environment A represents a SAT solver A, and the *survival function* $\mathcal{R}^A(x)$ (where x represents the *lifetime*) is the complement of the *solvability function* $\mathcal{S}^A(x) = 1 - \mathcal{R}^A(x)$ where x may represent *runtime*, *implications*, *backtracks*, etc. As demonstrated in the paper, a set of unrelated benchmarks or randomly generated SAT instances available today cannot measure the performance of SAT solvers reliably – there is no control on their 'hardness'. However, equivalence class instances as defined in this paper are, in effect, replicated instances of a specific reference instance. The proposed method not only provides a common platform for a systematic study and a *reliable* improvement of deterministic *and* stochastic SAT solvers alike but also supports the introduction and validation of new problem instance classes.

Keywords: satisfiability, conjunctive normal form, equivalence classes, experimental design, exponential and heavy-tail distributions, reliability function

AMS subject classification: 62F03, 62F25, 62P30, 6Q25, 68T20, 68T27, 68U20, 68W05, 94C10

1. Introduction

The propositional satisfiability problem, SAT, is at the core of NP-hard problems and has been studied in the context of automated reasoning, computer-aided design, computer-aided manufacturing, machine vision, databases, robotics, scheduling, inte-

grated circuit design, computer architecture design, computer networking, etc. The Web has become the universal resource to access large and diverse directories of SAT problem instances [43], SAT discussion forums [44], and SAT experiments [42], each with links to SAT-solvers that can be readily downloaded and installed. Up-to-date survey articles on the SAT problem and problem instances are also readily available on the Web, e.g., [13,18,27]. The performance of SAT solvers is being evaluated experimentally either in terms of randomly generated instances of SAT problems, e.g., [9,45], or structured instances, such as the instances from the DIMACS set [48] and the SATPLAN set [30]. Merits of either approach are subject to on-going critique and examination [8,23,24,33,35].

Traditional benchmarking reports with SAT solvers are based on relatively few experiments. Given p problem instances and s solvers, one performs a total of $p \cdot s$ experiments, typically recording a *runtime* cost function. As a measure of solver performance, various statistics are reported, such as sample mean, standard deviation, median, etc. Such an experiment would have statistical significance *if and only if* the p problem instances were of identical or near-identical 'hardness'. This typically is not the case. Experiments are performed on *single instances* of benchmarks, for example hole6, hole7, hole8, hole9, hole10 [42] as shown in figure 1. These particular instances, of increasing size, indeed describe the same (pigeon-hole) principle. However, as the size of the instance increases, its 'hardness' varies too much to be used as a set of single instances to evaluate the average performance of a SAT solver. One cannot tell whether the observed performance variability is induced by the solver or by the lack of control of the 'hardness' of problem instances. Increasing the number of such instances does not improve the reliability of statistical reports since there can be significant variability in reporting the solver performance for each single instance.

In a series of experiments devised by a *skeptic* [5], significant performance variability of state-of-the-art SAT solvers has indeed been demonstrated consistently on well-defined equivalence class instances of known and new SAT benchmarks. In experimental evaluation of algorithms, a skeptic plays a role analogous to that of an adversary in the worst-case complexity analysis. An adversary devises an input that forces an algorithm to perform badly enough to prove a lower bound (see, e.g., [1]). A skeptic, as the name suggests, is neither as malicious nor as rigorous as an adversary. The skeptic's purpose is to force imperfectly designed algorithms to exhibit large variability on input classes that should induce zero or near-zero variability. The notion of the skeptic as defined above has evolved from the related work in [3,4,14,15,20,21,29,46,47].

This paper expands on the work initiated in [5]. In particular, we not only streamline the formulation of equivalence classes and experimental methodology, we also extend the range of experiments and provide new insights to systematic study and improvements to a new generation of SAT solvers. Of particular importance is the insight that the experiments with equivalence class instances and SAT solvers share a common mathematical framework with experiments in component reliability.

Beyond improving reliability of solver comparisons, our approach also enables fair comparisons between SAT solvers in heretofore incomparable categories. Stochas-

Reference benchmark	Runtime (seconds) *chaff*	*sato*
hole6	0.01	0.04
hole7	0.32	0.11
hole8	0.95	5.40
hole9	5.49	6.44
hole10	36.04	69.65
sum	42.81	81.64

The tabulated results report the *runtime* performance of two state-of-the-art SAT solvers, *chaff* [37] and *sato* [51], as posted recently on the Web [42]. This type of format is traditional when reporting on the performance of CAD algorithms in most publications today. The statistics may be 'sum' of individual cost functions or equivalently, its sample mean. Typically, also shown here, benchmarks in the reported list of experiments represent *unrelated problem instances* of benchmarks. There is no experimental report on solver variability with respect to different but closely related problem instances of the same or comparable 'hardness'.

Figure 1. A traditional format of reporting experimental results with CAD algorithms. Here, a total of $5 \cdot 2 = 10$ experiments have been performed. Increasing the number of problem instances will increase the statistical significance of reported statistics *if and only if* the problem instances are of identical or near-identical 'hardness'.

tic search solvers, such as *walksat* [34] and *unitwalk* [22], are usually compared by doing statistical analysis of multiple runs *with the same inputs,* using different starting seeds (see, e.g., [22,25,26]). Deterministic solvers, such as *chaff* [4] and *sato* [51], on the other hand, are compared either on the basis of a single run or multiple runs with incomparable inputs as illustrated in figure 1. Our experimental methodology puts these different categories of solvers on a level playing field. By introducing syntactical transformations of a problem instance we can now generate, for each reference CNF formula, an *equivalence class* of as many instances as we find necessary for statistical significance of each experiment with either a deterministic or a stochastic solver. For the stochastic solver, results are (statistically) the same whether we do multiple runs with different seeds on identical inputs or with the same seed on a *class* of inputs that differ only via syntactical transformations. In the latter case, the same class of inputs can be used to induce a distribution of outcomes for a deterministic solver. In this paper, we analyze results with several state-of-the-art deterministic solvers and only a single stochastic SAT solver *(unitwalk* [22]). For a more comprehensive and up-to-date performance study of stochastic *and* deterministic SAT solvers, see [31].

The paper is organized as follows. Section 2 provides the context for our choice of benchmark instances and a preview of solver performance variability vis-a-vis the case of traditional experiments as reported in figure 1. The formal introduction of equivalence classes is supported by illustrative experiments in section 3. Section 4 formalizes the notion of the *solvability function* which relates closely to the well-known *survival function* in reliability theory. We use the solvability function statistics to differentiate either two SAT solvers applied to instances from the same class or two equivalence classes

when evaluated by the same solver. Section 5 analyzes reports of various experiments, contrasting solver performances on three groups of equivalence classes: (1) based on reference formulas from the 3-SAT set [27]; (2) based on reference formulas from the SATPLAN set [30]; and (3) based on scheduling problems introduced in [5]. For details about the experimental testbed that supports this paper, and new classes of scheduling instances, see [6].

2. Background and motivation

This research was performed in seven phases:

(1) installation of several SAT solvers readily available on the Web (*chaff* [37], *satire* [50], *sato* and *satoL* [51], and *unitwalk* [22]);

(2) installation of several benchmarks from the DIMACS set [48], the SATPLAN set [30], and miscellaneous instances, including queen problems packaged with the *sato* and *satoL* solvers [51];

(3) implementation of a solver prototype of the vanilla DPLL algorithm [11,12] with random/lexical selection of decision variables (*dp0_nat* and *dp0_lex*, based on pseudocode in [52]);

(4) implementation of a generator of scheduling problem instances;

(5) implementation of equivalence class generators and class instances;

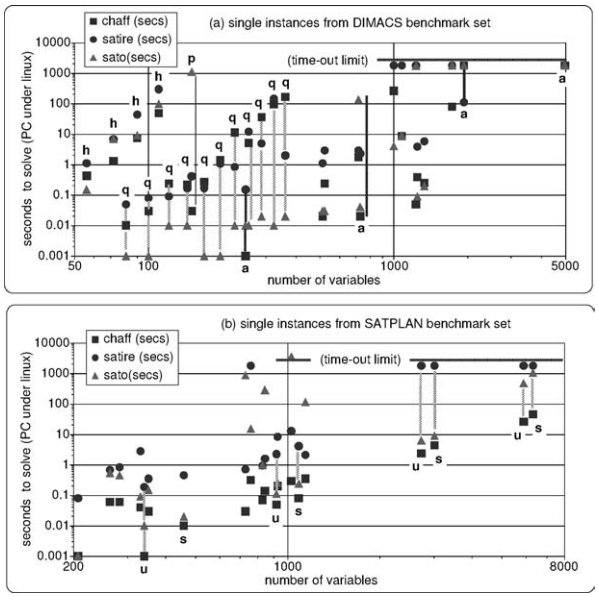
(6) implementation of various components of the experimental test bed to plan, initiate, post-process, and archive the results of experiments on the Web;

(7) a series of experiments and related case study analyses.

Preliminary results of this research were reported in [5]; for an update on the test bed, see [6].

To establish a context for our experiments, we strictly replicate earlier experiments of others, recording results in terms of the traditional method in figure 2. Rather than using a table, we present results of these experiments more compactly in figure 2. This representation also assists in visualizing the presence of asymptotic trends, grouping benchmarks into *families*, such as hole (marked with 'h'), queen (marked with 'q'), hanoi (marked with 'a'), bw_large_u (unsatisfiable, marked with 'u'), and bw_large_s (satisfiable, marked with 's'). A 'traditional' inspection of graphs in figure 2 suggests the following:

- For the hole family, *satire* appears out-ranked by the other two solvers; there is a cross-over of *chaff and sato*.
- For the queen family, *sato* appears to significantly out-rank the other two solvers; there are cross-overs of *chaff and satire*.
- For the hanoi family, *satire* is the only solver that does not time out on the instance of hanoi5 (1931 variables), while all three solvers time out on the instance of hanoi6 (4968 variables).



Trends can be readily observed in the graphical summaries of the *single-instance* experiments with some of the benchmarks from the DIMACS [48] and SATPLAN set [30]. Values of *runtime* are reported by each solver for instances marked as 'a' (hanoi), 'h' (hole), 'p' (pret), 'q' (queen) (DIMACS graph) and 'u' (bw_large_u) and 's' (bw_large_s) (SATPLAN graph).
Variability between solvers is apparent from these graphs, however nothing reported here (or in table 1) suggests the intrinsic and statistically significant variability of each solver alone – a theme explored in this paper.

Figure 2. SAT experiments on *single instances* of mostly unrelated benchmarks.

- For the bw_large_u family, *chaff appears* to out-rank the other two solvers, and *satire* times out for the last two instances (2729 and 5886 variables).

Given the method by which this data has been generated, such visual observations can be misleading. For example, the more elaborate experiments proposed in this paper demonstrate that there is no 'cross-over' in the average performance of *chaff* versus *sato* for the hole family; the performance of the latter is consistently much better for classes derived from all instance sizes. For a preview of a summary that replaces the traditional experimental report in figure 1, see the experiment description and results for the largest instance of the hole formula in figure 3. For each formula in figure 1 we replicate the experiment on $N = 32$ instances from the respective equivalence class. The 'hardness' of problem instances is guaranteed to be the same as that of the reference formula by construction, as described in the next section. Since these are our initial experiments on the subject, we prefer to work with samples of size $N > 30$. For example, the t-statistic has near-normal distribution when the number of samples exceeds 30 [2] and statistical reports based on 32 or more experiments will have more accurate confidence intervals of the mean, etc.

Judging the performance of two solvers on the basis of a single instance as shown in figure 1 is clearly misleading. The single-instance results are equivalent to our experiments reported under the column *initV* in figure 3 (performed on a 266 MHz workstation running under Linux). The *runtime* values under the column *initV* give the *appearance*

To assess solver variability with respect to problem instances of *identical 'hardness'*, we introduce an *equivalence class* of N problem instances for each *reference instance* such as those introduced in figure 1. Choosing $N \geqslant 32$ for statistical significance, we now perform at least $5 \cdot (1 + 32) \cdot 2 = 330$ experiments, a significant increase over $5 \cdot 2 = 10$ experiments reported in figure 1. We summarize various statistics for each reference formula class in a standardized table such as shown below. The column *initV* represents the value reported for the reference instance alone and corresponds to the columns reported for single instances in figure 1, with the proportionally larger execution times here due to our slower computational platform. Note however, the significant variability induced by the class instances on each solver. Moreover, the two solvers appear as if they were 'trained' on the reference instance. Both solvers fail 'to learn' that the class instances are of identical difficulty by the very construction of each instance in the class.
A few notes about the format of this table:

(1) The solverID ordering in tables is induced by sorting on *meanV*.

(2) The value of *initV* is included in reported statistics.

(3) The normal or exponential distribution hypotheses are accepted under a χ^2-test [10] at the 5% level of significance. Decisions about the near-normal, the near-exponential, the heavy-tail, and the incomplete distributions are based on criteria defined in the paper.

(4) A pointer to the original data and additional statistics, such as median, confidence intervals of the mean, etc., is available under

http://www.cbl.ncsu.edu/OpenExperiments/SAT/

. . .
. . .

costID = runtime (seconds)							
Class labels: *name* = *hole10*, *type* = *PC*, *size* = 32							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
sato	101	101	180	178	19.2	207	*normal*
chaff	47.0	47.0	473	468	144	838	*normal*
costID = implications							
Class labels: *name* = *hole10*, *type* = *PC*, *size* = 32							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
chaff	4.58e5	4.58e5	1.87e6	1.85e6	3.02e5	2.25e6	*near-normal*
sato	3.46e7	3.42e7	5.36e7	4.93e7	8.65e6	5.94e7	*near-normal*

Figure 3. A summary of significant statistics observed after experiments with several SAT solvers on instances from the *same* equivalence class. The format of this table is used throughout this paper to report results of all our experiments. Each table is generated automatically by post-processing tabular data posted on the Web. To conserve space in the paper, most tables summarize *runtime* results only.

that *chaff* is the faster solver. However, note the statistics in the adjacent columns. It is obvious that for the given class, *sato* not only is the solver with much better mean *run-*

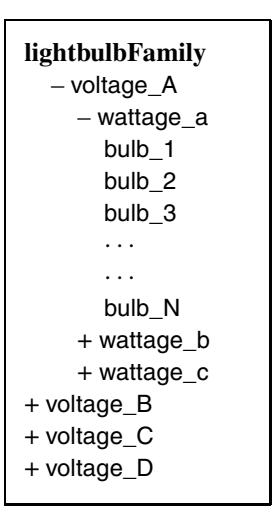
time, its variability is also significantly lower than the one reported for *chaff*. Statistics reported for *implications* indicate a different conclusion: solver *chaff* reports smaller mean and less variance – implying that its implementation may not be as efficient (on this class of problems) as that of *sato*.

3. Equivalence classes in CNF

Our recent experiments with a group of SAT solvers and several equivalence classes of problem instances have demonstrated the presence of intrinsic and significant variability in the performance of these solvers [5]. A more elaborate analysis of these results suggests a common mathematical framework with experiments in component reliability. In the latter, we observe the distribution of component *lifetime*; in the former, we observe the distribution of execution time (*runtime*). A *lifetime* distribution for hardware components is frequently found to have an exponential, Weibull, Pareto, or gamma distribution. Detailed analysis of data from our experiments with SAT solvers reveals normal distributions *and* exponential distributions of *runtime* and other related random variables, as well as other distributions similar to the ones observed in reliability applications. We argue that this insight is due to testing of SAT solvers on problem instances from well-defined equivalence classes. We briefly digress with a basic experiment in component reliability to emphasize this critical point.

3.1. Hardware component reliability

Consider the text-book experiment in testing the *lifetime* of lightbulbs as described in figure 4. It is easy to understand that the *lifetime* of a component may depend on its environment; e.g., the *lifetime* of a lightbulb in a projector may be significantly reduced



A reliability experiment with lightbulbs from the *same manufacturing line* may consider the classification schema on the left. First, we separate lightbulbs by *voltage* ratings $\{A, B, C, D\}$. Within each voltage rating, we separate lightbulbs by the *wattage type*, e.g., $\{a, b, c\}$. Each time we perform a lifetime test of any lightbulb class under specific operating conditions (e.g., forced-air ventilation, no ventilation), we install N lightbulbs with a given voltage and wattage type into a standard fixture, place the fixture into the specifically controlled operating environment, and turn on all the lightbulbs. Observations may be made daily to record the number of bulbs that burned out on the day of observation. Calculating the *mean-time-to-failure* T_0 from the observed experimental data, we expect to find a close fit of the lightbulb *survival function* with the exponential distribution with parameter $1/T_0$ [38].

Figure 4. A lightbulb family classification schema and *lifetime* testing.

if the projector fan is not turned on. What may be more surprising is the manifestation of *exponential distribution* of the lightbulb *lifetime*, i.e. the time it takes for the lightbulb to fail. Such distributions have been recorded for components ranging from lightbulbs and transistors [38], to a large number (213) of air-conditioning units in a fleet of 13 Boeing 720 jet airplanes [41], etc.

The most important point about these experiments is the fact that the components in each of the experiments are as identical as the manufacturing process would permit. Any 'mixing' of components, say lightbulbs of different wattage and from different manufacturing lines, would clearly compromise the experiment and any statistical analysis that follows. In our experiments with SAT solvers, we insist on well-defined equivalence classes of SAT problem instances for the very same reasons.

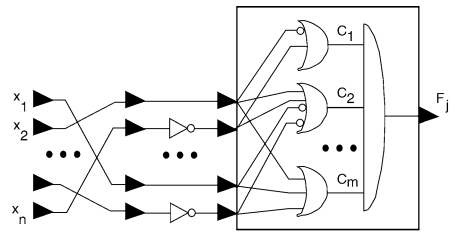
3.2. A CNF formula and its equivalence classes

The hardness of a SAT problem in a CNF formula is encoded in the structure of the CNF representation and the total number of satisfying assignments for the function the formula represents. It is simple to create an *isomorphism* class of CNF formulas, all of the same hardness as a given *reference formula*.

Our approach is based on the notion of equivalence classes of Boolean functions $\mathcal{F}$ as defined by Golomb [16] and illustrated in figure 5(a). Each CNF formula F consists of a set of clauses and each clause is a set of literals. A literal is either a positive integer i, denoting the variable x_i, or $-i$, denoting $\bar{x}_i$, the complement of x_i. While the problem of deciding whether a pair of functions F_i and F_j belongs to $\mathcal{F}$ is at least as hard as graph isomorphism, the problem of generating any size sample of functions in $\mathcal{F}$ is relatively simple. Here, we designate F_0 as the n-variable, m-clause, k-SAT *reference formula* and *re-write* it in accordance with well-defined transformations (variable permutation and complementation), giving rise to four distinct equivalence class types:

- I-class (identity) – variable names are preserved. Clauses and literals within a clause are randomly permuted.
  Class size upper bound: $(k!)^m m!$.
  Property: each solution vector is identical to a reference formula solution vector.
- P-class (permutation) – variables are 'renamed' by subjecting them to a random permutation π; clauses and literals are permuted randomly as in the I-class.
  Class size upper bound: $(k!)^m m!n!$.
  Property: each solution vector is a reference formula solution vector with the permutation π applied.
- C-class (complement) – variable names are preserved; variables in a randomly chosen subset S (each variable is in S with probability 0.5) are complemented; clauses and literals within a clause are randomly permuted.
  Class size upper bound: $(k!)^m m!2^n$.
  Property: each solution vector is a reference formula solution vector with the bits corresponding to S complemented.

(a) On generating a CNF formula instance in an equivalence class.



The dag with inputs $\{x_1, \ldots, x_n\}$, a *permutation/complementation* network, and a 'black box' with output F_j, depicts an instance from an equivalence class $\mathcal{F}$ of Boolean functions, analyzed for their invariants in [16]. Here, the black box represents a *reference CNF formula*.

(b) Samples of CNF formula instances from four equivalence classes.

The 6-variable, 12-clause *reference formula* v06_0004 below has four satisfying assignments. Subject to the four re-writing rules described in the text, each randomly chosen instance from each of the four equivalence classes not only maintains exactly four satisfying assignments, it also preserves the syntactic structure of the reference formula.

Reference formula:	{−1 −2} {−1 −3} {−1 −4} {−1 −5} {−1 −6} {−2 −3} {−2 −4} {−3 −4} {−5 −6} {1 2 3} {4 5 6} {−1 2 −3 4}
solution vectors:	(001001, 001010, 010001, 010010)
I-class formula:	{−1 −5} {−4 −2} {−1 −3} {−1 −2} {−4 −1} {2 3 1} {−2 −3} {5 4 6} {2 −1 4 −3} {−4 −3} {−5 −6} {−1 −6}
solution vectors:	(001001, 001010, 010001, 010010)
transformations:	none, solutions *are* reference formula solutions.
P-class formula:	{−4 −2} {−5 −6} {−3 −1} {−5 −4} {−6 −2} {1 3 5} {−4 −1} {6 2 4} {5 −4 −2 6} {−4 −3} {−5 −2} {−4 −6}
solution vectors:	(011000, 110000, 001001, 100001)
transformations:	variable permutation 462513 is applied to reference formula solutions.
C-class formula:	{4 −5 6} {1 −4} {−6 1} {−2 −3} {−6 5} {−4 −2} {2 −3 4 1} {−1 3 2} {1 −3} {−3 −4} {5 1} {1 −2}
solution vectors:	(101011, 101000, 110011, 110000)
transformations:	variable complementation of (1, 5) is applied to reference formula solutions.
PC-class formula:	{−5 4} {−6 5 2 −4} {2 −3} {−6 2} {2 −5} {4 2} {−6 −5} {−2 6 −4} {−1 2} {−3 −1} {5 3 1} {4 −6}
solution vectors:	(110101, 011101, 110000, 011000)
transformations:	variable permutation 246531 and complementation of (2, 4) is applied to reference solutions.

Figure 5. A method to generate instances of CNF formulas in four equivalence classes.

- PC-class (permutation and complement) – variable names are permuted randomly *and* variables are complemented randomly (as described under the P-class and C-class); clauses and literals within a clause are randomly permuted.

Class size upper bound: $(k!)^m m!n!2^n$.
Property: each solution vector is a reference formula solution vector, subjected to permutation π and with the bits corresponding to S complemented.

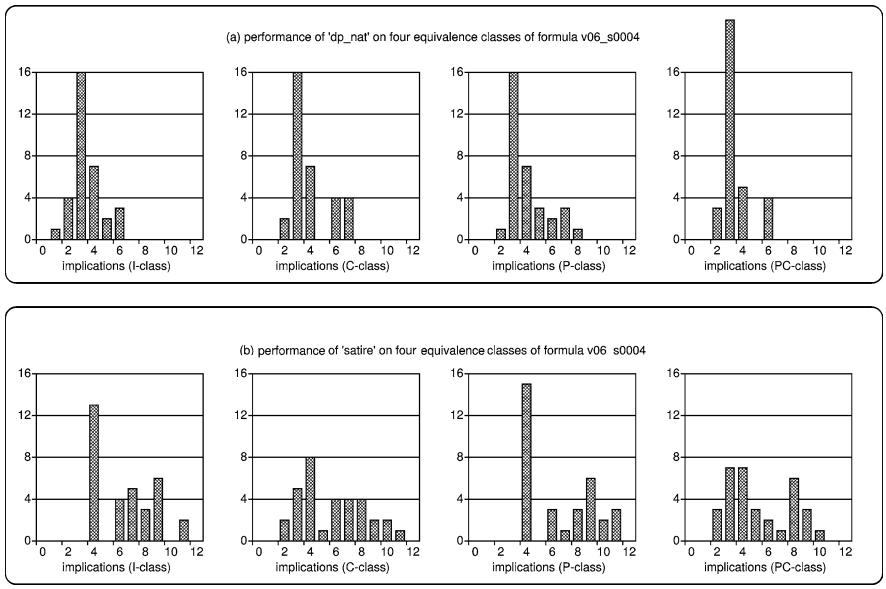
The rewriting neither changes the 'hardness' of the instance nor its syntactic structure: it simply permutes the set of all satisfying solutions of the reference formula. A specific example of the 6-variable, 12-clause formula in figure 5(b) illustrates the simple relationship between the four solution vectors of the reference formula and the solution vectors of each formula in the respective equivalence class.

The classes we define are based on isomorphisms in the following sense. For any formula F having n variables, let $H(F)$ be the n-dimensional hypercube with nodes corresponding to all possible variable assignments (with the usual graph-theoretic interpretation – two nodes are adjacent if their labels differ in one bit), and with nodes whose labels represent satisfying assignments colored. Two formulas F_1 and F_2 are in the same *isomorphism class* if $H(F_1)$ and $H(F_2)$ have an isomorphism [49] that preserves the coloring. Golomb [16] argues equivalence/isomorphism based on the fact that "minimization of logical circuitry can be confined, without loss of generality, to one representative function in each symmetry class".

3.3. SAT solvers and equivalence class instances

A total of five SAT solvers, *chaff*, *satire*, *sato*, *satoL* and *dp0_nat*, were applied to 32 instances in each of the four equivalence classes of the reference formula in figure 6. While part of the same software package, solver *sato* and solver *satoL* represent two different algorithms and data structures. Solver *sato* uses tries and sophisticated branching and backtracking heuristics, *satoL* uses linked lists and more traditional branching and backtracking. Each solver is a variation of the DPLL algorithm [11,12] – they differ only in choice of variable ordering and backtracking strategy. The solver *dp0_nat* orders variables based solely on the input appearance and uses the simplest possible backtracking strategy. The results of all experiments are summarized in figure 6. Since the instances are so small, we only report *implications*. The variability of statistics reported by the five solvers even on this small example is surprising indeed. While the best mean of 3.0 (with standard deviation of 0) is reported for the I- and PC-classes only by *satoL*, the best minimum value of 1 is found by the solver *dp0_nat* only (see the histogram). Results reported by *chaff* are the same for all classes: a mean of 6.0 and standard deviation of 0. Only *dp0_nat* reports mean and standard deviation that is consistently uniform for all classes. This behavior is induced by the solver design described earlier.

We repeated similar experiments on larger class instances; see figure 7 for a representative summary of statistics. Since class instances are of identical 'hardness' by construction, the class-to-class variability exposes erratic behavior of the solvers. These solvers clearly do not recognize the *intrinsic invariant characteristics* of instances in these classes – and until they do, we will observe the behavior as shown. In contrast, experiments with the *unitwalk* solver [22] introduced in section 5 demonstrate a SAT solver whose variability is *always induced uniformly* by instances from any of the four



Equiv. class	*chaff* mean/std	*satire* mean/std	*sato* mean/std	*satoL* mean/std	*dp0_nat* mean/std
I	6.00/0.00	6.31/2.28	8.00/0.00	3.00/0.00	3.44/1.19
C	6.00/0.00	5.63/2.49	6.63/1.91	4.84/2.20	4.03/1.56
P	6.00/0.00	6.47/2.65	7.34/0.79	3.00/0.00	4.09/1.53
PC	6.00/0.00	5.16/2.40	6.44/1.34	5.09/2.10	3.44/1.11

As we demonstrate later in the paper, the number of implications is a measure closely correlated with execution time for DPLL-based SAT solvers.
Even on this small illustrative data set, counting the number of implications clearly differentiates between the five solvers: *chaff* [37], *satire* [50], *sato* and *satoL* [51], *dp0_nat,* our own vanilla implementation of the DPLL algorithm [11,12].
Here, instances of isomorphic 6-variable, 12-clause CNF formulas can induce large variability in performance: from 1 implication/solution to 11 implications/solution. We discovered, on much larger formulas, comparable and much larger max/min performance ratios, both in *runtime* and *implications*.

Figure 6. Experiments with SAT solvers on instances from four equivalence classes.

equivalence classes introduced in this section. To make sure that we evaluate the true average-case performance of any SAT solver, we report our experiments strictly for instances from the PC-class; any exceptions to this rule will be noted explicitly.

Experiments summarized in figure 6 refer to class instances that are too small for detailed analysis of underlying distributions. Motivated by insights from simple experiments in component reliability in figure 4, the next section introduces a simple method to analyze data and distributions from our experiments with SAT solvers.

	ref = queen19_v00361 (class size = 32)		ref = hole10_v00110 (class size = 32)	
Equiv. class	*chaff* mean/stdev	*sato* mean/stdev	*chaff* mean/stdev	*sato* mean/stdev
I	144/9.41	0.020/0.0065	63.3/8.77	102/0.80
C	144/10.8	0.407/0.748	65.5/15.5	169/34.3
P	94.4/42.4	0.023/0.0074	478/82.6	109/1.61
PC	98.4/46.4	0.211/0.577	481/124	181/13.5

Figure 7. A case study of four equivalence classes from two reference formulas. Since class instances are of identical 'hardness' by construction, the class-to-class variability exposes the erratic behavior of the solvers. Only the instances from the PC-class will reliably induce the true average-case performance of *any* SAT solver.

4. Solvability distributions

The introductory experiments in table 3 and figure 6 apply different SAT solvers to problem instances from equivalence classes and imply that the observable parameters of each solver, such as *implications*, *backtracks*, *runtime*, etc., are random variables. In general, the probability model defined by the random variable X is described by its *cumulative distribution function* $F_X(x)$: $F_X(x) = P\{X \leqslant x\}$, where x is in the sample space of X and the range of $F_X(x)$ is the interval [0, 1].

We estimate $F_X(x)$ by performing an experiment that involves N independent trials as follows. Let A denote the SAT solver algorithm applied to N instances *selected randomly* from a given equivalence class of CNF formulas, and let x represent the value of a parameter observed upon solving each instance. Now, define the estimate of $F_X(x)$ as the *solvability function* $\mathcal{S}^A(x)$:

$$\mathcal{S}^A(x) = \frac{1}{N} \cdot (\text{number of observations that are } \leqslant x). \tag{1}$$

Note that $\mathcal{S}^A(x)$ reaches its maximum value of 1.0 only if the solver A returns a decision (satisfiable/unsatisfiable) *for all* of the N instances in the experiment. For some solvers, time-out limit or a backtrack limit may prevent finding solutions to all problem instances.

4.1. Distributions

In the next section, we report experimental results by including not only essential statistics such as sample mean and standard deviation, but also a classification of the underlying distributions. Each distribution induces a characteristic shape of the solvability function; four such shapes are illustrated in figure 8. The criteria we apply in this paper to classify each distribution from the experimental data are both objective and subjective. The objective criteria are based on 'goodness-of-fit' χ^2-test[1] to a normal and an

[1] The χ^2-test compares data from experimental observations with predictions based on the theoretical explanation of the phenomenon under investigation [10]. To evaluate the null hypothesis that the experimental distribution can be represented by the assumed theoretical distribution, a one-tailed test is used as

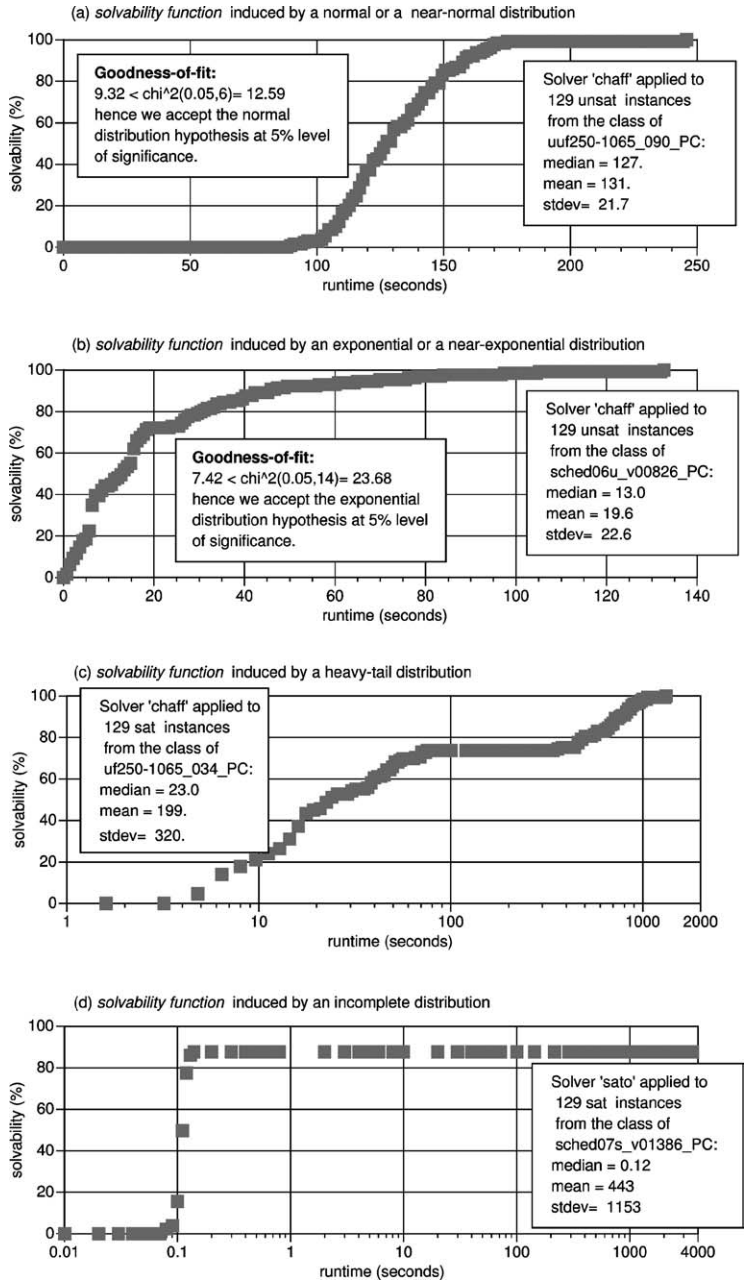


Figure 8. Solvability function types observed during SAT solver experiments.

described by $P(\chi^2 \leqslant \chi^2_{\alpha,\nu}) = 1-\alpha$. If $\chi^2 > X^2_{\alpha,\nu}$, the hypothesis should be rejected. Here, ν designates the degrees-of-freedom (determined from data and the assumed distribution type), and α represents the significance level. Typically, a value of $\alpha = 5\%$ is assumed.

exponential distribution, the subjective criteria are based on less comprehensive evaluations of the distribution statistics and are described below. A separate study is required to re-evaluate these distributions in terms of 'goodness-of-fit' to more complex distributions such as compound exponential, gamma, beta, zeta, Weibull, Pareto, and other distributions (see for example [32,39,40]).

To classify the distributions into types shown in figure 8, we use a simple heuristic that relies on a specific *order* in which we processes the reported statistics, and if required, we also invoke a χ^2-test:

- *an incomplete distribution* is declared when the maximum value of the solvability function is less than 0.95. Such distributions are either due to verification failures, timeout, or both.
- *an impulse distribution* is declared when the standard deviation statistic is 0.0. While not shown in figure 8, we encountered this distribution in some of our experiments, e.g., when the number of backtracks reported by a specific solver is a constant.
- *an exponential distribution* is declared when the hypothesis passes a 'goodness of-fit' χ^2-test (χ^2_{exp}) at 5% level of significance. Note that in this case, median $<$ mean, and mean $\approx$ stdev.
- *a heavy-tail distribution* is declared when indications of a heavy-tail distribution are apparent from the basic statistics of experimental data: median $\ll$ mean, mean $\ll$ stdev. However, fitting a heavy-tail distribution to a formula may require careful consideration of several factors as outlined in a comprehensive overview article on the subject [36].
- *a normal distribution* is declared when hypothesis passes a 'goodness-of-fit' χ^2-test (χ^2_{norm}) at 5% level of significance. Note that in this case, median $\approx$ mean, and mean $> (3 \cdot \text{stdev})$.
- *a near-normal distribution* is declared when

  (1) $\chi^2_{norm} < \chi^2_{\text{exp}}$,

  (2) $\text{stdev}/\text{mean} < 0.55$,

  (3) $0.8 < \text{mean}/\text{median} < 1.2$, and

  (4) $\text{mean} - 2 \cdot \text{stdev} > 0.0$.

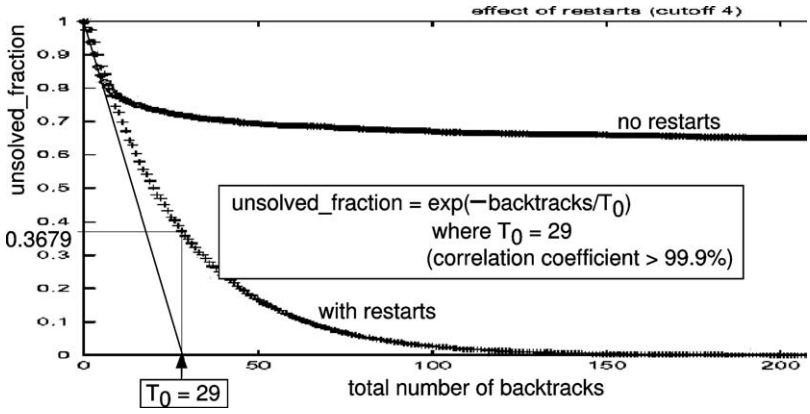
  While this distribution does not pass the normal distribution hypothesis χ^2-test at 5% level of significance, it may well pass such a test for a different level or by fitting to a more complex distribution.
- *a near-exponential distribution* is declared when the preceding test for the near-normal distribution is not accepted. While the distribution does not pass the exponential distribution hypothesis χ^2-test at 5% level of significance, it may well pass such a test for a different level or by fitting to a more complex distribution.

Overall, this simple heuristic has been very effective in providing a concise characterization of the distribution for each of the variables in the experiment (runtime, implications, backtracks, etc.) – in addition to the usual numerical statistics. See [6,7] for illustrative details.

4.2. Related distributions

A study published recently reports a significant number of heavy-tailed distributions "of over two dozen randomly picked instances from both the underconstrained and the critically constrained area" when solved by a specific backtracking algorithm [17]. The study also demonstrates significant improvement in the performance of the algorithm by re-starting the algorithm after four backtracks. A screen-shot of both distributions taken from this paper is shown in figure 9, along with mark-ups introduced by us to demonstrate that the much improved distribution has an excellent fit with an exponential distribution with the mean and the standard deviation of 29 backtracks.

Introducing restarts may work only for algorithms that exhibit heavy-tail cost distribution. The experiments with multiple SAT solvers in section 5 also demonstrate heavy-tail distribution for some solvers on some of the isomorphism class instances. However, for the same instance classes, experiments also show that completely different solvers exhibit normal, near-normal, exponential or near-exponential distribution.



A screen-shot from [17] shows a 'heavy-tail' distribution (no restarts) observed for a specific algorithm and an improved distribution (with restarts) observed for a modified algorithm that implements restarts. Our manual markups demonstrate that the new distribution has an excellent fit with exponential distribution with the mean of 29 backtracks.

Figure 9. An example of an algorithm inducing an exponential solvability function.

4.3. Component reliability

The solvability function as defined in this section is actually the complement of the *survival function* (also known as the *reliability function*), denoted as $\mathcal{R}^A(x) = 1 - \mathcal{S}^A(x)$ where x may represent the *lifetime* of any component in a batch of N components undergoing a test in an environment A [28]. Component reliability is an established field of its own. The fact that our current SAT experiments frequently reveal exponential and near-exponential distributions of observable parameters – similar to the lifetime of components such as lightbulbs, transistors, air-conditioning units – suggests that much may be learned about solvability of combinatorial problems in general, and SAT problem in particular, by studying techniques and experiments in component reliability.

5. Highlights of experimental results

Results of our experiments are organized hierarchically. The initial level consists of 'raw' files saved as direct outputs of each SAT solver and files that contain html- and tab-formatted data, generated by post-processing the raw files. This level is directly accessible on the Web, under a schema updated in [6]. In this paper, we use the format of figure 3 to summarize some of the web-posted results. This format aggregates, across all solvers, a subset of statistics for each equivalence class such as mean, standard deviation, minimum, maximum, and the distribution. Due to space limitation, we report on the statistics for *runtime* only; statistics for *implications*, *backtracks*, etc. are available from the Web. Results are presented in four subsections: (1) Overview of distributions, (2) Random versus isomorphism class instances, (3) Blocks-world versus scheduling class instances, and (4) Queen and hole class instances.

5.1. Overview of distributions

With two notable exceptions all results are for PC classes constructed from single reference instances. The exceptions, already identified in the literature as 'classes' [27], are the random classes, satisfiable (uf250-1065_R) and unsatisfiable (uuf250-1065_R), 100 instances each, with 250 variables and 1065 clauses. The 'hardness' of the instances in the random classes is far from uniform, as we demonstrate by studying up to four specific instances from each class in more detail. Thus, any results about a random class should not be considered on a par with our other results. We elaborate further on this point after giving an overview of our results.

Figure 10 shows a summary of distributions identified with most benchmarks in the single-instance experiments of figure 2. Experiments are reported for four solvers: *chaff* [4], *sato* and *satoL* [51], and *unitwalk* [22]. Experiments with *satire* [50] and *dp0_nat* (a vanilla DPLL solver described in section 3) would time out on larger benchmarks and results are posted on the Web only. All four solvers are applied to ten PC-classes of satisfiable (sat) instances, and only the first three solvers are applied to ten PC-classes of unsatisfiable (unsat) instances (since *unitwalk* is not a complete solver).

(a) observed distributions for PC classes

	chaff	sato	satoL	unitwalk
normal (N)	7	3	5	0
near-normal (nN)	3	6	8	0
exponential (E)	6	4	4	10
near-exponential (nE)	3	2	3	0
heavy-tail (HT)	1	4	0	0
incomplete (IN)	0	1	0	0

(b) sat equivalence classes and distributions

	chaff	sato	satoL	unitwalk
uf250-1065_R	HT	HT	E	HT
uf250-1065_027_PC	E	E	nE	E
uf250-1065_034_PC	HT	E	nE	E
uf250-1065_087_PC	nE	HT	nE	E
bw_large_b_s_PC	N	nN	nE	E
bw_large_c_s_PC	nE	N	nN	E
sched06s_v00828_PC	E	HT	nN	E
sched07s_v01386_PC	E	IN	nN	E
queen14_v00196_PC	E	E	E	E
queen16_v00256_PC	nE	E	E	E
queen19_v00361_PC	nN	HT	E	E

(c) unsat equivalence classes and distributions

	chaff	sato	satoL
uuf250-1065_R	IN	E	N
uuf250-1065_046_PC	N	nN	nN
uuf250-1065_074_PC	nN	nE	N
uuf250-1065_090_PC	N	nE	nN
bw_large_b_u_PC	nN	nN	N
bw_large_c_u_PC	N	N	N
sched06u_v00826_PC	E	HT	nN
sched07u_v01384_PC	E	nN	nN
hole08_v00072_PC	N	nN	N
hole09_v00090_PC	N	nN	N
hole10_v00110_PC	N	N	nN

Figure 10. Summary of *runtime* distributions induced by four SAT solvers on instances from twenty-two equivalence classes of CNF formulas.

The first of the eleven lines in each table of figures 10(b) and 10(c) is devoted to a random class while the three lines below it are for PC-classes built on selected instances of that class. Note that, while PC classes on single satisfiable instances typically induce an exponential or near-exponential distribution for most solvers, the random class of satisfiable instances induces a heavy-tail distribution in three cases out of four. For unsatisfiable instances, the typical PC-class induces a normal or near-normal distribution for most solvers, but the random class induces exponential distribution in two cases out of three.

The remaining seven PC classes come from four sat/unsat families: blocks-world (bw_x, introduced in [30]); scheduling (sched_x, introduced in [5] and expanded in [6]); queens (satisfiable, queen_x, introduced in [51]); and pigeon-hole (unsatisfiable, hole_x, introduced in [48]). There are 32–128 instances in each PC-class, with class size shown in each table reporting the statistics about the experiment. Figure 10 also displays statistics about the observed *runtime* distributions. PC-class instances induce distributions that clearly differentiate the four solvers:

chaff: there are twelve normal or near-normal and seven exponential or near-exponential distributions. For most unsatisfiable classes, distributions are normal or near-normal. There is one heavy-tail distribution for a satisfiable random class.

sato: there are ten normal or near-normal and four exponential distributions. For most unsatisfiable classes, distributions are normal or near-normal. There are five heavy-tail distributions, three for satisfiable classes from random, scheduling, and queens instances, and two for the unsatisfiable scheduling classes. One satisfiable scheduling class has an incomplete distribution.

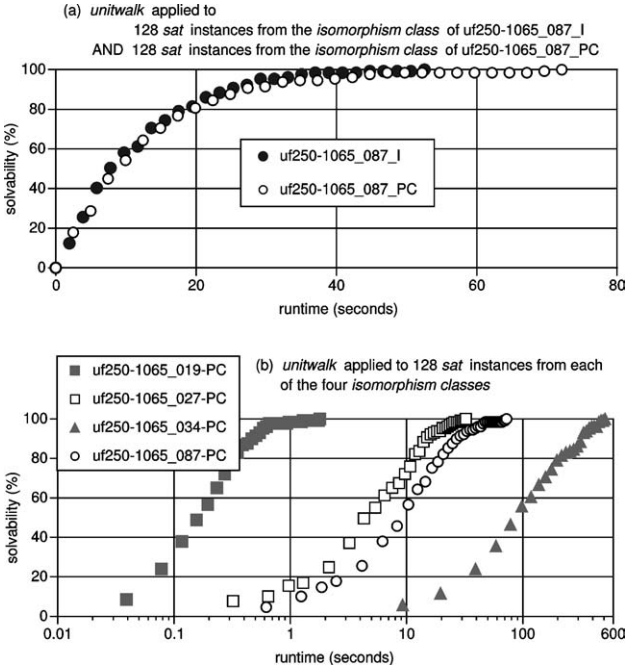
satoL: there are fourteen normal or near-normal and six exponential or near-exponential distributions. For all unsatisfiable instance classes, distributions are normal or near-normal.

unitwalk: all ten satisfiable PC classes induce exponential distributions.

5.2. Random versus isomorphism class instances

We now give a more detailed view of the fundamental difference between instances *generated randomly* and instances *selected randomly* from an isomorphism class. In the first case, only the number of clauses and the number of variables is maintained: a formula is in the class of satisfiable instances whether it has a single or many satisfying solutions. In the second case, the syntactical structure as well as the number of satisfying solutions is maintained for each instance in the class.

Consider the set of satisfying formulas from the random class uf250-1065_R and select four instances: #19, #27, #34, #87 (we explain the rationale for this selection later). See figure 11 for the summary of an illustrative case study on PC-classes of these four formulas and also the I-class and the II-class of formula #87. We introduced the II-class (of *identical instances*) to provide the traditional context for experiments with stochastic solvers such as *unitwalk* [22,25,26]. The associated solvability functions as reported by the *unitwalk* solver illustrate the most important conclusions: (1) for a reference formula, instances in II-, I- and PC-class are equivalent as to their 'hardness' as confirmed by a simple t-test; (2) there is a significant difference in 'hardness' of the four instances as demonstrated by the different solvability function for each class. No statistics test is required for the latter: the significance of this difference is apparent by inspection of the solvability functions in figure 11(b). We contrast the difference between the random instances and the PC-class instances by additional experiments, summarized in a figure and two tables. Figure 12 displays solvability functions reported by four SAT



Four reference instances from the set of 100 satisfiable random 3-SAT formulas in uf250-1065 [27] were chosen in this study: #19, #27, #34, and #87. For each formula, we generated an I- and PC-class of size 128, invoked *unitwalk* solver [22] on each instance, and recorded *runtime* (in seconds) required to find a satisfying solution for each instance. The solver-specific *solvability functions* on the left illustrate the most important conclusions:

(a) for each formula, instances in I- and PC-class are equivalent as to their 'hardness';

(b) there is a significant difference in 'hardness' of the four instances as demonstrated by the different solvability function for each class.

A t-test with an II-class? We define the II-class as the set of *identical instances* read by a *stochastic* local search solver such as *unitwalk* and executed under a different random seed. This method is the backbone of experiments reported in [22,25,26]. In this case study, the mean and the standard deviation of *runtime* (in seconds) reported by each solver on 128 instances in each class are:

uf250-1065_087_II	10.9	9.77
uf250-1065_087_I	12.8	13.0
uf250-1065_087_PC	12.4	12.7

Evaluating t-statistics for all pairs using the formula

$$t = (m_1 - m_2)/(\sigma\sqrt{(1/n_1 + 1/n_2)})$$

where $\sigma = \sqrt{((n_1 - 1)s_1^2 + (n_2 - 1)s_2^2)/(n_1 + n_2 - 2)}$, yields $t = -1.05$ for II-vs-PC, $t = -1.32$ for II-vs-I, and $t = 0.24$ for I-vs-PC. Since $|t| < 1.98$ we have to accept, at 5%-level of significance, that difference between the means between any two populations is not significant, and hence the I-, the PC- and the II-class are equivalent. However, to reliably measure the performance of deterministic as well as stochastic solvers, we use only instances from the PC-class.

Figure 11. A case study of equivalence classes based on four reference formulas from the set of 100 satisfiable random 3-SAT formulas in uf250-1065.

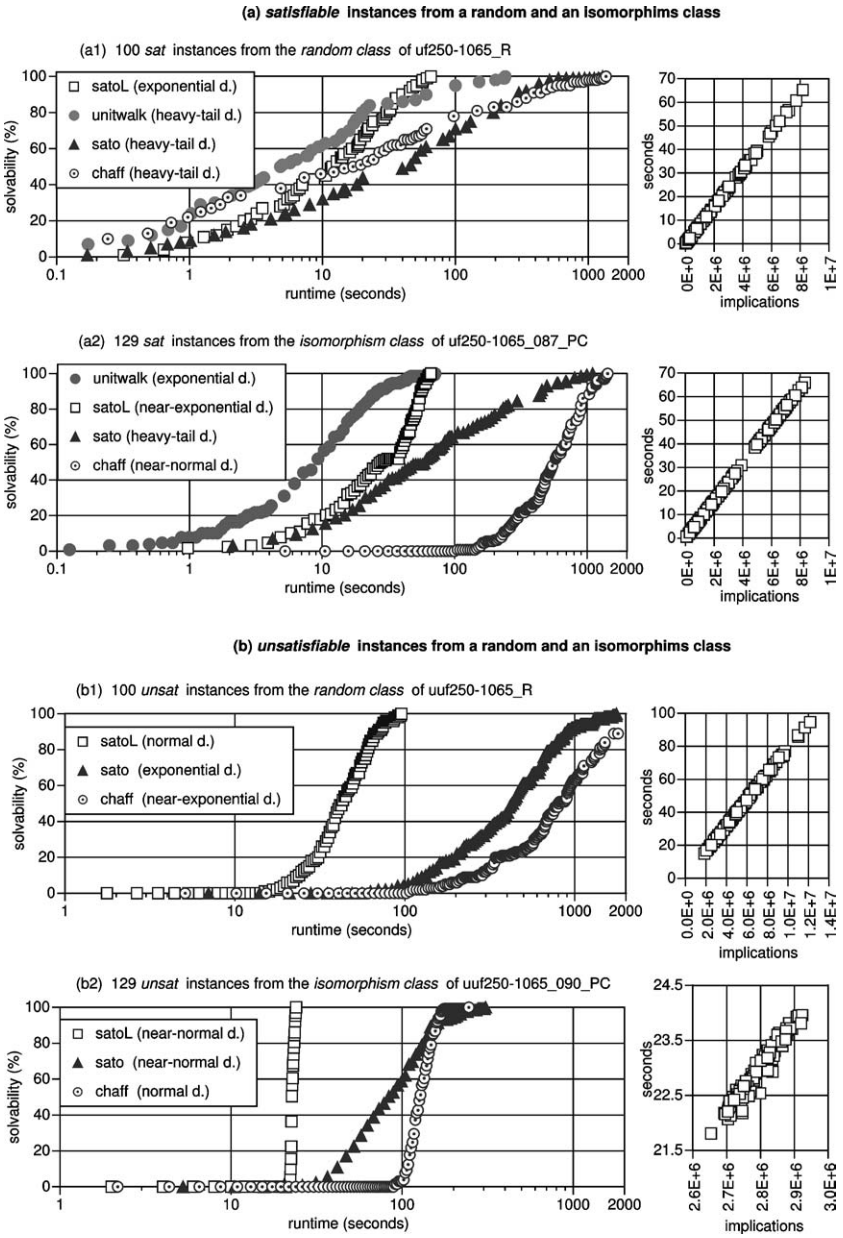


Figure 12. Solvability functions for random and isomorphism class instances.

solvers for (a) the original class uf250-1065_R of 100 randomly generated instances [27] that are satisfiable compared with that of a PC-class of uf250-1065_087_PC, one of the selected satisfiable instances, and (b) the original class uuf250-1065_R of 100 randomly generated instances [27] that are unsatisfiable compared with that of a PC-class of uuf250-1065_90_PC, one of the selected unsatisfiable instances. Complete statistics for

Table 1
Four solver comparisons on random *satisfiable* instances in the set uf250-1065. First, on the original instances from the set of 100. Subsequently, on three classes of isomorphism instances of 128, each class derived from an instance in the set of 100.

costID = runtime (seconds)

Original set: *name* = *uf250-1065*, *type* = *R*, *size* = *100*[a]

solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	–	0.12	12.0	17.0	16.0	65.3	*exponential*
unitwalk	–	0.02	4.85	21.4	43.1	237	*heavy-tailed*
sato	–	0.07	42.7	104	170	1242	*heavy-tailed*
chaff	–	0.01	16.5	115	243	1325	*heavy-tailed*

[a] Except for instances 027, 034, 087, the instances in the table above are different from the ones below. While the number of variables (250) and the number of clauses (1065) is the same for all instances, their functional and their structural characteristics, within the set of 100, are random.

costID = runtime (seconds)

Class labels: *name* = *uf250-1065_027*, *type* = *PC*, *size* = *128*

solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
unitwalk	0.72	0.08	4.42	6.79	6.46	32.1	*exponential*
satoL	46.8	0.22	15.6	26.7	19.7	56.0	*near-exponential*
chaff	115	8.66	51.3	69.8	61.9	356	*exponential*
sato	23.5	0.13	52.9	92.8	106	682	*exponential*

costID = runtime (seconds)

Class labels: *name* = *uf250-1065_034*, *type* = *PC*, *size* = *128*

solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	6.23	0.15	26.2	27.3	16.1	56.5	*near-exponential*
unitwalk	237	1.39	85.5	128	118	501	*exponential*
chaff	550	3.26	23.1	201	321	1315	*heavy-tailed*
sato	222	1.16	136	224	229	1008	*exponential*

costID = runtime (seconds)

Class labels: *name* = *uf250-1065_087*, *type* = *PC*, *size* = *128*

solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
unitwalk	21.2	0.02	9.19	12.4	12.7	72.8	*exponential*
satoL	52.0	0.59	27.4	31.1	19.5	66.0	*near-exponential*
sato	95.7	0.15	60.1	148	212	1100	*heavy-tailed*
chaff	1323	103	573	619	316	1431	*near-exponential*

each of these classes are summarized in tables 1 and 2. We next highlight our decisions and observations based on these summaries.

Table 2
Four solver comparisons on random *unsatisfiable* instances in the set uuf250-1065. First, on the original instances from the set of 100. Subsequently, on three classes of isomorphism instances of 128, each class derived from an instance in the set of 100.

costID = runtime (seconds)							
Class labels: *name* = *uuf250-1065*, *type* = *R*, *size* = *100*[a]							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	–	15.0	42.7	45.4	17.7	94.7	*normal*
sato	–	57.4	428	496	350	1761	*exponential*
chaff	–	103	747	788	423	t'out[b]	*incomplete*

[a] Except for instances 090, 074, 046, the instances in the table above are different from the ones below. While the number of variables (250) and the number of clauses (1065) is the same for all instances, their functional and their structural characteristics, within the set of 100, are random.
[b] Eleven instances time-out at 1800 seconds.

costID = runtime (seconds)							
Class labels: *name* = *uuf250-1065_090*, *type* = *PC*, *size* = *128*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	23.6	21.8	22.7	22.9	0.50	24.0	*near-normal*
sato	59.0	21.6	81.8	94.7	52.3	306	*near-exponential*
chaff	103	89.3	127	130	21.7	245	*normal*
costID = runtime (seconds)							
Class labels: *name* = *uuf250-1065_074*, *type* = *PC*, *size* = *128*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	36.2	34.8	36.6	36.5	0.73	38.2	*normal*
sato	237	80.3	261	304	159	1148	*near-exponential*
chaff	505	400	615	608	90.5	780	*near-normal*
costID = runtime (seconds)							
Class labels: *name* = *uuf250-1065_046*, *type* = *PC*, *size* = *128*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	69.7	65.3	77.6	75.9	5.07	83.2	*near-normal*
sato	1899	435	1134	1180	462	2224	*near-normal*
chaff	3048	1903	2587	2613	329	3417[c]	*normal*

[c] Three instances time-out at 3600 seconds.

Selection of reference formulas. Since our initial experiments were with *chaff*, we examined its *runtime* results on 100 satisfiable formula instances in uf250-1065_R to select a *reference formula* for an isomorphism class. A total of three formulas were selected and a PC-class of 128 instances was generated from each formula:

Table 3
SAT solver comparisons on *satisfiable* and *unsatisfiable* instances in the PC class of the bw_large family.

costID = runtime (seconds)							
Class labels: *name* = *bw_large_b_s*, *type* = *PC*, *size* = *128*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
chaff	0.07	0.02	0.08	0.09	0.04	0.20	*normal*
sato	0.22	0.22	0.78	0.84	0.33	1.91	*near-normal*
satoL	1.62	0.23	1.25	1.33	0.74	3.72	*near-exponential*
unitwalk	0.65	0.04	3.05	4.24	3.85	20.1	*exponential*
costID = runtime (seconds)							
Class labels: *name* = *bw_large_c_s*, *type* = *PC*, *size* = {*32*, *128*[a]}							
solverID	initV	minV	medV	meanV	Dev	maxV	Distribution
chaff	2.49	0.12	3.59	3.95	2.17	13.1	*near-exponential*
sato	9.01	9.01	115	120	36.4	234	*normal*
unitwalk	86.4	12.8	349	497	469	1734[b]	*exponential*
satoL	65.9	65.9	1763	1715	743	3440[c]	*near-normal*
costID = runtime (seconds)							
Class labels: *name* = *bw_large_b_u*, *type* = *PC*, *size* = *32*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
chaff	0.06	0.02	0.05	0.05	0.01	0.07	*near-normal*
sato	0.10	0.10	0.35	0.37	0.11	0.61	*near-normal*
satoL	0.45	0.31	0.45	0.48	0.12	0.70	*normal*
costID = runtime (seconds)							
Class labels: *name* = *bw_large_c_u*, *type* = *PC*, *size* = *32*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
chaff	2.35	1.57	2.62	2.67	0.70	4.58	*normal*
sato	6.25	6.25	47.2	45.3	13.0	68.9	*normal*
satoL	662	466	569	593	86.8	821	*normal*

[a] Only solutions with *chaff* are reported for the class size of 128.
[b] Two instances time-out at 1800 seconds.
[c] Two instances time-out at 3600 seconds.

- uf250-1065_027: chosen as being the closest in *runtime* value (115 seconds) to the mean value reported for *chaff*: for all 100 instances (116 seconds).
- uf250-1065_034: chosen since its *runtime* reported by *chaff* (552 seconds) is closest to $5 \cdot (\text{mean\_value}) = 575$ seconds.
- uf250-1065_087: chosen since its *runtime* reported by *chaff* (1320 seconds) is the maximum value reported.

Table 4
SAT solver comparisons on *satisfiable* and *unsatisfiable* instances in the PC class of the sched family.

costID = runtime (seconds)							
Class labels: *name = sched06s_v00828, type = PC, size = 32*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	0.03	0.02	0.03	0.03	0.01	0.05	*near-normal*
unitwalk	0.37	0.04	0.11	0.20	0.19	0.72	*exponential*
sato	0.04	0.03	0.05	4.39	18.0	91.0	*heavy-tailed*
chaff	6.46	0.29	7.14	17.6	20.3	75.8	*exponential*
costID = runtime (seconds)							
Class labels: *name = sched07s_v01386, type = PC, size = 128*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	0.09	0.05	0.07	0.07	0.01	0.13	*near-normal*
unitwalk	0.15	0.04	0.23	0.40	0.42	2.67	*exponential*
chaff	19.6	0.86	8.37	13.5	11.2	59.8	*exponential*
sato	0.11	0.08	0.11	0.11	0.01	t'out[a]	*incomplete*
costID = runtime(seconds)							
Class labels: *name = sched06u_v00826, type = PC, size = 128*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	0.08	0.08	0.10	0.10	0.01	0.11	*near-normal*
chaff	6.50	0.30	12.9	19.5	22.6	133	*exponential*
sato	0.09	0.08	0.10	28.1	278	3113	*heavy-tailed*
costID = runtime (seconds)							
Class labels: *name = sched07u_v01384, type = PC, size = 32*							
solveriD	initV	minV	medV	meanV	stDev	maxV	Distribution
satoL	0.21	0.21	0.29	0.29	0.02	0.30	*near-normal*
chaff	7.41	0.59	8.33	15.9	13.2	47.8	*exponential*
sato	0.20	0.20	0.25	0.25	0.01	3502[b]	*near-normal*

[a] Sixteen instances time-out at 3600 seconds.
[b] Four instances time-out at 3600 seconds.

We also used *chaff to* select unsatisfiable formula instances from uuf250-1065_R in similar fashion:

- uuf250-1065_090: chosen since its *runtime* reported by *chaff* (103 seconds) is the minimum value reported.
- uuf250-1065_074: chosen since its *runtime* reported by *chaff* (504 seconds) is closest to $5 \cdot (\text{min\_value}) = 515$ seconds.
- uuf250-1065_046: chosen since its *runtime* reported by *chaff* (1800 seconds) is the time-out value.

Summary of sat classes in figure 12 and table 1

- *Random versus PC classes*: With random instances or with single runs on isolated benchmarks we can never tell whether the performance, good or poor, is due to the interaction of the structure/hardness of the instance with the solver or extraneous factors such as input sequence, numbering of variables, etc. The occurrence of the heavy-tail distribution on the random class (figure 12(a1)) for most solvers stands out in contrast to its relative infrequency on PC-classes. We can say with high confidence that the heavy-tail distributions induced by the PC-class of uf250-1065_027 on *chaff* only and the PC-class of uf250-1065_087 on *sato* only are strictly due to deficiencies of the solvers (on these instances). Other solvers perform much better on the same classes. The fact that instances from the random class induce an exponential distribution on *satoL* whereas instances from PC-classes do not (standard deviation is never near the sample mean as it is in case of an exponential distribution) is an indication that the hardness of random instances is far from uniform. To characterize properties of the random class such as uf250-1065_R more reliably, the case study experiment in figure 11 should be extended to 32 or more randomly selected instances from the class.
- *Exponential distribution*: The exponential distribution of *runtime* for *unitwalk* is not induced by the particular isomorphism class instances. Rather, as shown in figure 10, it is the intrinsic property of the algorithm itself – which is expected, given its exclusive reliance on random sampling of the search space. The fact that the random class induces (in our experiments) a heavy-tail distribution on *unitwalk* is due to non-uniform hardness of instances in the class, as already pointed out. On the other hand, instances from a specific PC-class *may* induce an exponential distribution of *runtime* for any other solver: e.g., uf250-1065_027 induces exponential distributions (with different means) on both *chaff* and *sato*.
- *Solver sensitivity to instance features*: According to statistics in table 1, ranking solvers by the reported mean, the random class (topmost table) induces significant differentiation between two solver groups only: *satoL* and *unitwalk* in one, and *sato* and *chaff* in the other. On the other hand, each of the three isomorphism classes induces significant differentiation and also three distinct ranking orders of the four solvers. Analysis (with *sato*) of the three reference instances reveals lower bounds on the number of satisfying solutions to be 51, 3, and 47, respectively, for uf250-1065_027, uf250-1065_034, and uf250-1065_087. Not surprisingly, the *runtime* reported by *unitwalk* is largest for the class of uf250-1065_034_PC, which has the smallest number of known solutions.
- *Runtime versus implications*: Shown on the right-most side in figure 12(a) are near-linear correlations of *runtime* and *implications* for *satoL*. Similar correlations exist for other solvers on these and other class instances (see figure 12(b) and figure 13), indicating that in principle, the performance of SAT solvers may also be measured in terms of platform-independent parameters such as implications, backtracks, etc.

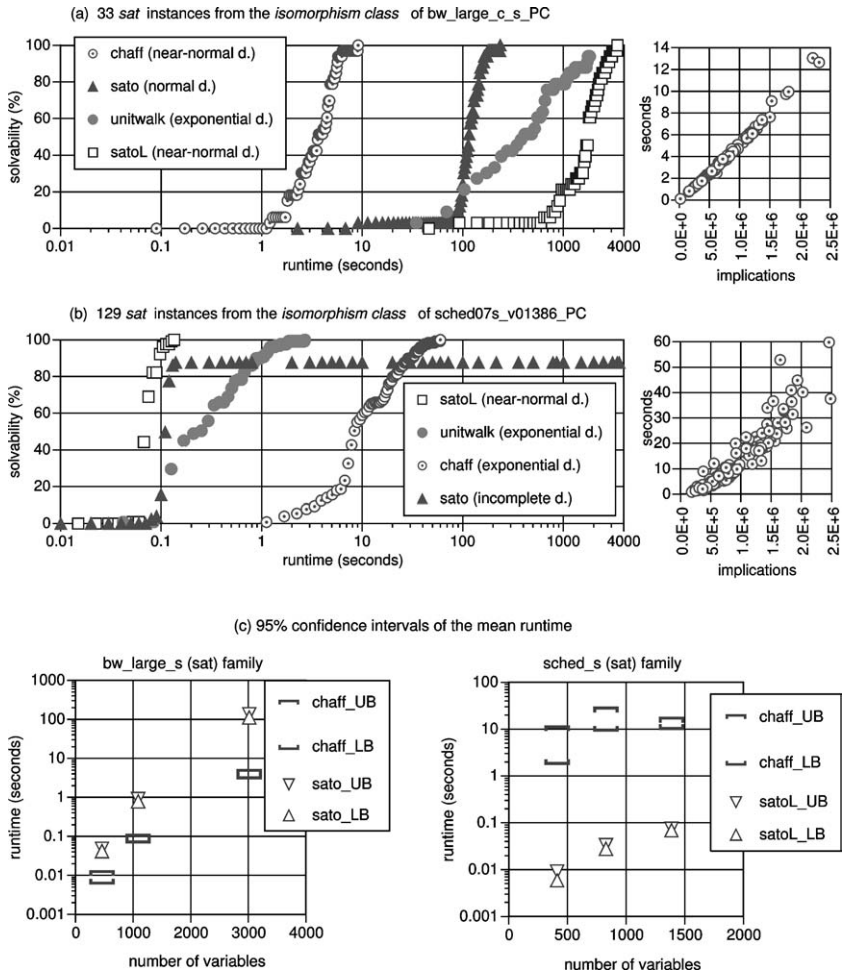


Figure 13. Solvability functions and 95% confidence intervals of mean *runtime* for bw and sched isomorphism class instances.

Summary of unsat classes in figure 12 and table 2.

- *Random versus PC classes*: Compared to satisfiable random class instances in figure 12(a1), solvability functions for the unsatisfiable random class instances in figure 12(b1) have smaller max/min range but show more variability in terms of the mean values (but not the coefficient of variation, a ratio of mean to standard deviation). Moreover, no heavy-tail distributions have been induced on the three solvers by the unsatisfiable random class instances. The report of exponential and near-exponential distributions by *sato* and *chaff* may be an indication that instances differ significantly in 'hardness'. However, this report is off-set by the normal distribution reported by *satoL*. Indeed, the rank order of *satoL*, *sato*, *chaff* remains unchanged when we perform experiments on instances from the three PC-classes. What has

changed are the distributions: now only normal and near-normal distributions are reported.

- *Variability in PC classes*: Compared to satisfiable PC-class instances in figure 12(a2), the variability of mean values *and* standard deviations is significantly reduced. However, the differences between solvers are larger for the case of unsatisfiable PC-class instances. In particular, *satoL* is ahead of other solvers by a wide margin.

5.3. Blocks-world versus scheduling class instances

In contrast to the 3-SAT instances discussed in the preceding subsection, there is not much that is random about the instances discussed in this section. We juxtaposed two different sets of reference formulas, bw_large and sched, because they induce extreme opposite behavior on *chaff* and *satoL*. The sched formulas also induce a heavy-tail-like incomplete distribution on *sato*. Case studies of solver behavior on such test cases may help us to improve SAT algorithms in general.

One set of formulas is based on the satisfiable and the unsatisfiable families of bw_large problems, a subset of the SATPLAN benchmarks which represent a collection of satisfiability problems based on AI planning scenarios developed in [30]. The most difficult of these come from the well-known blocks-world domain in AI (see, e.g., [19]). The other set of formulas is based on the satisfiable and the unsatisfiable families of sched problems which have been created as part of the initial experiments reported in [5]. These formulas are based on unit-length-task scheduling instances. For a description of their construction see [6].

As with 3-SAT instances, the same SAT solvers are now applied to PC-classes of bw_large and sched instances. Summaries of experimental result statistics are shown in tables 3, 4 and figure 13.

- *Solvability functions*: Compare solvability functions induced on four SAT solvers by the class of bw_large_c_s_PC and the class of sched07s_v01386_PC (figure 13). The contrast between *chaff* and *satoL* is striking and we are still looking for an explanation. On the other hand, note the heavy-tail-like incomplete distribution on *sato* by instances from the class sched07s_v01386_PC.
- *Asymptotic behavior*: We study the asymptotic behavior of solvers for a number of PC-classes introduced in [5] and this paper. Much more data than we can present in this paper is posted on the Web [7]. An example of such data collected from the web-based statistical summaries is shown in figure 13(c). Here we grouped together 95% confidence intervals of reported mean values for several solvers, induced by three PC-class instances from the bw_large family and three PC-class instances from the sched family. Note that the 95% confidence intervals of the means are relatively stable for the experiments shown, except for the case of *chaff* when applied to the sched family.
- *Distribution characteristics*: While *chaff* is way ahead of other solvers on PC-classes from bw_large, distributions induced onto solver are normal or near-normal both for

satisfiable and unsatisfiable instances. The exception, as mentioned earlier, is *unitwalk*, which maintains the exponential distribution across all PC-class instances. On the other hand, PC-classes from the sched family induce additional types of distributions, while *satoL* exchanges its ranking with *chaff*. In fact, we observe distributions that are exponential, near-normal, heavy-tail, and incomplete.

5.4. Queen and hole class instances

We conclude this section by summarizing results with two well-known sets of reference formulas: the queen set represents a family of satisfiable formulas [51] that model the n-queens problem, and the hole set represents a family of unsatisfiable formulas [48] that model the n-hole pigeon-hole principle. Formula instances from both families are well-structured, exhibit high degree of symmetry, and can be readily scaled from few variables and clauses to many variables and clauses with a single parameter n. This property in particular makes them ideal candidates to demonstrate the asymptotic behavior of SAT solvers. Comprehensive case study experiments with I-, C-, P-, and PC-classes of each formula for a range of values of n are posted on the Web and also described in [5]. A brief summary of these studies is included as table 5.

- *Distribution characteristics*: The PC-classes of queen problems induce exponential, near-exponential, near-normal, and heavy-tail distributions. The exponential distribution is expected for *unitwalk*. What is surprising is the report about exponential and also heavy-tail distribution for *sato* whose data structure was designed to deal with queen-like problem structures. However, despite the large standard deviation of its heavy-tail distribution, the reported mean is still the best of all solvers.
  The PC-classes of hole problems induce normal or near-normal distribution for all solvers, which is consistent with our observations for PC-classes of other unsatisfiable formulas.
- *Asymptotic runtime characteristics*: The solver ranking order for the PC-classes of queen problems remains the same with increasing n. Solver *sato* is at the top, chaff at the bottom. The performance of *chaff* degrades dramatically with increasing n.
  For the range of n shown, solver ranking for the PC-classes of hole problems remains the same with increasing n. Solver *sato* is at the top, chaff at the bottom. However, the performance of *chaff* is not degrading as rapidly as that of *sato* as the value of n increases – suggesting that there will be a cross-over point for *chaff* and *sato*!

6. Conclusions

We have reported highlights of robust, statistically sound, and repeatable experiments using existing SAT solvers on equivalence classes built from established benchmark problem instances. We have also established a connection between experimental analysis of algorithms and statistical reliability theory, from which we can infer that our approach is likely to generalize to many other settings, particularly the study of heuristics and algorithms for NP-hard problems. Most importantly, our approach uniformly

Table 5
SAT solver comparisons on instances in the PC classes of the queen *satisfiable* and the hole *unsatisfiable* families.

costID = runtime (seconds)							
Class labels: *name* = *queen16_v00256*, *type* = *PC*, *size* = *32*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
sato	0.02	0.01	0.02	0.07	0.09	0.29	*exponential*
unitwalk	0.05	0.00	0.05	0.10	0.09	0.34	*exponential*
satoL	0.10	0.01	0.27	0.24	0.18	0.66	*exponential*
chaff	5.01	0.52	0.95	1.23	0.83	5.01	*near-exponential*
costID = runtime(seconds)							
Class labels: *name* = *queen19_v00361*, *type* = *PC*, *size* = *128*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
sato	0.01	0.01	0.04	0.21	0.57	3.47	*heavy-tailed*
unitwalk	0.32	0.01	0.28	0.39	0.37	1.75	*exponential*
satoL	1.16	0.04	2.94	3.29	2.76	13.3	*exponential*
chaff	34.7	32.1	86.3	97.9	46.6	249	*near-normal*
costID = runtime (seconds)							
Class labels: *name* = *hole09_v00090*, *type* = *PC*, *size* = *32*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
sato	9.11	9.11	17.1	17.0	2.12	21.5	*near-normal*
satoL	13.3	13.3	17.7	17.7	1.13	19.6	*normal*
chaff	7.21	7.21	181	173	48.6	275	*normal*
costID = runtime (seconds)							
Class labels: *name* = *hole10_v00110*, *type* = *PC*, *size* = *32*							
solverID	initV	minV	medV	meanV	stDev	maxV	Distribution
sato	101	101	180	178	19.2	207	*normal*
satoL	151	151	217	216	13.4	231	*near-normal*
chaff	47.0	47.0	473	468	144	838	*normal*

supports reliable performance comparisons between deterministic as well as stochastic SAT solvers.

Some of our findings are surprising, some merit further investigation, and some are easily explained, but the work we have done undoubtedly gives a clearer and more complete picture of the performance of the solvers than any previous experimental study. As a result, we have built a solid foundation for future work on improved solvers, on the creation of challenging problem instances, on the development of hardness-equivalent instance classes, and on accessible software environments for experimental SAT research.

There is clearly no simple answer to the question "what is the best solver?" But the performance of specific solvers on specific classes derived from specific benchmarks can lead to the gathering of some of the best ideas in solver technology on the one hand, or the remedy of identifiable deficiencies on the other. One solver in particular, *unitwalk*, because of its predictable performance, exponential distribution of runtime on all PC classes, has already enabled us to develop a faster solver and to make educated guesses about relative difficulty or number of satisfying solutions for various benchmarks [6,31].

Many open problems remain. Among these are the following.

- The DPLL-based solvers in our study incorporate strategic enhancements that guide (a) the choice of branching variable, (b) the choice of branch to explore first, and/or (c) the backtracking method. The particular mixture used appears to have been guided to some extent by single-instance benchmark experiments. It would be useful to study some of these tricks in isolation – the extent to which each specific modification leads to improvement and on what classes of problem instances.
- The main disadvantage of *unitwalk* is that it is not complete; it cannot report with certainty that a formula is unsatisfiable. Can it be combined with a complementary complete solver so that the latter will quickly determine unsatisfiability after unitwalk has been run for "long enough" and failed to find a satisfying assignment?
- We anticipate a new generation of solvers whose distribution (of any reasonable performance measure such as *runtime)* on any isomorphism class resembles that on the identity class and whose overall performance is as good as or better than current solvers. When such solvers are available, it will make sense to categorize the hardness of problem instances in a more formal way. Can this be done so that instance classes of equivalent hardness include instances that are not isomorphic in our current sense while still retaining all of the statistical properties we have observed? Would there be any way to define a partial ordering of instances based on hardness or would any such characterization be solver-dependent?
- We have already applied similar techniques (classes of "equivalent" problem instances to induce algorithm performance variability) in other problem domains [3, 4,14,15,20,21,29,46,47]. To what extent do the statistical observations we have made carry over into algorithms for other NP-complete decision problems? What about heuristics that find good but suboptimal solutions to optimization problems where the measure of merit is solution quality rather than execution time? Is there a good statistically sound model for characterizing the tradeoff between execution time and solution quality?
- A key byproduct of our work is the large collection of archived data we have accumulated and made available [7], as well as an automated process for replicating our results [6], with or without modification. These resources are fertile ground for research by statisticians, algorithm designers, data visualization experts, and others. The volume of experimental data is already large compared to what a typical statistician encounters, and will grow larger as we continue our work. The data is less noisy than most. And the process by which we create isomorphism classes, execute algo-

rithms, collect, and tabulate results, has been repeated and refined often enough for us to make it available to a larger community. We hope that these resources become an inducement to a variety of new research activities.

Acknowledgements

The experiments, as reported in this paper, could not have taken place without SAT solvers such as *chaff*, *satire*, *sato*, *satoL*, and *unitwalk*. We thank the authors for the ready access and the exceptional ease of installation of these software packages.

We also thank Dr. Jason Osborne from the Department of Statistics (NCSU) for the advice on the likelihood ratio test statistic for exponential distributions. We appreciate the re-assurance that in our experiments with 128 samples, the simpler t-test statistic has sufficient power to resolve the hypothesis of equal population means of two exponential distributions.

The constructive comments from the reviewers and the continued encouragement from Dr. John Franco, University of Cincinnati, have helped in clarifying the concepts and presenting the results of our experiments.

References

[1] S. Baase and A. Van Gelder, *Computer Algorithms*, 3rd edition (Addison-Wesley, Reading, MA, 2000).

[2] G.E.P. Box, W.G. Hunter and J.S. Hunter, *Statistics for Experimenters: An Introduction to Design, Data Analysis and Model Building* (Wiley, 1978).

[3] F. Brglez, Design of experiments to evaluate CAD algorithms: which improvements are due to improved heuristic and which are merely due to chance?, Technical Report 1998-TR@CBL-04-Brglez, CBL, CS Dept., NCSU, Raleigh, NC (April 1998). Also available at `http://www.cbl.ncsu.edu/publications/#1998-TR@CBL-04-Brglez`.

[4] F. Brglez and R. Drechsler, Design of experiments in CAD: Context and new data sets for ISCAS'99, in: *Proc. of IEEE 1999 International Symposium on Circuits and Systems – ISCAS'99* (May 1999). A reprint is accessible from `http://www.cbl.ncsu.edu/publications/#I999-ISCAS-Brglez`.

[5] F. Brglez, X.Y. Li and M. Stallmann, The role of a skeptic agent in testing and benchmarking of SAT algorithms, in: *Proc. of Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, `http://www.cbl.ncsu.edu/publications/` (May 2002).

[6] F. Brglez, M.F. Stallmann and X.Y. Li, SATbed – An environment for reliable performance experiments with SAT instance classes and algorithms, in: *Proc. of SAT 2003, Sixth International Symposium on the Theory and Applications of Satisfiability Testing*, Portofino, Italy, ed. S.M. Ligure (May 5–8, 2003). A revised version available at `http://www.cbl.ncsu.edu/publications/`.

[7] F. Brglez, M. Stallmann and X.Y. Li, SATbed home page: A tutorial, a user guide, a software archive, archives of SAT instance classes and experimental results, `http://www.cbl.ncsu.edu/OpenExperiments/SAT/` (2003).

[8] C. Coarfa, D.D. Demopoulos, A.S.M. Aguirre, D. Subramanian and M.Y. Vardi, Random 3-SAT: The plot thickens, in: *Principles and Practice of Constraint Programming* (2000) pp. 143–159.

[9] S. Cook and D. Mitchell, Finding hard instances of the satisfiability problem: A survey, `http://dream.dai.ed.ac.uk/group/tw/sat/sat-survey3.ps` (1997).

[10] E.L. Crow, F.A. Davis and M.W. Maxfield, *Statistics Manual* (Dover, New York, 1960).
[11] M. Davis, G. Logemann and D. Loveland, A machine program for theorem-proving, Communications of the ACM 5(7) (1962) 394–397.
[12] S. Davis and M. Putnam, A computing procedure for quantification theory, Journal of the Association for Computing Machinery 7(3) (1960) 201–215.
[13] I.P. Gent and T. Walsh, The search for satisfaction, `http://dream.dai.ed.ac.uk/group/tw/sat/sat-survey2.ps`.
[14] D. Ghosh, Generation of tightly controlled equivalence classes for experimental design of heuristics for graph-based NP-hard problems, PhD thesis, Electrical and Computer Engineering, North Carolina State University, Raleigh, NC (May 2000). Also available at `http://www.cbl.ncsu.edu/publications/#2000-Thesis-PhD-Ghosh`.
[15] D. Ghosh and F. Brglez, Equivalence classes of circuit mutants for experimental design, in: *Proc.of Intl. Symp. Circuits and Systems (ISCAS)* (May–June 1999). Also available at `http://www.cbl.ncsu.edu/publications/#1999-ISCAS-Ghosh`.
[16] S.W. Golomb, On the classification of boolean functions, IRE Transactions on Information Theory 5 (1959) 176–186.
[17] C.P. Gomes, B. Selman and N. Crato, Heavy-tailed distributions in combinatorial search, in: *Principles and Practice of Constraint Programming* (1997) pp. 121–135.
[18] J. Gu, P. Purdom, J. Franco and B. Wah, Algorithms for the satisfiability (SAT) problem: A survey, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 35 (1997) pp. 19–152, `http://dream.dai.ed.ac.uk/group/tw/sat/sat-survey.ps`.
[19] N.C. Gupta and D.S. Nau, On the complexity of blocks-world planning, Artificial Intelligence 56(2–3) (1992) 223–254.
[20] J.E. Harlow and F. Brglez, Design of experiments for evaluation of BDD packages using controlled circuit mutations, in: *Proc. of the International Conference on Formal Methods in Computer-Aided Design (FMCAD'98)*, Lecture Notes in Computer Science, Vol. 1522 (Springer, 1998) pp. 64–81. Also available from `http://www.cbl.ncsu.edu/publications/#1998-FMCAD-Harlow`.
[21] J.E. Harlow and F. Brglez, Design of experiments and evaluation of BDD ordering heuristics, International Journal on Software Tools for Technology Transfer (STTT): Special Issue on BDDs (2001).
[22] E. Hirsch and A. Kojevnikov, UnitWalk: A new SAT solver that uses local search guided by unit clause elimination, 2001, PDMI preprint 9/2001, Steklov Institute of Mathematics at St. Petersburg (2001).
[23] J. Hooker, Testing heuristics: We have it all wrong, Journal of Heuristics 1 (1996) 33–42.
[24] J.N. Hooker, Needed: An empirical science of algorithms, Operations Research 42(2) (1994) 201–212.
[25] H.H. Hoos and T. Stützle, Evaluating Las Vegas algorithms – pitfalls and remedies, in: *Proc. of UAI-98* (Morgan Kaufmann, San Mateo, CA, 1998) pp. 238–245.
[26] H.H. Hoos and T. Stützle, Local search algorithms for SAT: An empirical evaluation, Journal of Automated Reasoning 24 (2000).
[27] H.H. Hoos and T. Stützle, SATLIB: An online resource for research on SAT, in: *Proc. of SAT'2000* (IOS Press, 2000) pp. 283–292, `http://www.satlib.org`.
[28] F. Jense, *Electronic Component Reliability: Fundamentals, Modelling, Evaluation, and Assurance* (Wiley, New York, 1996).
[29] N. Kapur, D. Ghosh and F. Brglez, Towards a new benchmarking paradigm in EDA: analysis of equivalence class mutant circuit distributions, in: *Proc. of ACM International Symposium on Physical Design* (April 1997).
[30] H. Kautz, D. McAllester and B. Selman, Encoding plans in propositional logic, in: *KR'96: Principles of Knowledge Representation and Reasoning* (1996) pp. 374–384. The SATPLAN benchmark set is available from `http://sat.inesc.pt/benchmarks/cnf/satplan/`.

[31] X.Y. Li, M.F. Stallmann and F. Brglez, QingTing: A local search SAT solver using an effective switching strategy and an efficient unit propagation, in: *Proc. of 2003-SAT Issue on Satisfiability Testing,* Lecture Notes in Computer Science, Vol. 2919 (2003) pp. 53–68. A significant revision of the paper published in *Proc. of Sixth International Symposium on the Theory and Applications of Satisfiability Testing*, Portofino, Italy, ed. S.M. Ligure (May 5–8, 2003). Available at `http://www.cbl.ncsu.edu/publications/`.

[32] H. Lilliefors, On the Kolmogorov–Smirnov test for the exponential distribution with mean unknown, Journal of the American Statistical Association 64 (1969) 387–389.

[33] J.P. Marques-Silva, On selecting problem instances for evaluating satisfiability algorithms, in: *Proc. of ECAI Workshop on Empirical Methods in Artificial Intelligence (ECAI-EMAI)*, 2000.

[34] D.A. McAllester, B. Selman and H.A. Kautz, Evidence for invariants in local search, in: *Proc. of AAAI/IAAI* (1997) pp. 321–326.

[35] D. Mitchell, A remark on benchmarks and analysis, in: *Proc. of IJCAI-99 Workshop on Empirical AI* (1999).

[36] M. Mitzenmacher, *A Brief History of Generative Models for Power Law and Lognormal Distributions* (Allerton, 2001).

[37] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang and S. Malik, Chaff: Engineering an efficient SAT solver, in: *IEEE/ACM Design Automation Conference (DAC)* (2001). Version 1.0 of Chaff is available from `http://www.ee.princeton.edu/chaff/zchaff/zchaff.2001.2.17.src.tar.gz`.

[38] I. Olkin, L.J. Gleser and C. Derman, *Probability, Models, and Applications* (Macmillan, New York, 1960).

[39] J.A. Osborne and T.A. Severini, Inference for exponential order statistic models based on an integrated likelihood function, Journal of the American Statistical Association 95 (2000) 1220–1228.

[40] J.A. Osborne and T.A. Severini, The Lorenz curve for model assessment in exponential order statistic models, Journal of Statistical Computation and Simulation 72 (2002) 87–97.

[41] F. Prochan, Theoretical explanation of observed decreasing failure rate, Technometrics 5 (1963) 375.

[42] Sat-Ex: The experimentation web site around the satisfiability, `http://www.lri.fr/~simon/satex/satex.php3`.

[43] SATLIB – The Satisfiability Library, `http://www.satlib.org` (2003).

[44] SAT Live! Up-to-date links for the SATisfiability problem, `http://www.satlive.org`.

[45] B. Selman, D.G. Mitchell and H.J. Levesque, Generating hard satisfiability problems, Artificial Intelligence 81(1–2) (1996) 17–29.

[46] M. Stallmann, F. Brglez and D. Ghosh, Heuristics and experimental design for bigraph crossing number minimization, in: *Proc. of the First Workshop on Algorithm Engineering and Experimentation (ALENEX 99)* (January 1999). Also available at `http://www.cbl.ncsu.edu/publications/`.

[47] M. Stallmann, F. Brglez and D. Ghosh, Heuristics, experimental subjects and treatment evaluation in bigraph crossing minimization, Journal on Experimental Algorithmics (2001). Also available at `http://www.cbl.ncsu.edu/publications/#2001-JEA-Stallmann`.

[48] M.A. Trick, Second DIMACS challenge test problems, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26 (1993) pp. 653–657. The SAT benchmark sets are available at `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability`.

[49] D.B. West, *Introduction to Graph Theory* (Prentice-Hall, Englewood Cliffs, NJ, 1996).

[50] J. Whittemore, J. Kim and K. Sakallah, SATIRE: a new incremental satisfiability engine, in: *IEEE/ACM Design Automation Conference (DAC)* (2001). Version 1.0.0 of SATIRE is available from `http://andante.eecs.umich.edu/satire/Satire.tgz`.

[51] H. Zhang, SATO: An efficient propositional proven, in: *Conference on Automated Deduction* (1997) pp. 272–275. Version 3.2 of SATO is available from `ftp://cs.uiowa.edu/pub/hzhang/sato/sato.tar.gz`.

[52] H. Zhang and M.E. Stickel, *Implementing the Davis–Putnam Method* (Kluwer Academic, Dordrecht, 2000).