DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
To appear, 1997.

Finding Hard Instances of the Satisfiability Problem: A Survey

Stephen A. Cook and David G. Mitchell

Abstract. Finding sets of hard instances of propositional satisfiability is of interest for understanding the complexity of SAT, and for experimentally evaluating SAT algorithms. In discussing this we consider the performance of the most popular SAT algorithms on random problems, the theory of average case complexity, the threshold phenomenon, known lower bounds for certain classes of algorithms, and the problem of generating hard instances with solutions.

1. Introduction

Propositional Satisfiability is the problem of determining, for a formula of the propositional calculus, if there is an assignment of truth values to its variables for which that formula evaluates to true. By SAT we mean the problem of propositional satisfiability for formulas in conjunctive normal form (CNF). Although SAT is apparently intractable in the worst case, many instances of the problem are easily solved in practice. Finding ways to generate sets of hard instances is important for understanding the complexity of the problem, and for providing challenging benchmarks for experimental evaluation of algorithms.

The first, and one of the simplest, of the many problems which have been shown to be **NP**-complete, SAT holds a central position in the study of computational complexity. As the dual of propositional theorem proving, it is amenable to the proof of non-trivial lower bounds based on lengths of proofs. Instances which have only long proofs in certain proof systems are intractable for corresponding classes of algorithms. On the other hand, many practical problems are **NP**-hard and may be transformed efficiently to SAT, or have component problems which can be. Thus, a good SAT algorithm would likely have considerable utility. Since it seems improbable that a polynomial time algorithm will be found, "good" might mean performing well on average, or with high probability, or on a class of "interesting" inputs. The hard inputs for a class of algorithms characterize the limitations of those algorithms, and point up where additional research is needed.

1991 *Mathematics Subject Classification.* Primary 68Q25; Secondary 03B05, 03F20.

The first author was supported by the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre. The second author was supported by the Natural Sciences and Engineering Research Council of Canada.

The remainder of the paper is organized as follows. In section 1.1 we provide definitions and terminology. Since the notion of hard instances must usually be in terms of particular algorithms, section 2 considers some of the most popular and successful SAT algorithms. Section 3 examines the average case performance of these algorithms, and properties of randomly generated formulas, including the "threshold phenomenon". We follow this with a discussion of the theory of average case complexity in section 4. Lower bounds for certain classes of algorithms can be shown based on sets of unsatisfiable instances, whereas finding sets of hard *satisfiable* instances is a challenging problem. These issues are discussed in sections 5 and 6, respectively.

1.1. Terminology and Definitions. Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of n boolean variables. A (partial) *truth assignment* for U is a (partial) function $\mathcal{T} : U \to \{\mathbf{true}, \mathbf{false}\}$. Corresponding to each variable u are two *literals*, u and $\neg u$. A literal u (resp. $\neg u$) is **true** under $\mathcal{T}$ iff $\mathcal{T}(u) = \mathbf{true}$ (resp. $\mathcal{T}(u) = \mathbf{false}$). We call a set of literals a *clause*, and a set or sequence (tuple) of clauses a *formula*. We say variable u is *mentioned* in a clause C if $u \in C$ or $\neg u \in C$. If ϕ is a formula, then $vars(\phi)$ is the set of variables mentioned in ϕ.

Let ϕ be a formula, $U = vars(\phi)$, and C a clause in ϕ. We interpret ϕ as a formula of the propositional calculus in conjunctive normal form (CNF), so that a truth assignment $\mathcal{T}$ for U *satisfies* C iff at least one literal $u \in C$ is **true** under $\mathcal{T}$, and $\mathcal{T}$ *satisfies* ϕ iff it satisfies every clause in ϕ. For brevity we sometimes call a satisfying assignment for ϕ a *solution*. We will write $\phi(u)$ for the result of setting a literal u to **true** and *simplifying*. That is, $\phi(u) =^{def} \{C \mid C \in \phi, \{u, \neg u\} \cap C = \emptyset\} \cup \{C \backslash \neg u \mid C \in \phi, \neg u \in C\}$.

The restriction of SAT to instances where all clauses have length k is denoted k–SAT. Of special interest are 2–SAT and 3–SAT: 3 is the smallest value of k for which k–SAT is **NP**-complete [**Coo71**], while 2–SAT is solvable in linear time [**EIS76, APT79**]. Horn–SAT is the restriction to instances where each clause has at most one unnegated variable. Horn–SAT is solvable in linear time [**DG84**], as are a number of generalizations such as Re-nameable Horn–SAT, the class of formulas that can be converted to Horn merely by *renaming* (reversing the sign) of some variables [**Lew78, Asp80**], and Generalized Horn–SAT, an extension of Horn–SAT based on a nesting property [**CH91, CCH$^+$90**] (see also [**CC92, SAFS95**]).

In what follows, we will use n for the number of variables in a formula, m for the number of clauses and k for the clause "length" (e.g., number of literals). When discussing distributions of formulas, n, m and k will refer to parameters to a distribution, rather than particular formulas, and we trust meaning will be clear from context.

2. Algorithms

A procedure for SAT is *sound* if every input on which it returns yes is satisfiable, and *complete* if in addition it returns yes (in finite time) on every satisfiable input. In practice, we often need a procedure to return a solution when the input is a satisfiable formula – the *search* or *function* version of SAT – and the most popular algorithms do return solutions.

2.1. Complete Methods. Since resolution is refutation complete, a simple method for testing satisfiability is to generate all possible resolvents, and then check if the empty clause has been generated. Davis and Putnam [**DP60**] introduced a method (hereafter called DP60), in which variables are eliminated one-by-one from the formula by, at each step, generating all possible resolvents based on a chosen variable and then deleting all clauses mentioning that variable. Each step generates a sub-problem with one fewer variable, but possibly quadratically more clauses. This is easily seen to be a special case of regular resolution.

DP60 employs two heuristics, the *pure literal rule* and the *unit clause rule*, which together state: If some variable occurs in the current formula in a clause of length 1, or occurs only negated, or occurs only unnegated, then choose such a variable to eliminate next. This is effective because the remaining formula must be smaller than the original. It is not hard to see that DP60 may generate an exponential number of clauses in the general case, but at most quadratically many on any instance of 2–SAT. Careful implementation of the unit clause rule results in linear-time handling of (Generalized/Re-nameable) Horn formulas which are unsatisfiable, but not necessarily those which are satisfiable. Indeed, DP60 seems ill-suited for use on satisfiable formulas, since many resolvents may be generated even when a satisfying assignment can be found easily by other methods.

Davis, Logemann and Loveland [**DLL62**] found that in implementation DP60 generated an unmanageable number of resolvent clauses, and replaced the "elimination rule" with a "splitting rule". In this version, selecting a variable leads to two smaller sub-problems – one for each truth value – instead of a single, possibly large, sub-problem. The resulting procedure (which we will call DPLL) is a backtracking depth-first search through (partial) truth assignments, augmented by the unit clause and pure literal heuristics;

Procedure DPLL(CNF formula: ϕ)
    if ϕ is empty **return** yes.
    else if there is an empty clause in ϕ **return** no.
    else if there is a pure literal u in ϕ **return** DPLL($\phi(u)$).
    else if there is a unit clause $\{u\}$ in ϕ **return** DPLL($\phi(u)$).
    else
(*)   select a variable v mentioned in ϕ.
        if DPLL($\phi(v)$)=yes **then**
            return yes.
        else
            return DPLL($\phi(\neg v)$).
        end
    end

In the literature, both DP60 and DPLL have often been called "the Davis-Putnam Procedure". However, the two methods perform quite differently on some problems, and are incomparable with respect to complexity analysis [**Gol79, DR94**].

The variable selection rule (*) may be as simple as choosing the first remaining variable in ϕ, or it may be quite sophisticated. In the original version the rule used is; pick a variable occurring in the first clause of minimal length. The most popular rules currently are variants on ideas suggested in [**DLL62, Gol79**], and described by Pretolani [**Pre93**] as *Moms* variable selection strategy: branch on a variable (or literal) with Maximum number of occurrences in minimum size clauses. A wide range of heuristics can be found in the literature. In particular, Dubois and colleagues [**DABC93**], Crawford and Auton [**CA96**], and Freeman [**Fre94**], have employed careful evaluation of heuristics in developing extremely fast implementations.

Variants of DPLL work quite well in practice, and are probably the most widely known and used SAT testing methods. DPLL also appears to be close to the best we currently know how to do in terms of worst-case performance. The worst case time for this procedure on 3–SAT is $O(2^{0.762n})$, and a very small modification improves it to $O(2^{0.694n})$ [**VanG96**]. The current best bound on the time to decide membership in 3–SAT is $O(2^{0.582n})$ [**Sch96**], using a set of complex refinements to DPLL which seem unlikely to be useful in practice. Complex methods to reduce the size of the search tree often do not lead to corresponding reductions in actual execution time, because of the additional work needed at each node.

Since DPLL may find a solution and exit early, it seems much better suited to use on satisfiable instances than DP60. As in the case of DP60, with careful implementation of the unit clause rule, DPLL handles unsatisfiable (Generalized/Renameable) Horn formulas well, but without enhancement it performs sub-optimally on 2–SAT and satisfiable Horn formulas.

We may call the backtrack tree of DPLL on an unsatisfiable formula a "DPLL proof". Corresponding to every DPLL proof is a tree resolution refutation of size no larger than the DPLL proof tree [**Gol79**, Chapter 3] (sometimes there are also much smaller tree refutations).

Many other complete methods have been devised, including for example Iwama's method for enumerating sets of non-solutions [**Iwa89**], Larrabee's method of trying to extend solutions to the 2–CNF subset of a general formula [**Lar92**], and Gallo and Urbani's use of Horn relaxations [**GU89**]. While some of these perform well on some classes of problems, they have not been as widely studied or tested as DPLL.

2.2. Incomplete Methods. Incomplete methods can be thought of as *model finders*: they cannot prove unsatisfiability, but are often much better than the known complete methods at finding satisfying assignments when they exist. Typically these methods employ some notion of randomized local search. That this was a promising approach for SAT is a relatively recent discovery, made independently by Selman et al [**SLM92**] and Gu [**Gu92**]. Selman's algorithm GSAT was inspired by a very closely related technique developed by Minton and his colleagues for Constraint Satisfaction Problems [**MJPL90**]. An intriguing contribution was the report by Sosic and Gu [**Gu89, SG90**] that this approach yielded very fast solutions to n-queens problems, which had previously been popular as a benchmark (known explicit solutions not-with-standing).

In local search, a cost function is defined over truth assignments such that global minima correspond to satisfying assignments. One guesses a truth assignment as a potential solution, and then tries to improve that guess incrementally by checking truth assignments within a neighborhood of the current one for one with lower cost. In almost all work to date, the initial guess is a random truth assignment, the cost function is the number of clauses not satisfied by the current truth assignment, and the neighborhood is the set of truth assignments at Hamming distance one from the current guess.

The most widely studied of these algorithms is Selman's GSAT [**SLM92**]. At each step of this algorithm the truth assignment of one variable is "flipped". The variable selected is the one which leads to the best neighboring truth assignment.

Ties are broken randomly. A flip is made *even if the best flip increases the number of unsatisfied clauses*. This differs from the classical notion of local search, in which an improvement is made at every step, and search is terminated when no improving step is available. If restricted to improving steps, `GSAT` performs very poorly, and in practice steps which make no change to the score dominate the search, except during an short initial descent phase [**SK93**].

One's intuition might be that such an algorithm will always get stuck in local minima, and the surprise of it's success is that this happens much less often than one might expect. However, it does happen and the procedure may need to be executed many times from different starting points to solve hard instances. Since for incomplete methods there is no *a priori* termination condition in the event of not finding a solution, `GSAT` is parameterized by two bounds; "max-flips" to limit the number flips to do before re-starting with a new initial guess, and "max-tries" to terminate altogether. Gu, in the development of his SAT1 family of local search based algorithms, also employed a variety of techniques for escaping local minima, including backtracking and random flips as well as restarts [**Gu93**]. Determining a general restart criterion would appear to be a tricky problem (see for example [**GW95, HK93**]). However it seems that for some related algorithms, which we discuss next, re-starting is not necessary.

In a similar approach to model finding, which we will call iterative repair, unsatisfied clauses are viewed as "symptoms" of a "flaw" in the truth assignment to be "repaired": If a formula is satisfiable, then any truth assignment which does not satisfy some clause must be wrong on at least one variable mentioned in that clause. This approach is distinguished from from local search in that every iterative repair step is directly addressed to a flaw, but a step may make the natural cost functions arbitrarily worse. A generic iterative repair algorithm is,

Procedure `IR`(CNF formula: ϕ)
    Guess a truth assignment $\mathcal{T}$.
    while $\mathcal{T}$ does not satisfy ϕ.
        if time or iteration limit exceeded **then return** no.
        Select a clause $C \in \phi$ not satisfied by $\mathcal{T}$.
        Modify $\mathcal{T}$ to satisfy C.
    end while
    return yes.
end `IR`

The simplest version of iterative repair is the `Random Walk` algorithm, which repeatedly selects a clause at random from those currently not satisfied, and then flips one of the variables in that clause, chosen at random. In the case of satisfiable instances of 2–SAT, `Random Walk` (indeed any reasonable iterative repair algorithm) may be analyzed as a gambler's ruin, and shown to solve such instances in $O(n^2)$ time on average [**Pap91**]. Although `Random Walk` will always *eventually* solve any satisfiable CNF formula, it is not hard to demonstrate 3–CNF examples with exponential expected time [**Pap94**], and in experimentation the method fails badly on random 3–SAT.

Selman, Kautz and Cohen [**SKC94**], found that a probabilistically greedy version of `Random Walk` performed better than `GSAT` on most types of problems tested. On each iteration this algorithm, called `WalkSAT`, selects an unsatisfied clause at random, and then tosses a coin to determine if the variable to be flipped will be chosen randomly from this clause, or will be the variable in this clause that leads to the greatest number of satisfied clauses when flipped. Another version that was found to work very well tosses a coin at each iteration to decide between taking a `Random Walk` step or a `GSAT` step. In both algorithms, the case of flipping a literal non-greedily serves to prevent getting stuck in local optima, so that restarting becomes less important.

Viewed schematically, these procedures bear considerable resemblance to simulated annealing (SA) [**JAMS91**], and suitably tuned versions of SA do in fact perform comparably with the algorithms just described [**Sp93, BAH+94, SKC94**]. The basic SA algorithm for SAT begins with an initial random truth assignment, and repeats the following step; Pick a random variable, and compute δ, the change in the number of un-satisfied clauses when the variable is flipped. If $\delta \leq 0$ make the flip, otherwise make the flip with probability $e^{-\delta/T}$, where T is the *temperature*, and is usually a decreasing function of the number of steps taken. The low-probability flips which decrease the number of satisfied clauses serve as a mechanism to escape local minima.

Randomized model finding algorithms have been the subject of considerable recent study. While many variants do not work well, a number of versions perform *much* better than `DPLL` and other complete methods on almost all benchmark problems that have been tested. However, they are only useful in applications where the incompleteness is an acceptable trade-off for speed at model finding. Further, each of them is subject to defeat by relatively easily constructed examples – although satisfiable formulas which are hard both for the known deterministic algorithms and the model finding methods appear not to be common.

Almost no progress has been made in the analysis of randomized model-finding algorithms for SAT. As mentioned above, all iterative repair methods work in polynomial expected time for satisfiable instances of 2–SAT and (via a simple rewriting trick), Re-nameable-Horn–SAT. These problems, however, have linear time deterministic algorithms. Koutsoupias and Papadimitriou [**KP92**] proved for a version of randomized 3–SAT that any local search algorithm will almost always find solutions to satisfiable instances with at least $\Omega(n^2)$ clauses, however random formulas with more than a linear number of clauses are almost always unsatisfiable.

Other incomplete algorithms include use of standard mathematical programming and numerical optimization methods. Experiments have been reported, for example, using branch-and-bound, cutting planes, and interior point methods [**Hoo88, HF90, JW90, KKRR90**]. Although many experimenters have done SAT testing with mathematical programming methods, most published reports have used problems which are quite easy for a good implementation of `DPLL` [**ML96**], so the effectiveness of these approaches is not well established in the literature.

3. Average-Case Performance

The theory of **NP**-completeness is based on *worst-case* complexity. The fact that 3–SAT is **NP**-complete implies, assuming $\mathbf{P} \neq \mathbf{NP}$, merely that any algorithm for 3–SAT must take an infeasible amount of time for infinitely many inputs. To explain the behavior of algorithms in practice, the theory of average-case complexity is more appropriate. For this we need to supply a probability distribution on formulas for each input length. Two families of "random formulas" have been the subject of significant study, one based on fixed clause lengths and the other based on random clause lengths. Formula distributions may be used in analyzing algorithms, in empirically evaluating algorithms, or as a subject in problem complexity.

The first average case analysis of SAT of which we are aware was carried out by Goldberg [**Gol79**] on random clause length formulas. Formulas from the random clause length (or "fixed density") model are constructed in the following way: For each of the m clauses, include each of the $2n$ literals with probability p (where p and m may be functions of n). It follows from Goldberg's work that, for any constant value of p, DPLL solves these formulas in time $O(mn^2)$ on average. This first positive result was put in perspective by Franco and Paull [**FP83**], who showed that it was a consequence of a favorable choice of distribution, rather than favorable properties of DPLL: A constant number of guesses of random truth assignments will find one that satisfies an instance from this family with probability tending to 1 as n grows. (More recently it has been shown that DP60 has linear time average performance on these same formulas [**HTL92**].)

Numerous further analyses of these formula distributions have been published, employing a variety of algorithms which take advantage of different properties which hold in different areas of the parameter space. Deterministic algorithms are now known which solve instances of this family in polynomial average time for all but a vanishingly small part of the parameter space. The formulas not yet known to be solvable efficiently on average occur, roughly, when the expected clause length is a little less than $ln(m)$. See the study by Franco and Swaminathan [**FW97**] for details. See also [**Pur90, Fra86**] for earlier work. It is worth noting that the algorithms involved are quite simple, and easily implemented. For most practical purposes, then, this family of formula distribution must be regarded as easy on average, and not a likely source of hard instances. Experimental study confirms this, as we discuss below.

Franco also found that the fixed-clause length formulas took exponential time on average for DPLL when finding all solutions, and suggested this might be a more interesting distribution for study [**FP83**]. Fixed-clause-length formulas are generated by selecting clauses uniformly at random from the set of all possible (non-trivial) clauses of a given length. We call such sets *random* k–SAT. The empirical performance of a version of DPLL on random 3–SAT was investigated in [**MSL92**] (see Figure 1). When c is small, say less than 3, most instances are are very quickly solved. When c is large, say more than 6, instances are harder than those at small ratios, but only moderately. In the region between these ratios average difficulty is dramatically greater. Also between these ratios, the probability of satisfiability shifts smoothly from near 1 to near 0 (as independently reported earlier in [**SD89**]). It is intriguing that the peak in difficulty occurs near the ratio where about half of the formulas are satisfiable – especially since this is the same region that appears hardest to analyze. The same pattern of hardness was found in [**LT92**], using a substantially different algorithm, and we have conjectured that this general pattern will hold, qualitatively, for all reasonable complete methods. The same pattern is also found for larger values of k, but with the transition at higher ratios, and the peak difficulty for DPLL much greater [**ML96**].

Random clause length formulas have also been studied empirically, (and until recently were the most popular in reported experiments). It is readily apparent that most of these formulas are easy, because they often contain empty clauses, unit clauses and trivial clauses. Thus, experimenters shifted to a model where these

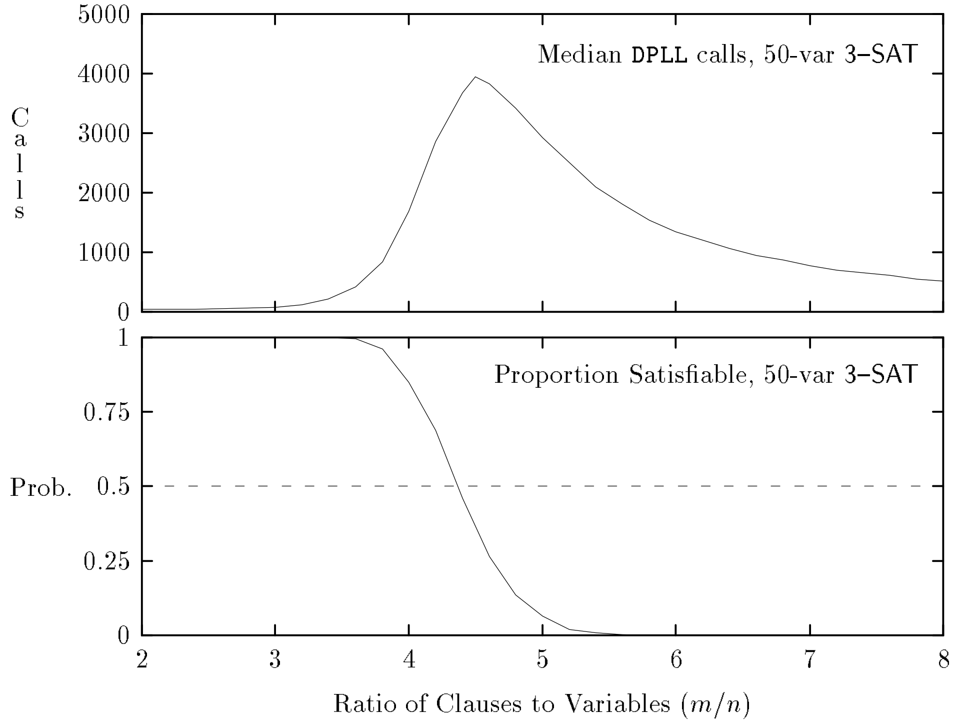

FIGURE 1. Random 3–SAT: DPLL performance and probability of being satisfiable.

three clause types are excluded. In [**MSL92, ML96**], the performance of DPLL on these modified formulas was investigated. When p is made a function of n so that expected clause length remains constant as n is increased, these formulas exhibit the same easy-hard-easy pattern, and the same satisfiable-to-unsatisfiable transition, as the random 3–SAT formulas (as was also observed in [**HF90**]). However, the peak in difficulty at the transition region is much less dramatic with these formulas, and they are very much easier to test than similar sized fixed-clause length formulas. Moreover, it was found that most experiments in the literature had been done at clause-to-variable ratios which were in the easy region, and thus are not very informative about the quality of the algorithms tested. In some cases, formulas with thousands of variables and tens of thousands of clauses can be consistently solved with no more than two or three backtracks each.

Gent and Walsh [**GW94**] investigated rare instances to the left of the "hard region" which appear extremely hard because of excessive run times for DPLL. These appear in both random clause and fixed clause formula families in the "easy" region, where most instances are trivial to show satisfiable. However, such instances seem to be amenable to attack by one or more existing enhancements to backtracking [**SK96, CA96, BS96**].

Thus it appears that the hard formulas in the transition region for random k–SAT are the most useful random formulas of the kind we have so far discussed for evaluating the performance of algorithms. Recently Bayardo and Schrag [**BS96**] have described "literal-regular 3–SAT" instances, which are like random 3–SAT instances, except the number of occurrences of each literal in an instance is forced to be the same. Their experiments indicate that such distributions still exhibit a transition region for the clause/variable ratio, but now shifted to about 3.6 from 4.3, and their random instances in the transition region are significantly harder to test (for DPLL) than are standard random 3–SAT formulas in the transition region.

3.1. The Threshold Conjecture. Experimentally, the probability of an instance of random 3–SAT being satisfiable shifts with the ratio of clauses-to-variables, from being almost 1 with ratios much below 4 to being almost 0 at ratios much above 5. The range of ratios over which this transition occurs becomes smaller as n is increased. A similar pattern holds for other clause lengths, at different ratios. This, together with the common occurrence of threshold phenomena in other random combinatorial structures such as random graphs, suggests the following threshold conjecture: For each k, there is some c^* such that for each fixed value of $c < c^*$, random k–SAT with n variables and cn clauses is satisfiable with probability tending to 1 as $n \to \infty$, and when $c > c^*$, unsatisfiable with probability tending to 1. For the case of random 2–SAT, the conjecture has been shown true, and $c^* = 1$ [**Goe92, CR92**].

In the case of $k = 3$, the current bounds on the location of this threshold, if it exists, are $3.003 < c^* < 4.598$. This lower bound was given by Frieze and Suen [**FS92**] who, extending previous work by Chao and Franco [**CF90**], and Broder, Frieze and Upfal [**BFU93**], gave an algorithm GUCB which, with probability tending to 1, finds a satisfying truth assignment (in polynomial time) for instances of random 3–SAT whenever $c < 3.003$. Suitable variants of DPLL succeed in finding solutions whenever GUCB does, and in the same time, and therefore will almost always find solutions to these instances in polynomial time (although polynomial average time does not necessarily follow).

The upper bound is due to Kirousis, Kranakas and Krizanc [**KKK96**], which improves previous bounds of 4.64 by Dubois and Boufkhad [**DB96**], 4.78 by Kamath et al [**KMPS94**], and 5.19 reported by several authors. The easy 5.19 bound comes from observing that a fixed assignment t satisfies a random 3-literal clause with probability $\frac{7}{8}$ and hence t satisfies a random instance of 3–SAT with probability $(\frac{7}{8})^{cn}$. The expected number of assignments satisfying the random instance is thus $2^n(\frac{7}{8})^{cn}$, and this (and hence the probability that the instance is satisfiable) tends to 0 as $n \to \infty$ for $c > \log_{\frac{8}{7}} 2 = 5.191...$. (The argument also shows that for $c < 5.19$ the expected number of assignments satisfying a random instance grows exponentially with n. This may help explain the success of local search methods on satisfiable instances.)

The improved upper bounds given in [**DB96, KKK96**] are based on the observation that if an instance ϕ is satisfiable, then some assignment t *maximally satisfies* ϕ, meaning t satisfies ϕ, but flipping the value of t on any single variable from **false** to **true** falsifies ϕ. Dubois and Boufkhad obtained the bound $c^* < 4.642$ by estimating the expected number of maximally satisfying assignments, which is significantly smaller than the number of satisfying assignments. Kirousis *et al* independently developed the same method, and by considering double flips obtained the bound $c^* < 4.598$ stated above.

Curiously, direct arguments such as this have so far failed to give interesting lower bounds for the threshold. The known lower bounds result from analyzing specific algorithms, as explained above.

Attempts have also been made to construct a quantitative model of the transition region based on empirical data. This also appears quite challenging (see [**SK96, CA96**]).

3.2. WalkSAT performance near the threshold. Random instances appear to be hardest when generated near the threshold ratio. `DPLL` and its variations are hopelessly slow on many instances of random 3–SAT at this ratio when n is much larger than 400. However, Selman et al [**SKC94**] found that `WalkSAT` solves most of the satisfiable cases for much larger n. With $n = 2000$ variables, the best current estimate of the ratio at which 50% of formulas will be satisfiable is about $c = m/n = 4.24$, or $m = 8480$ clauses. `WalkSAT` solved about 50% of the formulas they generated at this ratio, in about one hour per formula on average. This suggests that `WalkSAT` found satisfying assignments for most of the satisfiable instances attempted in this difficult region, which is a remarkable performance. Although it no doubt failed on some satisfiable instances, we know of no other procedure that would succeed.

4. Average-Case Completeness

The theory of **NP**-completeness tells us that 3–SAT is as hard as any problem in **NP**, in the sense of worst-case complexity. It is reasonable to ask for a comparable result in average-case complexity. Thus we want a distribution on 3–SAT instances for each length l which makes the problem complete in some average-case sense. The theory of average-case completeness tells us that such distributions exist, but there is evidence that none of them is "natural".

This theory was initiated by Levin [**Lev86**] and extensively developed by others (see [**BCGL92**] and [**Gur91**]). Traditionally it requires specification of a *global* distribution to all instances of a problem D. Here we adapt the theory to follow the tradition in average-case analysis of algorithms, where one need only specify a family of *local* distributions, i.e. for each l a distribution μ_l on instances of length l (see [**MS95**]).

If D is a decision problem, then D_l denotes the restriction of D to inputs of length l. A *randomized decision problem* is a sequence $\langle D_l, \mu_l \rangle$, where D is a decision problem and μ_l is a probability distribution on instances of D_l. Such a problem is in **AvP** (average **P**) iff some deterministic algorithm solves it in time *polynomial on average* in the following technical sense: there exist $\epsilon > 0$ and a bound B such that for all l

$$\sum_{|x|=l} \frac{(T(x))^\epsilon}{l} \mu_l(x) < B,$$

where $T(x)$ is the time required by the algorithm on instance x. It follows easily that for any $D \in \mathbf{P}$ and any sequence $\langle \mu_l \rangle$, the sequence $\langle D_l, \mu_l \rangle$ is in **AvP**.

DistNP is the class of randomized decision problems $\langle D_l, \mu_l \rangle$, where $D \in$**NP**, and $\langle \mu_l \rangle$ is uniformly polytime computable.

Completeness for **DistNP** requires a suitable notion of reduction. We say $\langle D_l, \mu_l \rangle$ *reduces via* f to $\langle D'_m, \mu'_m \rangle$ if; 1) $x \in D \Leftrightarrow f(x) \in D'$ for all D-instances x, and 2) f does not map any set of instances with significant μ_l-probability into an instance with very small μ'_m probability. One way to formalize condition 2) is: For every l and every set Y of D_l-instances which are all mapped by f to the same D_m-instance y, the ratio $\mu_l(Y)/\mu'_m(y)$ is bounded by $q(l)$, for some polynomial q. We say that the reduction is *polytime* if f is computable in polytime.

Notice that **DistNP** does not correspond to **NP** in the same sense that **AvP** corresponds to **P**, since a problem in **DistNP** is not necessarily allowed to be in "average **NP**", but must be in **NP** itself. The notion **AvNP** of average **NP** has been defined, and Wang and Belanger [**WB93**] proved that any problem complete for **DistNP** is also complete for the larger class **AvNP**, provided that more general "polytime on average" reductions are allowed, as opposed to the strict polytime reductions defined above.

Levin found a natural distribution for which the tiling problem is complete for **DistNP** with respect to polytime reductions. Thus if tiling with this distribution is in **AvP**, then all of **DistNP** (and in fact **AvNP**) is in **AvP**. A few other such natural examples have been found ([**Gur91**]), but very few compared to the vast number of **NP**-complete problems known. SAT is not among the examples, as we now explain.

Makowsky and Sharell [**MS95**] define the class of *negation-symmetric* distributions on SAT or k–SAT to be all those with the following property: If an instance I' results from an instance I by replacing a particular literal with its negation throughout, then I and I' have equal probability. This includes all the random formulas we have discussed here, and probably all those to be found in the literature. Extending results of Gurevich [**Gur91**] they showed that provided **DEXP**$\neq$**NEXP** (a slightly stronger assumption than **P**$\neq$**NP**, but nonetheless expected to be true) k–SAT is not **DistNP**-complete under deterministic reductions for any of these distributions.

On the other hand, 3–SAT is **NP**-complete, and hence every **NP** problem, including the tiling problem, is reducible to it via some deterministic polytime reduction. In fact, the reduction from tiling to 3–SAT can be made one-one with a polytime inverse, so this reduction applied to the distribution family making tiling **DistNP**-complete will induce a distribution family making 3–SAT **DistNP**-complete. However this induced distribution will not be natural in any sense.

Polytime and average polytime reductions are deterministic reductions. A notion of *randomizing reduction* has been defined, and examples given which are complete for **DistNP** with respect to these reductions but (assuming **DEXP**$\neq$**NEXP**) not with respect to deterministic reductions [**VL88, Gur91**]. Makowsky and Sharell ([**MS95**], p90) suspect that their incompleteness result is due to the restriction to deterministic reductions. Thus a major open problem is to find a natural distribution for which SAT is **DistNP**-complete under randomizing reductions.

5. Lower Bounds

Assuming **P** $\neq$ **NP** no correct algorithm for SAT can operate in polytime, and assuming **AvP** $\neq$ **AvNP** no correct algorithm for SAT with a complete distribution family can operate in average polytime. Without making such assumptions, the best we can hope for with current techniques is lower bound proofs which apply to specific algorithms. In fact, no interesting unconditional lower bounds are known on the performance of the incomplete methods (model finders) described in section 2.2. However strong lower bounds are known on the performance of specific complete methods, but these only apply to their performance on unsatisfiable instances, and result from known lower bounds on the lengths of proofs for specific propositional proof systems.

The computation of either DP60 or DPLL on an unsatisfiable input instance gives rise to a regular resolution proof for that instance whose number of lines is bounded by the number of steps in the computation. Hence the exponential lower bound on regular resolution proofs [**Tse70, Gal77, BA80**] gives rise to worst-case exponential lower bounds on the time required for both DP60 and DPLL, and the exponential lower bounds for general resolution [**Hak85, Urq87**] show that no smarter use of resolution will help.

From our point of view, the strongest lower bound for proof systems is due to Chvatal and Szemeredi [**CS88**], since their bound yields an exponential lower bound on the *average-case* performance of DP60 and DPLL. They show that for each fixed constant c there is $\epsilon > 0$ so that for random 3–SAT with $m = cn$ clauses, the probability that there exists a resolution refutation of size less than $2^{\epsilon n}$ tends to 0 as $n \to \infty$. An immediate corollary is that DP60 and DPLL take exponential time with high probability on random 3–SAT when $c > 4.598$, since random instances are then very likely unsatisfiable (see section 3.1). Notice however that if c is not a constant and exceeds n^2 then sub-linear length proofs almost surely exist [**Fu95, MS95**].

The experimental finding that random 3–SAT with large c is easy is not in conflict with the exponential lower bound for large (constant) ratios, but only shows that at a given n these are much easier than formulas in the transition region. Crawford and Auton studied the empirical scaling behavior of their implementation of a very refined version of DPLL, and found that the average solution cost grew roughly as $2^{n/19.5}$ near the transition region, and something like $2^{n/68}$ at higher ratios [**CA96**].

A slightly stronger proof system than resolution is the cutting plane system. Some work has been done using cutting planes in a SAT tester, for example [**Hoo88**]. Recently, exponential worst-case lower bounds have also been given for cutting plane proofs [**Pud96**], but so far nothing comparable to the average-case resolution lower bound of Chvatal and Szemeredi has been shown.

Exponential lower bounds have not been shown for some more powerful proof systems, such as Frege systems and extended Frege systems. A lower bound for extended Frege systems would yield a similar lower bound for any complete SAT tester whose correctness can be proved using feasibly constructive methods [**Coo75, Kra95**].

6. Hard Satisfiable Instances

As we mentioned, the known unconditional lower bounds apply only to hard unsatisfiable instances, and furthermore existing model finders perform remarkably well on satisfiable instances even near the difficult threshold ratio on random 3–SAT (see section 3.2). It seems that the problem of generating random hard satisfiable instances is related to a traditional problem in cryptography theory. More precisely, Russell Impagliazzo has pointed out that generating hard *solved* instances of 3–SAT is equivalent to computing a one-way function, which in turn is equivalent to generating pseudo-random numbers and private key cryptography [**ILL89, Luby96**]. (It may be easier to generate hard *satisfiable* instances than hard *solved* instances, but we have no insight on this.)

The problem of generating hard solved instances can be explained as follows: Find a polytime function h which takes a string r (representing random bits) to a pair $h(r) = \langle t(r), \phi(r)\rangle$, where the assignment $t(r)$ satisfies the formula $\phi(r)$, and for a random string r it is difficult, given $\phi(r)$, to find *any* satisfying assignment for $\phi(r)$.

The one-way function corresponding to the above h is the map which takes r to $\phi(r)$. This function can be computed in polytime, but it is hard to invert: If given $\phi(r)$ I could find *any* r' mapping to $\phi(r)$ (i.e. $\phi(r') = \phi(r)$) then I have found $t(r')$ which satisfies $\phi(r)$, violating the assumption above.

Conversely, suppose $f : \{0,1\}^* \to \{0,1\}^*$ is a length-preserving one-way function. Then f can be computed in polytime, but for a random x, given $y = f(x)$, it is difficult to find *any* x' such that $f(x') = y$. This f can be used to generate hard solved instances of 3–SAT as follows.

Since f is length preserving and polytime, for each n there is a polysize Boolean circuit C_n which takes the n input bits $x_1, ..., x_n$ representing the string x to the n output bits $y_1, ..., y_n$ representing the string $y = f(x)$. If the circuit has s gates, then we may introduce s new variables $z_1, ..., z_s$, one for each gate, and express the correctness of each gate by the conjunction of at most four 3-literal clauses, which assert that the output of the gate is correct with respect to its inputs. (For each gate computing a circuit output value y_i, we should replace the variable z_j for that gate by the variable y_i.) The conjunction of all such clauses is a formula ϕ_n in the x's, y's, and z's asserting the correctness of the circuit. Now given an n-bit string $a = a_1...a_n$ we can compute $f(a) = b = b_1...b_n$ and we can compute the formula $\phi(a) = \phi_n \wedge y_1' \wedge ... \wedge y_n'$, where y_i' is y_i if $b_i = 1$ and otherwise y_i' is $\neg y_i$. Further, we can compute the satisfying assignment $t(a) = a_1, ..., a_n, b_1, ..., b_n, c_1, ..., c_s$ to the variables $x_1, ..., x_n, y_1, ..., y_n, z_1, ..., z_s$ in $\phi(a)$, where c_j is the value of gate j when the inputs are $a_1, ..., a_n$. Finally, we define $h(a) = \langle t(a), \phi(a)\rangle$. Notice that the formula $\phi(a)$ can be computed from b (without knowledge of a), and if given $\phi(a)$ any satisfying assignment can be found, then this assignment would give rise to a string a' such that $f(a') = b$.

6.1. SAT **challenge.** Several conjectures on the difficulty of number-theoretic functions imply the existence of one-way functions, but the most basic such conjecture is that factoring large integers is difficult. A specific problem supposed to be difficult is: Given the product $M = PQ$ of two random n-bit primes P and Q, find P and Q. A SAT instance would be an encoding of a boolean multiplier circuit computing the known product M from unknown inputs P and Q. Variables are the bits of P and Q (the inputs to the circuit), together with outputs of the gates of the circuit. Clauses assert the correct behavior of the gates, and assert that the outputs of the circuit represent the given value of M. This problem differs from most other benchmarks in that there is essentially a unique satisfying assignment.

Challenge: Report the largest n for which your SAT solver can (consistently) find P and Q within one hour.

Part of the challenge is to find a suitable multiplier circuit: not too complex, and probably not too deep (see for example [**Weg**], section 3.2).

The state of the art for number-theoretic factoring methods seems to be around $n \simeq 200$ bits (about 60 decimal digits for each prime). We believe that the current state for the SAT encoding approach is very much less than this.

References

[APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, March 1979.

[Asp80] Bengt Aspvall. Recognizing disguised NR(1) instances of the satisfiability problem. *Journal of Algorithms*, 1:97–103, 1980.

[BS96] Roberto J. Bayardo, Jr. and Robert Schrag. Using CSP look-back techniques to solve exceptionally hard SAT instances. In *Proc. of the Second Int'l Conf. on Principles and Practice of Constraint Programming, (Lecture Notes in Computer Science 1118)*, 46-60, Springer, 1996.

[BA80] Mordechai Ben-Ari. A simplified proof that regular resolution is exponential. *Information Processing Letters*, 10(2):96–98, 1980.

[BCGL92] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *JCSS*, 44, 193-219, 1992.

[BAH+94] A. Beringer, G. Aschemann, H.H. Hoos, M. Metzger and A. Weiß. GSAT versus Simulated Annealing. In *Proc. ECAI-94*, pages 130–134, 1994.

[BFU93] A. Broder, A. Frieze, and E. Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. In *Proc. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993.

[CA96] J.M. Crawford and L.D. Auton. Experimental results on the cross-over point in random 3-SAT. *Artificial Intelligence*, 81:31–57, 1996.

[CCH+90] Vijaya Chandru, Collette R. Coullard, Peter L. Hammer, Miguel Montañez, and Xiaorong Sun. On renamable horn and generalized horn functions. *Annals of Mathematics and Artificial Intelligence*, pages 33–47, 1990.

[CH91] Vijaya Chandru and John Hooker. Extended Horn sets in propositional logic. *Journal of the ACM*, 38(1): 205–221, 1991.

[CF90] M. Chao and J. Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k satisfiability problem. *Information Sciences*, 51:289–314, 1990.

[CR92] V. Chvátal and B. Reed. Mick gets some (the odds are on his side). In *Proc. of the 33rd IEEE Symposium on the Foundations of Computer Science*, Pittsburgh, 1992.

[CS88] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, 1988.

[CC92] Michele Conforti and Gérard Cornuéjols. A class of logic problems solvable by linear programming. In *Proc. of the 33rd IEEE Symposium on the Foundations of Computer Science*, Pittsburgh, 1992.

[Coo71] Stephen Cook. The complexity of theorem proving procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158, New York, 1971. Association for Computing Machinery.

[Coo75] Stephen Cook. Feasibly constructive proofs and the propositional calculus. In *Proc. 7th Ann. ACM Symp. on Theory of Computing*, 83-97, 1975.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[DR94] R. Dechter and I. Rish. Directional Resolution: The Davis-Putnam Procedure, Revisited. In *Proc. KR-94*, 134–145, 1994.

[DG84] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulas. *Journal of Logic Programming*, 1(3):267–284, 1984.

[DABC93] O. Dubois, P. Andre, Y. Boufkhad and J. Carlier. SAT versus UNSAT. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, David S. Johnson and Michael A. Trick (eds.), Dimacs Series in Discrete Mathematics and Computer Science (26), American Mathematical Society, 1993.

[DB96] O. Dubois, Y. Boufkhad. A General Upper Bound for the Satisfiability Threshold of Random r-SAT Formulae. Preprint, 1996. See also: Les lois de tout our rein, in *Pour la Science*, July 1995.

[EIS76] S. Even, A. Itai, and A Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, 5(4), 1976.

[FP83] J. Franco and M. Paull. Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Math*, 5:77–87, 1983.

[FW97] J. Franco and R.P. Swamanithan. Average Case Results for Satisfiability Algorithms Under the Random Clause Model. To appear in: *Annals of Mathematics and Artificial Intelligence*.

[Fra86] John Franco. On the probabilistic performance of algorithms for the satisfiability problem. *Information Processing Letters*, 23:103–106, August 1986.

[Fre94] J.W. Freeman. Improvements to propositional satisfiability search algorithms., Doctoral Dissertation, University of Pennsylvania, Philadelphia, PA, 1994.

[FS92] A. Frieze and S. Suen. Analysis of three simple heuristics on a random instance of k-SAT. Preprint, 1992. To appear in *Journal of Algorithms*.

[Fu95] Xudong Fu. The complexity of the resolution proofs for the random set of clauses. Preprint, 1995.

[Gal77] Zvi Galil. On the complexity of regular resolution and the Davis-Putnam procedure. *Theoretical Computer Science*, 4:23–46, 1977.

[Goe92] A. Goerdt. A threshold for unsatisfiability. In *Proc. of the 17th International Symposium on Mathematical Foundations of Computer Science*, Prague, August 1992.

[Gol79] A. Goldberg. On the complexity of the satisfiability problem. Courant Computer Science Report No. 16., 1979. New York University.

[GU89] Giorgio Gallo and Giampaolo Urbani. Algorithms for testing the satisfiability of propositional formulae. *Journal of Logic Programming*, 7:45–61, 1989.

[Gu89] Jun Gu. Parallel algorithms and architectures for very fast search. Ph.D. Thesis, Department of Computer Science, University of Utah, 1989.

[Gu92] Jun Gu. Efficient local search for very large-scale satisfiability problems. *SIGART Bulletin*, 3(1):8–12, 1992.

[Gu93] Jun Gu. Local search for satisfiability problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4):1108–1129, July 1993.

[Gur91] Yuri Gurevich. Average case completeness. *JCSS* 42,3, 346-398, 1991.

[GW94] I.P. Gent and T. Walsh. The hardest random SAT problems. In *Proceedings, KI-94*, 1994.

[GW95] I.P. Gent and T. Walsh. Unsatisfied variables in local search. In *Proceedings, AISB-95*, pages 73–85, 1995. (Appears as a the book entitled *Hybrid Problems, Hybrid Solutions*, J. Hallam (Ed.), IOS Press, 1995.)

[Hak85] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[HK93] Steven Hampson and Dennis Kibler. Plateaus and plateau search in boolean satisfiability problems: When to give up searching and start again. Workshop Notes: 2nd DIMACS Challenge, 1993.

[Hoo88] J.N. Hooker. Resolution vs. cutting plane solution of inference problems: some computational experience, *Operations Research Letters*, 7(1):1–7, 1988.

[HF90] J.N. Hooker and C. Fedjki. Branch-and-cut solution of inference problems in propositional logic. *Annals of Mathematics and Artificial Intelligence*, 1:123–139, 1990.

[HTL92] T.H. Hu, C.Y. Tang, and R.C.T. Lee. An average case analysis of a resolution principle algorithm in mechanical theorem proving. *Annals of Mathematics and Artificial Intelligence*, 6:235–252, 1992.

[ILL89] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random number generation from one-way functions. *Proc. 21st STOC*, 1989, pp 12-24.

[Iwa89] Kazuo Iwama. CNF satisfiability test by counting and polynomial average time. *SIAM Journal on Computing*, 18(2):385–391, 1989.

[JAMS91] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[JW90] Robert E. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[KKRR90] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.

[KMPS94] Anil Kamath, Rajeev Motwani, Krishna Palem, and Paul Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. In *Proc. FOCS-94*, 1994.

[KKK96] L.M. Kirousis, E. Kranakis, and D. Krizanc. Approximating the unsatisfiability threshold of random formulas. Preprint, 1996.

[KP92] Elias Koutsoupias and Christos H. Papadimitriou. On the greedy algorithm for satisfiability. *Information Processing Letters*, 30:53–55, 1992.

[Kra95] Jan Krajíček. *Bounded Arithmetic, Propositional logic, and Complexity Theory*, Cambridge, 1995.

[Lar92] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, pages 6–22, January 1992.

[LT92] T. Larrabee and Y. Tsuji. Evidence for a satisfiability threshold for random 3CNF formulas. Technical Report UCSC-CRL-92-42, CRL, University of California, Santa Cruz, November 1992.

[Lev86] Leonid A. Levin. Average case complete problems. *SIAM J. Comput.*, 15(1):285–286, February 1986.

[Lew78] Harry R. Lewis. Renaming a set of clauses as a horn set. *Journal of the ACM*, 25(1):134–135, January 1978.

[Luby96] Michael Luby. *Pseudorandomness and Cryptographic Applications.* Princeton University Press, 1996.

[MJPL90] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings AAAI-90*, pages 17–24, 1990.

[MS95] Jahann A. Makowsky and Abraham Sharell. On average case complexity of SAT for symmetric distribution. *J. Logic Computat.*, 5(1):71–92, 1995.

[ML96] David G. Mitchell and Hector J. Levesque. Some pitfalls for experimenters with random SAT. *Artificial Intelligence*, 81:111–125, 1996.

[MSL92] D.G. Mitchell, B. Selman, and H.J. Levesque. Hard and easy distributions of SAT problems. In *Proc. AAAI-92*, San Jose, CA, 1992.

[Pap91] Christos H. Papadimitriou. On selecting a satisfying truth assignment. In *Proc. FOCS-91*, pages 163–169, 1991.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison–Wesley, 1994.

[Pre93] D. Pretolani. Solving satisfiability problems: An algorithm implementation challenge? (extended abstract). Workshop notes, 2nd DIMACS Challenge, 1993.

[Pud96] Pavel Pudlák. Lower bounds for resolution and cutting planes proofs and monotone computations. *J. Symbolic Logic* (to appear).

[Pur90] P. Purdom. A survey of average time analyses of satisfiability algorithms. *Journal of Information Processing*, 13(4), 1990.

[Sch96] Ingo Schiermeyer. Pure Literal Look Ahead: An $O(1,497^n)$ 3-Satisfiability Algorithm (Extended Abstract). In: *Workshop on the Satisfiability Problem, Technical Report, Siena, April 29-May3, 1996*, J. Franco, G. Gallo, H. Kleine-Büning, E. Speckenmeyer, C. Spera (Eds.), University of Köln, 1996.

[SAFS95] John Schlipf, Fred Annextein, John Franco and R.P. Swaminathan. On finding solutions for extended Horn formulas. *Information Processing Letters*, 54:133-137, 1995.

[SK93] Bart Selman and Henry Kautz. An empirical study of greedy local search for satisfiability testing. In *Proc. AAAI-93*, pages 46–51, 1993.

[SKC94] Bart Selman, Henry Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings, AAAI-94*, pages 337–343, 1994.

[SK96] Bart Selman and Scott Kirkpatrick. Critical behavior in the computational cost of satisfiability testing. *Artificial Intelligence*, 81:273–295, 1996.

[SLM92] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings, AAAI-92*, San Jose, CA, 1992.

[SD89] J.C. Simon and O. Dubois. Number of solutions of satisfiability instances – applications to knowledge bases. *Inter. J. of Pattern Recognition and A.I.*, 3(1):53–65, 1989.

[SG90] R. Sosic and J. Gu. A Polynomial Time Algorithm for the N-Queens Problem. *SIGART Bulletin*, 1(3), 1990.

[Sp93] William M. Spears. Simulated annealing for hard satisfiability problems. In, Workshop Notes from the 1993 DIMACS Challenge.

[Tse70] G.S. Tseitin. On the complexity of derivation in propositional calculus. *In*; Slisenko (Ed.), *Studies in Constructive Mathematics and Mathematical Logic – Part II, A.O.*, pages 115–125. Consultants Bureau, New York, 1970.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.

[VanG96] Allen Van Gelder. Personal Communication.

[VL88] Ramarathnam Venkatesan and Leonid A. Levin. Random instances of a graph coloring problem are hard. *Proc. 20th STOC*, 217-222, 1988.

[WB93] J. Wang and J. Belanger. On average **P** vs. average **NP**. In *Complexity Theory – Current research* (K. Ambos-Spies, S. Homer, U. Schöning, eds.), Cambridge University Press, 1993, pp.47-67.

[Weg] Ingo Wegener. *The Complexity of Boolean Functions*, Wiley and Teubner, 1987.

Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 3G4 CANADA

E-mail address: `sacook@cs.toronto.edu`

Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 3G4 CANADA

E-mail address: `mitchell@cs.toronto.edu`