An Empirical Study of Random SAT

by

David G. Mitchell
B.Sc. University of Toronto 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the Department
of
Computing Science

SIMON FRASER UNIVERSITY
August 1993

APPROVAL

Name: David G. Mitchell

Degree: Master of Science

Title of thesis: An Empirical Study of Random SAT

Examining Committee: Dr. Slawomir Pilarski
Chair

Dr. James Delgrande
Senior Supervisor

Dr. Arvind Gupta
Supervisor

Dr. James Crawford
External Examiner, AT&T Bell Laboratories

Date Approved: August 9, 1993

Title of Thesis/Project/Extended Essay

An Empirical Study of Random SAT

Author:

(signature)

David G. Mitchell

(name)

Aug. 12/93

(date)

Acknowledgements

I would like to thank Hector Levesque for starting me on this line of research and Bart Selman for help with early stages of the work, and many valuable discussions. Thanks to Jim Delgrande who supported and encouraged the work presented here and suggested many improvements to the thesis. Thanks also to Jimi Crawford and Larry Auton for helpful discussions and for the use of parts of their SAT testing program "tableau". Arvind Gupta and Jimi Crawford also provided many valuable comments on an earlier draft. I would also like to acknowledge AT&T Bell Laboratories for a valuable summer internship and for permission to use the above mentioned software for this work.

Abstract

Randomly generated formulas are commonly used for evaluating the performance of propositional satisfiability (SAT) testing procedures. Yet, little is known about the characteristics of such formulas. Here results are presented from an extensive empirical study using the Davis-Putnam Procedure (DPP) to test formulas from several distributions.

Each distribution exhibits a characteristic "easy–hard–easy" pattern of expected running time for DPP, as a function of the ratio of clauses to variables. This pattern is closely related to the proportion of formulas which are satisfiable. The other distribution parameters also have characteristic effects on these two measures. Previous empirical work is reviewed in light of this new data, demonstrating that an understanding of the patterns shown here is necessary for effective use of random formulas.

The data confirm previous observations that random SAT instances are often easy, thus challenging the validity of several studies which intended to demonstrate the efficacy of particular procedures. However, they also show that sets of moderately sized problems which are very hard to solve in practice do exist. Given an understanding of the results shown here, it is possible to select distributions of random formulas that are an effective tool for evaluating SAT testing procedures.

This work provides insight into the nature of randomly generated test sets and the pitfalls to be avoided in their use.

Contents

Acknowledgements iii
Abstract iv
List of Tables vii
List of Figures viii
1 Introduction 1
2 Satisfiability Testing 8
2.1 Definition of SAT 8
2.2 SAT Testing Procedures 9
2.2.1 The Davis-Putnam Procedure 9
2.2.2 Mathematical Programming 12
2.2.3 Local Search 12
2.2.4 Other Methods 13
2.3 Worst-Case Complexity 14
3 Random SAT 16
3.1 Introduction 16
3.2 Average-Case Analyses 17
3.3 Random $\mathcal{E}k$SAT 18
3.4 Random kSAT 20
3.5 Review of Experiments with Random SAT 21
4 Basic Patterns in Random SAT 25
4.1 Methods and Materials 25
4.1.1 SAT Testing Procedure 25
4.1.2 Measurement and Statistics 27

4.1.3 Formula Distributions 29
4.2 Testing random kSAT with DP 29
4.2.1 Varying m/n in random 3SAT 29
4.2.2 The easy-hard-easy pattern 31
4.2.3 Proportion of Satisfiable Formulas 32
4.2.4 Factoring Satisfiable and Unsatisfiable Subsets 33
4.2.5 Number of Satisfying Assignments 35
4.2.6 Varying n in Random 3SAT 37
4.2.7 Varying k in Random kSAT 40
5 Issues and Further Experiments 44
5.1 Testing Random $\mathcal{E}k$SAT with DP 45
5.2 Empirical Hardness of Random SAT 50
5.3 Critique of Previous Experiments 56
5.3.1 Random $\mathcal{E}k$SAT and Related Distributions 57
5.3.2 Random kSAT Formulas 62
5.3.3 Uniformly Distributed Clause Lengths 65
6 Discussion 69

List of Tables

4.1 Effect of increasing n on DP calls in random 3SAT 39

5.1 Effect of increasing n on DP calls in random $\mathcal{E}$3SAT 50
5.2 Effect of n on DP Calls: comparison of distributions 54
5.3 Locations of Tests in Gallo & Urbani (1989) 64

List of Figures

2.1 The Davis-Putnam Procedure 11

4.1 The Procedure DP 27
4.2 Median DP calls for 50-variable random 3SAT. 30
4.3 Satisfiable proportion of random 3SAT formulas. 33
4.4 Median DP calls for random 3SAT: satisfiable *vs.* unsatisfiable. . . . 34
4.5 Average number of solutions for satisfiable random 3SAT. 36
4.6 Median DP calls for random 3SAT: varying n. 38
4.7 Satisfiable proportion of random 3SAT: varying n. 39
4.8 DP Performance on random kSAT formulas: varying k 42

5.1 Median DP calls for 50 variable random $\mathcal{E}$3SAT. 46
5.2 50 variable random $\mathcal{E}$3SAT, satisfiable *vs.* unsatisfiable. 47
5.3 Average number of solutions for satisfiable random $\mathcal{E}$3SAT 48
5.4 Median DP calls for random $\mathcal{E}$3SAT: varying n. 49
5.5 Median DP Calls for random $\mathcal{E}k$SAT formulas: varying k. 51
5.6 Comparing random 3SAT and random $\mathcal{E}$3SAT: DP calls with varying n. 52
5.7 Growth of DP calls with n for various distributions 53
5.8 Median DP calls for formulas in Hooker (1988) *vs.* random 3SAT . . 57
5.9 Median DP calls for random $\mathcal{E}$5SAT and formulas from Hooker & Fedjki (1989) 59
5.10 Median DP Calls for random $\mathcal{E}$3SAT 60
5.11 Median DP calls for UD[1,7] 66
5.12 Median DP calls for UD[2,7] 67

Chapter 1

Introduction

The propositional satisfiability problem (SAT) is: *Given a boolean formula in conjunctive normal form (CNF), determine if there is an assignment of truth values to the variables of the formula which makes the formula true.* SAT plays a major role in the study of computational complexity and in several applied problems.

As the first problem to be shown NP-complete (Cook 1971), it has played an important role in the development of structural complexity theory. Categorizing a problem as "in P" or "NP-hard" is the most common and generally accepted way to indicate its computational tractability, and Garey and Johnson (Garey and Johnson 1979, Section 3.1) include 3SAT in their list of "6 basic NP-complete problems". More than twenty years after the seminal work of Cook the question "is P=NP?" remains one of the most fundamental open questions in computer science, and while the collection of closely related unresolved issues continues to grow, SAT remains central.

By definition NP-completeness entails (polynomial time) reducibility to all the other NP-complete problems, but SAT is also closely tied to many computational tasks in a more concrete sense. For example, it underlies many tasks in Artificial Intelligence (AI) which rely either directly or implicitly on SAT testing. In many cases, this stems from the fact that SAT is the dual of propositional theorem proving; A sentence α is logically implied by a set of sentences Σ if and only if $\Sigma \cup \{\overline{\alpha}\}$ is unsatisfiable. (Further, a satisfying truth assignment to $\Sigma \cup \{\overline{\alpha}\}$ is a model for Σ which

provides a counter-example to $\Sigma \supset \alpha$. In this model all sentences in Σ are true but α is false.) Thus, a SAT testing procedure which is efficient at showing unsatisfiability can be useful in automated deductive reasoning (the task for which propositional logic was originally invented). Default reasoning systems employ satisfiability in a slightly different way, usually using rules such as: "If it is consistent with what we know to assume α, then assume α". That is, if $\Sigma \cup \{\alpha\}$ is satisfiable then replace Σ with $\Sigma \cup \{\alpha\}$, and continue. Another use of SAT testing with widespread applicability is for solving constraint satisfaction problems (CSPs) (Mackworth 1991a). In a CNF formula encoding a CSP, each clause of the formula encodes a constraint on the assignment of values to the CSP variables. Finding a satisfying assignment to (or a model of) the formula representing a CSP is equivalent to finding a solution to the original CSP (Mackworth 1991b). Based on the success of some procedures which are effective at finding satisfying assignments, but not as successful at theorem proving (see 2.2.3), it has been argued that it may be useful to re-formulate as model-finding tasks many problems traditionally handled by theorem-proving (Selman, Levesque, and Mitchell 1992; Kautz and Selman 1992).

It could be argued that propositional reasoning is not of very great interest, since most general tasks in artificial intelligence require use of languages with much greater expressiveness than the propositional calculus. Nonetheless, many interesting problems *can* be expressed in the propositional calculus[1]. For some problems which are normally expressed in more expressive first-order languages as a matter of convenience, individual problem-solving episodes can be carried out in a propositional framework. Sometimes this can be done simply by reasoning over the Herbrand base (Genesereth and Nilsson 1987) of a first-order theory. In this and similar approaches, such as (Davis 1963; Lee and Plaisted 1990), propositional satisfiability is tested explicitly in the service of reasoning in a more powerful logic. Moreover, every decision (or semi-decision) procedure for a logical system stronger than classical propositional logic includes within it a SAT testing procedure. If one cannot even do propositional reasoning efficiently, the goal of reasoning effectively with more expressive languages would seem hopeless.

[1]To be precise, all the problems in NP

Applications outside of AI include a number of tasks in VLSI design and testing. For example, Larrabee (Larrabee 1992; Ferguson and Larrabee 1991) has shown how to generate CNF formulas such that a solution to the formula identifies a good test input for some particular fault in a VLSI design. Kamath and his colleagues (Kamath, Karmarkar, Ramakrishnan, and Resende 1991) report solving problems of logic circuit design by finding solutions to SAT instances which encode the input-output relations for a required logic circuit.

Since SAT is NP-complete, unless P=NP any algorithm which solves it will take at least super-polynomial time in the worst case. All known sound and complete procedures for SAT have worst-case exponential time complexity. However, it does not follow that *all* instances are hard. It is even possible that almost all instances of SAT which would occur in practical applications are easy; NP-completeness may result only from what could be considered a small set of pathalogical cases. The problem of polyhedral scene labeling (Waltz 1975) for example, was shown to be NP-complete (Kirousis and Papadimitriou 1985), yet programs based on Waltz's algorithm perform very well in practice. In a slightly different vein, the Simplex method for linear programming is exponential for some pathalogical cases (although polynomial time algorithms do exist for linear programming) but it performs extremely well in practice. From a practical point of view then, it seems worthwhile to develop a better understanding of how well different kinds of SAT testing procedures can be expected to perform in practice, independently of the worst case analytic results.

One approach to estimating what may happen in practice is to carry out average-case, rather than worst-case, analyses. The first work on average-case complexity of SAT (Goldberg 1979) considered formulas produced by selecting a fixed number of clauses uniformly at random from the set of all 3^n clauses that can be constructed from n variables. Goldberg showed that these formulas are solved on average in polynomial time by the Davis-Putnam Procedure. This result has been cited, by (Doyle and Patil 1991) for example, in arguments against "undue" concern for worst-case results, although doubt has since been raised about the significance of this result (see Sections 3.2 and 5.2 of this thesis). Since Goldberg's initial contribution, many

additional papers have been published with average-case results for a variety of variants of the Davis-Putnam Procedure, as well as some other procedures, using similar distributions of formulas. While these results are very interesting, there are many limitations to the average-case analysis approach.

The average-case analyses for SAT have been carried out by examining the performance of particular algorithms with particular formula distributions as input. Since average-case results are often much more difficult to arrive at than worst-case results for the same problem, these results are for very simple algorithms and for the distributions which are the most convenient to analyze. For many questions one would like to ask, no answers are available or likely to be soon forthcoming, and the analytic results that are available often fall far short of addressing concerns of those interested in solving real-world problems. For one thing, they are almost always from the asymptotic point of view, showing how the algorithm performs on the given class of instances *as the number of variables goes to infinity.* However, the performance of the algorithms, and basic properties of the formulas themselves, may be quite different in the limit than for any moderate-sized problem, so these results may tell us very little about the difficulty of solving the large but realistic sized problems one would encounter in a real application. As a simple example, an algorithm with $O(n^{10000})$ run time is faster in the limit that one with $O(2^{n/10000})$, yet for solving problems of sizes encountered in practice the second may be effective and the first useless. (Indeed, the method of choice for Linear Programming is the exponential worst-case Simplex method, even though polynomial time algorithms are known). There do appear to be practical examples of this, such as sets of SAT instances known to be average-case exponential, but which are easy to solve in practice for surprisingly large numbers of variables (Chvátal and Szemerédi 1988; Mitchell, Selman, and Levesque 1992).

Given these problems, it might seem that an empirical approach would be more fruitful. After all, if there are real problems to solve, why not simply test our algorithms on the actual problems? Given a large set of problems from the domain of concern (or for a more general approach, collections from several quite different domains), researchers could then work toward procedures that perform well on these test problems. There are several difficulties with this, not least of which is that such

collections often do not exist. This is especially a problem in AI applications, where collections of real knowledge bases are extremely rare, and very expensive to build. Many other problems equivalent to SAT are almost never formulated in a convenient manner for treating as SAT instances, perhaps because there is no SAT testing procedure which is seen as being able to solve a variety of real problems in practice. (In contrast, linear programming allowed the simplex method to be viewed as an effective solution method for a very wide range of problems. These had previously been tackled independently, but it has since become standard to present all of them in linear programming format.) Since investigators see SAT as too hard to solve in general, they may focus on representations with which they feel more familiar, or which make explicit aspects of their domain which intuitively seem valuable for performance optimization. Another problem is that we do not yet have widely available SAT testing programs capable of solving real problems in many of the domains of interest, so it would it would be very hard to make progress without another source of less challenging test problems.

Another approach would be to construct a collection of benchmark problems, deemed to be "challenging" or "realistic" by those working in the field. The most serious problem with this approach is that optimizing a procedure to a particular problem set often simply tunes it to the exact set of instances used, most likely at the expense of better performance over sets of problems to be tested in practice. This makes the test set useless for predicting performance in practice, and also leads to poorer results than could be had with a better approach. (Of course, this can also occur with collections of real problems, although perhaps to a lesser degree.) A good benchmark set would have to be designed to cover all the structurally interesting cases that make the problem hard, and be extensive enough to resist having particular procedures look very good or very bad due to luck or tuning, rather than general effectiveness. Some investigators have made small collections of problems from particular domains that seem to be of some use; see for example (Kamath, Karmarkar, Ramakrishnan, and Resende 1990; Selman, Levesque, and Mitchell 1992). However, in general we have neither good models of real problem distributions, nor significant collections of realistic problems to test our algorithms on. Thus, for empirical

methods we also seem to be forced to use randomly generated formulas, at least as a supplement to the real ones that are available.

Almost all reported work on empirical evaluation of SAT testing procedures does use random formulas. Clearly, results based on testing random formulas are only informative to the degree that the choice of random formulas is a reasonable one. To the extent the formulas used are indeed representative of some real problem, or at least challenging in some reasonable sense, they provide a valuable resource: a nearly inexhaustible supply of easily generated test instances. If the formulas used are extremely unrealistic though, for example if they are all trivial to test, there would be some doubt about the usefulness of any data produced by studying them. Previous workers were certainly aware of the *potential* importance of distribution selection. Yet, with a few exceptions, surprisingly little effort was made to validate the distributions that were used. The main goal of this thesis is to begin to redress this state of affairs. Of course, even the value of very challenging random formulas may questioned on the grounds that real problems probably have very structure, so that performance of a procedure on the random formulas may tell us very little about performance on real problems. However, to a large degree, they are for the time being "the only game in town" for large-scale testing.

Data are presented from an empirical study of randomly generated formulas. Unlike previous studies, very large sample sizes are used and a wide range of values for the parameters to the distributions are considered. The resulting data show clear patterns in the expected difficulty of testing of random formulas as a function the parameter values. This makes concrete the argument that the choice of distribution is quite important, and shows that some distributions which have been used in previous work are much more suitable as a source of test instances than others. The results indicate that understanding the basic properties of the distributions of random formulas is essential to effective empirical evaluation of SAT testing programs.

The remaining chapters are organized as follows. In Chapter 2, I give a formal definition of the SAT problem and discuss SAT testing algorithms and worst-case complexity results. Chapter 3 discusses random formulas, and reviews average-case results and the experimental literature using random formulas. The first sections of

Chapter 4 describe the methods and materials used for the present investigation. The remainder of that chapter explores in detail one family of random formula distributions, called random kSAT, showing the effects of varying each of the distribution parameters. The first section of Chapter 5 examines a second family of distributions: the one most commonly used in the literature. The second section compares this family of formula distributions to the previous one, with special attention to how testing time increases with increasing problem size. The remainder of Chapter 5 reviews the experimental SAT testing literature in light of the properties of random formulas revealed by the current experiments. Examples are chosen to illustrate how failure to understand these properties affects the validity or informativeness of data based on testing random formulas. Chapter 6 reviews the main results and discusses the implications of the work.

Chapter 2

Satisfiability Testing

In this chapter I specify the SAT problem precisely and review SAT testing algorithms along with worst-case complexity results. Average-case complexity results will be discussed in Chapter 3.

For uniformity, throughout the remainder of the thesis n will be used to denote the number of variables, m the number of clauses, and k the clause length. All refer to parameters to a distribution family, rather than actual counts for any particular formula.

2.1 Definition of SAT

Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of n boolean variables. A *truth assignment* $\mathcal{T}$ for U is a function $\mathcal{T} : U \rightarrow \{true, false\}$. Corresponding to each boolean variable u are two *literals*, the positive literal u and the negative literal $\overline{u}$. We say a literal u is *true* under $\mathcal{T}$ iff $\mathcal{T}$ maps the variable u onto *true*, and otherwise is *false*. Likewise, a literal $\overline{u}$ is *true* under $\mathcal{T}$ iff $\mathcal{T}$ maps the variable u onto *false*, and otherwise is *false*. We say $\overline{u}$ is the *negation* of u, and vice versa. A *clause* is a set of literals. We interpret a set of clauses as a *formula* of the propositional calculus in conjunctive normal form (CNF). We say a variable u is *mentioned* in a clause (or formula) if either of the corresponding literals, u or $\overline{u}$, occurs in it. A clause is *defined over* U, iff every variable mentioned in the clause is a member of U. A formula is defined over U iff every clause in the formula

is. Let ω be a formula defined over U, and C be a clause in ω. A truth assignment $\mathcal{T}$ for U *satisfies* C iff at least one literal $u \in C$ is *true* under $\mathcal{T}$. An assignment $\mathcal{T}$ for U satisfies ω iff it satisfies every clause in ω. (In the language of classical logic, $\mathcal{T}$ is a *model* of ω.) A satisfying assignment for a formula is also sometimes called a solution to it.

The SAT problem is:

> Instance: A set U of boolean variables and a formula ω defined over U.
> Question: Is there a truth assignment for U that satisfies ω?

2.2 SAT Testing Procedures

A *SAT Testing Procedure* is a procedure that takes a propositional CNF formula as input and returns either *yes* or *no* within a finite time. We say a SAT testing procedure is *sound* iff every formula to which it answers *yes* is satisfiable, and *complete* iff it answers *yes* to every satisfiable formula. The term *SAT testing program* is used to refer to an actual implementation of some SAT testing procedure.

Although SAT is defined as a decision procedure, many applications require a procedure that returns a satisfying truth assignment (a *witness*) for satisfiable formulas. Fortunately, most practical SAT testing procedures (including the one used for the present research) can easily be implemented to address this need at minimal computational cost.

2.2.1 The Davis-Putnam Procedure

The best known and most widely used SAT testing procedures are variants of the so-called[1] Davis-Putnam Procedure (Davis, Logemann, and Loveland 1962). Many variants have been proposed and implemented, and hereafter when I use the term "the Davis-Putnam Procedure " I mean "one or another of the common variants of this method". In section 4.1.1 I present the exact procedure used for the experiments reported in this thesis. I will use "DP" when referring to this procedure in particular.

[1]See the discussion in 2.2.4

The basic procedure is essentially a divide-and-conquer method. Given a formula ω, at each step a variable u is selected from among those mentioned in ω, and two subproblems are created, one with u set to *true*, the other with u set to *false*. Each of the subproblems is simplified as follows. If u has just been set *true* (resp. *false*), delete from ω all clauses in which u ($\overline{u}$) occurs, and delete $\overline{u}$ (u) from all clauses in which it occurs. Now, ω is satisfiable iff at least one of the (simplified) subproblems is. Each step of the Davis-Putnam Procedure reduces the problem of satisfiability of ω to the satisfiability of two new formulas, each with fewer literals than ω. If any subproblem is reduced to the empty set, then it is satisfiable. If any subproblem contains an empty clause, then it is unsatisfiable. It is easy to keep track of the variables that have been set during this search process, and thus return a witness if satisfiability is established.

The most straightforward implementation is as a recursive procedure which can be viewed as standard backtracking, or as a depth-first search through a space of partial-assignments, implicitly enumerating all possible truth assignments. A number of enhancements are commonly used to improve the efficiency of the basic backtracking search. The two most common are the *pure literal rule* and the *unit clause rule.* The pure literal rule checks for the existence of any variable u for which one corresponding literal is mentioned in the formula, but not its negation. If u is mentioned but not $\overline{u}$, then u can be immediately set *true*; $\overline{u}$ need not be considered. The unit clause rule ensures that any time there is a clause of length one, the variable mentioned in that clause is set immediately to the value which makes the clause true. Again, the other setting need not be considered since it will immediately produce a contradiction (ie., the simplification operation will generate an empty clause). A more-or-less prototypical version of the Davis-Putnam Procedure is given in Figure 2.1.

The unit clause and pure literal rules do not change the logic of the procedure, but are rules for recognizing variables on which splitting can be avoided. If the variables set by these rules were left to be set by the normal backtracking order, the same contradictions or solutions would be found, but using the rules avoids some unnecessary searching. The effect of these rules is more significant than just setting the one variable correctly as soon as possible. Since applying one of these rules results

ProcedureDavis-Putnam(ω):

1. If ω is empty, return *yes*.
2. If ω contains an empty clause, return *no*.
3. (Unit Clause Rule): If there is a unit clause $\{u\}$ in ω, set u *true*, and return the result of calling **Davis-Putman** on the resulting (simplified) formula.
4. (Pure Literal Rule): If there is a literal u mentioned in ω who's negation is not, set u to *true* and return the result of calling **Davis-Putman** on the resulting formula.
5. (Splitting Rule): Pick an unassigned variable u mentioned in ω, and set u *true*. If **Davis-Putman** called on the resulting formula returns *yes*, then return *yes*. Otherwise, set u false, and return the result of calling **Davis-Putnam** on the resulting formula.

Figure 2.1: The Davis-Putnam Procedure

in eliminating a variable, often new pure literals (by either rule) or new unit clauses (only by the unit clause rule) are created. In the case of the unit clause rule, an entire sequence of applications of this rule (a process called *unit propagation*) can be carried out in time linear in the size of the formula (Dowling and Gallier 1984).

In practice, implementation of the the unit clause rule is extremely beneficial. In fact it is easy to demonstrate that it is all but essential, since even very small formulas, say with 25 variables, can take overwhelmingly long to test for a procedure without the unit clause rule. It is often assumed that the pure literal rule is also valuable, but there appears to be no substantial verification of this in the literature. Many other heuristics have been tried, with varying degree's of success. Subsumption checking has also been used (Loveland 1978) and heuristics based on clause length and frequency of variable occurrence were suggested in (Goldberg 1979). Reports of the use of additional heuristics can be found in (Crawford and Auton 1993; Zabih and McAllester 1991; Gallo and Urbani 1989).

2.2.2 Mathematical Programming

Much of the experimental work[2], in SAT testing has been done within the operations research community where mathematical programming techniques are widely used. Transformation from SAT to 0–1 integer programming is straightforward (Blair, Jeroslow, and Lowe 1986; Hooker 1988b), and several investigations have examined the performance of methods such as branch-and-bound and branch-and-cut search (Hooker 1988c; Hooker and Fedjki 1989), and the interior-point method (Kamath, Karmarkar, Ramakrishnan, and Resende 1990). No experiments have been reported which establish the effectiveness of the mathematical programming methods relative to other standard methods such as the Davis-Putnam Procedure . A major reason for this is that the experiments were almost all carried out using inappropriate selections of formulas, as will be shown in Section 5.3.1.

2.2.3 Local Search

Recently, local search based procedures for finding solutions to SAT instances have been presented by several workers (Selman, Levesque, and Mitchell 1992; Gu 1992; Gent and Walsh 1993). A generic local search procedure for SAT is:

Procedure Local-Search-SAT (ω, U):

1. Generate an initial truth assignment $\mathcal{T}$ for U.
2. If $\mathcal{T}$ satisfies ω return *yes*.
3. If the stopping criterion has been reached, return *no*.
4. Replace $\mathcal{T}$ with the "best" truth assignment which is a "near neighbour" of $\mathcal{T}$ and go to 2.

"Near neighbours" of $\mathcal{T}$ are typically truth assignments differing on only one variable. "Best" is typically taken to mean satisfying the largest number of clauses in the formula. The stopping criterion is usually some number of iterations of steps 2–4. As can be seen from this description, generally these procedures are sound but incomplete SAT testers. That is, if *yes* is returned then the formula is satisfiable, but

[2] and some theoretical work, for example (Hooker 1988a; Chandru and Hooker 1991)

if *no* is returned, there is a (presumably small) non-zero probability the the formula is satisfiable, but the procedure failed to find a solution.

The best of these procedures perform at least as well on satisfiable instances as the best sound and complete procedures, and much better for some kinds of problems (Selman, Levesque, and Mitchell 1992). However, they are only useful in applications where the incompleteness is an acceptable cost for some other benefit. In particular, they are not likely to be useful as theorem provers. With respect to theorem proving they are complete but unsound, but more importantly they are not expected to be faster at showing unsatisfiability (theoremhood) than the traditional sound and complete methods.

2.2.4 Other Methods

Davis and Putnam presented a procedure that combines a variable elimination rule with the unit clause and pure literal rules (Davis and Putnam 1960). Perhaps because the latter two rules were introduced there, (Davis and Putnam 1960) is commonly miss-cited as the source of the procedure described in section 2.2.1. The variable elimination method repeatedly picks a variable from the formula, then adds to the formula all the clauses that can be generated by resolving on that literal[3], and then removing all subsumed and tautological clauses from the formula. At each step, the number of variables in the formula is reduced by one. The difficulty with this method is that an exponential number of clauses may be generated, and it was this difficulty that lead Davis et al. to introduce the procedure that is usually called the Davis-Putnam Procedure (Davis, Logemann, and Loveland 1962).

A procedure which is somewhat complementary to the Davis-Putnam Procedure was introduced in (Iwama 1989). Each clause in a formula may be seen as eliminating some set of truth assignments as potential solutions. This procedure examines each clause, then each pair of independent clauses, then each such triple, and so on. At each step, it counts how many truth assignments are ruled out as solutions by the clause

[3]Two clauses can be *resolved* iff for some variable u, u is a member of one clause and $\overline{u}$ is a member of the other. The result of resolving $\{u, x_1, \ldots x_i\}$, and $\{\overline{u}, y_1, \ldots y_j\}$ is $\{x_1, \ldots x_i, y_1, \ldots y_j\}$.

or combination of clauses considered. After completion, if all 2^n truth assignments have been ruled out, then the formula is unsatisfiable, otherwise it is satisfiable. This procedure tends to get faster as clause lengths increase, whereas the Davis-Putnam Procedure gets slower. Note that this procedure cannot easily be adapted to return a witness for satisfiable formulas.

Two non-clausal methods (i.e., for which the input need not be in CNF) are presented in (Van Gelder 1988) and (Haralick and Wu 1987). Van Gelder's method is quite similar to the Davis-Putnam Procedure when inputs are restricted to clausal form. The method of Haralick and Wu employs some 40 inference rules, and I am aware of no attempt to determine the relation between this and other systems.

2.3 Worst-Case Complexity

SAT is NP-complete, and thus all known algorithms for it take exponential time in the worst case. The best published upper bound is $O(1.62^n)$ for 3SAT, where n is the number of variables (Monien and Speckenmeyer 1985). A number of polynomial time sub-classes, most based on syntactic restrictions, have been studied. The best known of these are HORN-SAT, in which no clause may contain more than one positive literal, and 2-SAT, in which no clause has length greater than two. Both of these cases can be solved in linear time (Dowling and Gallier 1984; Even, Itai, and Shamir 1976; Aspvall, Plass, and Tarjan 1979). Several other polynomial time sub-sets have been found, for example (Chandru and Hooker 1991; Gallo and Scutella 1988; Yamasaki and Doshita 1986). According to (Schaefer 1978), any polynomial time subset of SAT falls into one of five easily described classes. Two of these classes are formulas which can be written as 2CNF and HORN-CNF [4], and two others are trivial. The remaining class is given by an algebraic description, which does not have an obvious syntactic interpretation.

Although no non-trivial lower bound is known for SAT, such lower bounds are known for some SAT testing procedures. The Davis-Putnam Procedure can be shown to be a special case of regular resolution (Goldberg 1979) (a resolution procedure

[4] Actually, they give six classes, but one of these can be trivially transformed to HORN-CNF .

is regular if any given literal is annihilated at most once) which was shown to be exponential by (Tseitin 1968).

Chapter 3

Random SAT

3.1 Introduction

Use of the term *random SAT* is in analogy to *random variable.* A random variable is a set of values together with a probability distribution function defined over those values. A random SAT problem is a set of formulas Ω, together with a probability distribution function ϕ over the elements of Ω. Formulas produced according to such a distribution are called *random formulas.* Typically, ϕ is specified implicitly by giving a method of selecting clauses from which to construct each formula. A family of distributions is determined by a set of parameters together with a mapping from the parameters onto the set of possible distributions. Every average-case analysis of SAT is an analysis of a random SAT problem; it looks at the arithmetic mean of the performance of an algorithm across all formulas $\omega \in \Omega$, weighted according to ϕ. An empirical investigation of random SAT is a study of formulas which have been generated pseudo-randomly according to (a good approximation to) the specified distribution.

There is a small (but perhaps important) discrepancy between the usual definition of a formula (as in section 2.1) and the normal practice with random SAT . In the latter case, formulas are viewed as being constructed by selecting clauses randomly *with replacement* from some given set of clauses. Since the same clause may appear more than once in a formula constructed this way, such formulas are *sequences* of clauses,

not sets. This slight modification to the normal (logical) view of formulas is more-or-less standard practice in both analytic and experimental studies of random SAT . On the one hand it simplifies many analyses, and on the other it makes actually generating formulas less costly. (It may even be argued that this is a more realistic model of applied SAT testing).

3.2 Average-Case Analyses

The first average-case analysis of SAT appeared in Allen Goldberg's PhD. thesis (Goldberg 1979). There he showed that for formulas with m clauses chosen uniformly at random (with replacement) from all 3^n possible clauses defined over n variables, the Davis-Putnam Procedure has average-case complexity of $O(mn^2)$. Goldberg then extended this model as follows. Let $0 < p < 1$. Then construct a clause by mentioning in it each variable with probability p, and determining the sign of the literals by negating each included literal with probability 0.5. The previous distribution is the special case of this one with $p = 2/3$. Regardless of p, all possible clauses defined over n variables are included in the distribution, but p biases the distribution toward shorter or longer clauses. The lengths of clauses are distributed according to a binomial distribution, with the mean clause length being np. Goldberg's analysis showed that the Davis-Putnam Procedure can test these problems in polynomial time on average, for all constant values of p.

Examining these distributions further, (Franco and Paull 1983) showed that randomly picking truth assignments will solve all but an exponentially small number of instances from such a distribution in polynomial time. This is weaker than a polynomial average time result, since the few instances not solved in polynomial time could take so long that the average is exponential, but it does strongly suggest that in this family of distributions very easy instances are highly predominant. (Franco and Paull 1983) argued that it is an unrealistic model of average SAT, and is not an appropriate choice for analysis or evaluation of SAT procedures. Based on a preliminary analysis, they suggested that randomly generated formulas with fixed clause lengths might be a more suitable choice. Average-case analyses of fixed clause length formulas are

discussed in Section 3.4.

Goldberg and several other researchers went on to give further results for formulas like these, but allowed p to vary as a function of n. They found various parameter ranges where some variants of the Davis-Putnam Procedure performed in polynomial average time, or where simplified versions required exponential average time. Polynomial average-time versions have been given for most parameter ranges. Paraphrasing (Franco 1986a): with high probability, a random formula generated according to this distribution has no solution unless the average number of literals per clause is increasing with m and n, but if this is the case then the probability that a clause is satisfied by a random truth assignment is increasing fast enough that a random truth assignment will satisfy a random formula with high probability. Reviews and recent contributions can be found in (Franco 1986a; Franco and Ho 1988; Iwama 1989; Purdom 1990).

This thesis focuses on formula distributions where average clause length is constant, so p varies inversely with n. Results in (Franco 1986a) and (Franco and Ho 1988) show that, for this case also, a polynomial time algorithm solves a random instance with probability approaching 1 as n goes to infinity. Purdom and his colleagues (Brown and Purdom 1981; Bugrara, Pan, and Purdom 1989; Purdom and Brown 1987) show true polynomial average case results for a slightly different family of problem distributions. In this work, rather than mentioning each variable in a clause with some fixed probability, they mention each possible *literal* with a fixed probability. The result is that clauses which contain both u and $\overline{u}$ (and are thus trivially satisfiable) may occur. This distribution should be no harder (as argued in (Purdom 1983)) for all reasonable values of p. In light of Franco's results and the experimental results reported in Chapter 5, this family of distributions is not considered further.

3.3 Random $\mathcal{E}k$SAT

In empirical work it was soon recognized that formulas from the distributions described above were often very easy in practice. One common sense explanation is that there is a high probability of a formula containing an empty clause (and thus

being trivially unsatisfiable), or containing a large enough number of unit clauses that it is extremely easy to test (since after unit propagation, the remaining formula will be quite small). As a consequence, modified versions came into use, first disallowing empty clauses, and later unit clauses as well. For these formulas, the clause lengths follow a distribution which is the same as a binomial distribution truncated at one end.

The choice of parameters for describing the binomial clause length formula distributions was made from the point of view of performing the analyses of them. There are other sets of parameters which could be chosen to represent and investigate exactly the same problem space. In empirical work, it has been usual to consider primarily the case where the average clause length is held constant as the number of variables grows. The experimental results in Chapter 4 suggest that this is a useful viewpoint, and it also makes for easier comparison, at least at the intuitive level, with the next most common family of random formula distributions which uses fixed length clauses. Experimental and theoretical work also shows that interesting patterns occur as the number of clauses is varied, and as will be seen, these patterns occur at the same ratios of of clauses-to-variables regardless of the number of variables in a formula.

As a consequence, the work presented in this thesis views the problem space in terms of the three alternate parameters: n (the number of variables), m/n (the ratio of clauses to variables, which will continue to be expressed as m/n), and k (the average clause length). When no special restrictions are placed on clause lengths, the transformation from using m and p to using m/n and k is trivial, since $p = k/n$. However, we are most interested in distributions where clause lengths of less than 2 are not permitted. For these formulas, this equation holds in the limit as n goes to infinity, but is only approximately correct for small n. The family of distributions constructed this way, with minimum clause length of 2 and average clause length held determined by the parameter k and held constant as n varies, will be called *random* $\mathcal{E}kSAT$. No significant analyses have been performed on this family.

3.4 Random kSAT

The next most used family of distributions is that in which formulas are constructed as follows. There are three parameters n, m/n and k, where m,n and k are fixed positive integers. The parameters have the same meanings as before except that we have clauses of fixed length k, rather than average length k. An instance is created by selecting m clauses uniformly at random, and with replacement, from the set of all $2^k\binom{n}{k}$ clauses of length k which can be constructed from n variables, with no variable mentioned more than once in a clause. The name used for this distribution family is random kSAT. In contrast to the binomial clause length families, it has been common to view the fixed clause length formulas in terms of the ratio of clauses to variables.

Instances of random 3SAT can be shown to be almost certainly unsatisfiable, with probability tending to one as n goes to infinity, whenever $m/n > 5.19$ (Franco and Paull 1983; DuBois 1991). It has also been shown that for every constant value of $m/n > 5.6$, any resolution proof of the unsatisfiability of a random 3SAT formula will take exponentially many steps (Chvátal and Szemerédi 1988). It follows immediately that the Davis-Putnam Procedure will take exponential average time for formulas in this region. Frieze and Suen, extending earlier results in (Chao and Franco 1990) and (Chvátal and Reed 1992), give an algorithm that finds a satisfying assignment to a random 3SAT instance, with probability tending to 1 as n goes to infinity, for all $m/n < 3.003$ (Frieze and Suen 1992)[1]. With $m/n < 1$, these solutions can be found in polynomial time with probability approaching 1, and with $m/n < 2.9$, solutions can be found in polynomial time with probability bounded below by a constant (Chao and Franco 1986). The only significant result for the region between $n = 3.003$ and $n = 5.19$, is that guessing truth assignments for this distribution does not find solutions in polynomial time (Franco 1986b). The only polynomial average case results for random 3SAT (except for the case of $k = 2$, which can be solved in linear time in the worst case), are for $m/n < 1$ (Franco 1984). Results similar to those of random 3SAT are attainable for $4 < k < 40$.

[1]Actually, they use the traditional notion of a formula as a set rather than a sequence, but the result is expected to be the same for the sequence version (Frieze 1993)

3.5 Review of Experiments with Random SAT

In this section, the main empirical work using random SAT is reviewed, describing the motivation of the work and the kinds of formulas used. Careful examination of this work is left to Chapter 5, after the main data from the current experiments has been presented.

The reporting of computational experiments in SAT testing appears to begin in 1986 with Blair and his colleagues (Blair, Jeroslow, and Lowe 1986). In this paper, largely devoted to showing relations between methods in mathematical programming and propositional theorem proving, they reported experiments testing three existing mathematical programming software packages on random formulas generated in a manner similar to random kSAT. Most of the formulas used had fewer than 40 variables, and because many details were left unspecified, their results appear not to be replicatable, or comparable to other work.

Hooker extended the work of (Blair, Jeroslow, and Lowe 1986), first comparing programs implementing resolution theorem proving and branch-and-bound (Hooker 1988c), and later comparing several related mathematical programming approaches (Hooker and Fedjki 1989). The first of these papers reported generating several thousand random formulas with 10 to 50 variables, according to a distribution like random $\mathcal{E}k$SAT, but allowing unit clauses. The main claim of the paper was that the resolution method was much inferior. In section 5.3.1 I show that resolution solves these formulas easily, so the fault must be with the particular implementation used.

Hooker and Fedjki compared three mathematical programming methods, using formulas distributed much like those of random $\mathcal{E}$5SAT (Hooker and Fedjki 1989). Here Hooker and Fedjki justified using the binomial clause length family by arguing that fixed clause lengths are highly unrealistic. They did recognize the problem that the formulas used in (Hooker 1988c) tend to contain too many unit clauses to be challenging. They generated formulas by the same method but with clause length restricted to be at least two. They set $np = 5$, which gives an average clause length of approximately (but slightly greater than) 5. This distribution approaches random $\mathcal{E}$5SAT as n goes to infinity.

The work of (Hooker and Fedjki 1989) also offered the first recognition in print of the effect of the number of clauses on the likelihood of being satisfiable, and on the expected difficulty. Based on very simple estimate of the expected number of solutions, they argued that formulas with low clause-to-variable ratio will almost all be satisfiable, and will be easy because they will have many solutions. From an additional probabilistic estimate, they argued that formulas with many clauses will almost always be unsatisfiable, and that the hardest problems should be expected between these two regions; in this thesis this area is called the "transition region". Their test set consisted of 80 formulas of 100 variables and 50 formulas of 50 variables, with formulas at several clause-to-variable ratios between 6 and 15.

Kamath and his colleagues (Kamath, Karmarkar, Ramakrishnan, and Resende 1990) also mapped CNF formulas into 0-1 integer programming problems and employed mathematical programming software – this time software implementing the interior-point method (Karmarkar, Resende, and Ramakrishnan 1988). They used small samples of formulas from the random $\mathcal{E}k$SAT distributions at a variety of parameter values, including formulas of many thousands of variables.

The mathematical programming experiments cited above used linear programming relaxation to aid in solving SAT formulated as 0-1 integer programming. Gallo and Urbani (Gallo and Urbani 1989) took a similar approach, but remained within the framework of SAT. They used a linear-time program for HORN-SAT to prune the search space for an implementation of the Davis-Putnam Procedure, by checking before application of the splitting rule that at least the current set of Horn clauses were satisfiable[2]. They developed two new algorithms, and compared their performance with that of a standard variant of the Davis-Putnam Procedure . As test problems, they used randomly generated formulas from two distribution families. One set of tests they performed with random 3SAT, and another with formulas having clause lengths uniformly distributed in the range [1,7]. I call these formulas UD[1,7] formulas. They also performed some additional tests using random formulas with a fixed clause

[2]Indeed, the relaxation method is actually implicit in almost all work with the Davis-Putnam Procedure. See (Chandru and Hooker 1988) for a description of the relation between linear programming relaxation and unit propagation.

length of 3 and pre-specified proportions of Horn clauses, but the method of obtaining these particular formulas was not made clear.

In 1991 and 1992 a SAT competition was held at the University of Paderborn (Buro and Büning 1992). The organizers requested program submissions in the form of C programs which took a formula in a given format, and determined the satisfiability of the formula. They selected a fixed set of randomly generated problems to test the entries, running all programs on the same formulas, using a single machine and compiler as the test bed. Their criterion for winning was simply the time to correctly report the status of the entire set of formulas. The formulas selected were 480 random kSAT formulas with a variety of choices of numbers of variables and clause lengths, and 80 formulas from a distribution mixing many short clauses with a few very long clauses. Formula sizes up to several hundred variables were used. They found that many submitted programs failed to successfully handle all the formulas, and that the fastest submissions were all variants of the Davis-Putnam Procedure.

(Gu 1992) and (Selman, Levesque, and Mitchell 1992) introduced local search procedures for SAT. (Gu 1992) performed experiments with random kSAT formulas at a variety of parameter values, including formulas of up to 5000 variables, testing 30 formulas at each sample point. (Selman, Levesque, and Mitchell 1992) tested their method on random 3SAT formulas with the clause-to-variable ratio set to 4.3, near the hardest point as predicted by the data in (Mitchell, Selman, and Levesque 1992).

(Haralick and Wu 1987) used random formulas, but in a non-clausal language with all the standard logical symbols of propositional calculus. (Cheeseman, Kanefsky, and Taylor 1991) generated kSAT instances by transformation from random k-colorability problems. (Zabih and McAllester 1991) generated direct encodings of random graph colouring instances. It is not clear how these distributions are related to any others in the literature.

A small number of experimenters have used non-randomly generated formulas. For example, (Zabih and McAllester 1991) and (Selman, Levesque, and Mitchell 1992) use instances of the N-queens problem encoded in CNF. Although N-queens has long been regarded as a benchmark problem in some communities, recent results of both experimental and theoretical nature suggest that the problem is in fact extremely easy,

although backtracking methods typically do not perform particularly well. (Selman, Levesque, and Mitchell 1992) also used the graph colouring problems from (Johnson, Aragon, McGeoch, and Schevon 1991), and boolean induction problems from (Kamath, Karmarkar, Ramakrishnan, and Resende 1991) to demonstrate the speed of their procedure.

Chapter 4

Basic Patterns in Random SAT

This chapter presents empirical data from testing the satisfiability of a very large number of formulas from the random kSAT family of distributions, using the Davis-Putnam Procedure. The primary objective is to develop a basic understanding of the effects of each of the parameters to the distributions; n (the number of variables), m/n (the ratio of clauses to variables) and k (the clause length). The graphs in this chapter show the patterns of typical time to test a formula as measured by the number of steps performed by the Davis-Putnam Procedure, along with the proportion of formulas which are satisfiable, as each of these parameters is varied. The data presented in this chapter is a much extended version of preliminary data, based on smaller sample sizes and a modest range of parameter values, published in (Mitchell, Selman, and Levesque 1992).

4.1 Methods and Materials

4.1.1 SAT Testing Procedure

The SAT testing procedure used for this study is a version of the Davis-Putnam Procedure, which was chosen because of it's simplicity and widespread use, and because it is believed that variants of the Davis-Putnam Procedure (perhaps incorporating carefully tailored heuristics), are the fastest sound and complete procedures available.

Also, the Davis-Putnam Procedure it is a special case of resolution, the most common form of theorem proving used in the AI community.

The program used here, which will be referred to as "DP", incorporates the splitting rule and the unit clause rule, but no other enhancements. In particular, it does not incorporate the pure literal rule because, although very commonly used, its utility is not well established. The reader familiar with the literature may note that the most-analyzed procedure is actually the one with the pure-literal rule but not the unit-clause rule. However, this algorithm does not perform well enough in practice to serve the present purpose. The splitting rule by itself (blind backtracking) is essentially useless in practice. The procedure DP is arguably the simplest known procedure which performs well enough to carry out an investigation of this kind, and since it is a restricted version of the fastest methods in use, it is a kind of "base case" of the best methods available for the task. Thus, the data presented here are a useful starting point for comparison of the properties of the formula distributions, and provide a kind of lower bound on the performance that a reasonable program should be able to attain.

Experience has shown that extremely careful implementation is required to produce a program that performs acceptably in practice; the exponential growth makes it very easy to produce surprisingly inefficient code by neglecting an apparently minor inefficiency or waste of space. Although careful implementation will never make an exponential algorithm polynomial, it can make all the difference in practical terms between a useful and a useless program. Since carrying out informative experiments requires very large numbers of non-trivial problems to be solved, a poor implementation effectively makes a study such as the present one impossible. The present implementation is derived from that used by (Crawford and Auton 1993), and employs a number of indexing methods and programming tricks developed by them. Their implementation, however, employs a number of heuristics carefully tuned to optimize performance on random 3SAT formulas, whereas the version used for the present experiments implements the simple algorithm described in this section. The program was implemented in the C language, and compiled with the standard SunOS C compiler. The experiments were carried out on a Sun SparcStation 10 with 36MHz

SuperSPARC cpu.

The procedure DP is shown in Figure 4.1. The description assumes there is some total ("lexical") ordering of the variables (in practice, they are simply numbered 1-to-n). Each splitting rule invocation selects, among those variables which still occur

Procedure DP(ω):

1. If ω is empty, return *yes*.
2. If ω contains an empty clause, return *no*.
3. (Unit Clause Rule): If there are one or more unit clauses in ω, select the least variable which is mentioned in a unit clause, and set this variable to satisfy that unit clause true. Simplify the formula, increment the step counter, and return the result of calling DP on the resulting (simplified) formula.
4. (Splitting Rule): Let v be the least variable in U which has not been assigned a truth value. Assign it the value *true*, increment the step counter, and call DP on the simplified formula. If this call returns *yes*, then return *yes*. Otherwise, give v the value *false*, increment the step counter, and return the result of calling DP on the re-simplified formula.

Figure 4.1: The Procedure DP

in the current subproblem[1], the least variable which has not been assigned a truth value, and first assigns it the value *true*, then only if this fails, the value *false*. Each recursive call to DP is considered one step for the purpose of measuring performance. This is the same as counting one step every time a variable is assigned a value, and the same as counting nodes in the search tree.

4.1.2 Measurement and Statistics

The primary interest here is to understand the *typical* performance of DP on random kSAT formulas. Thus, we want a statistical measure which indicates where the "typical" case, or where the central bulk of cases lies. For a set of formulas at a particular

[1]In the process of simplification after setting one or more variables, it sometimes happens that all clauses which mention some as yet unassigned variable have been removed from the remaining formula. The version of the Davis-Putnam Procedure in (Mitchell, Selman, and Levesque 1992) fails to notice these, and thus branches more times than necessary.

set of parameter values, the distribution of DP running times on a set of random formulas – whether measured by DP steps, by cpu time, or any other reasonable measure – has quite high variance and is extremely skewed, with values hundreds or thousands of times the average being frequent. The arithmetic mean is very sensitive to extreme values, and for this kind of highly asymmetric distribution, the median is more robust and provides a better indicator of the typical performance than the mean (Beckman and Cook 1983; Tukey 1960). Thus, as a measure of performance, the median number of DP calls (or steps) is reported. In fact, the mean and median are highly correlated, and the patterns to be discussed are almost identical by either measure. The main consequence of the choosing the median is that the curves in the graphs are much less noisy, leading to a clearer picture of the patterns of concern. Other possible measures, such as counting splitting rule applications or cpu-time rather than DP calls, are so highly correlated that there seems little point in debating which is preferable. In addition to median performance, this chapter also examines the proportion (or per cent) of formulas which are satisfiable.

The issue of variability of execution times, even for formulas from exactly the same distribution, is a complex one. It was noted above the distributions are highly skewed and that extreme values are common. Variances are often much larger than the means. But there are also very rare, but even more extreme cases. For example, of 5000 random 3SAT formulas with 50 variables at $m/n = 2$, the vast majority of formulas were tested in less than 50 steps, a few took several hundred steps, and a small number took several thousand steps. However, one single extreme formula took over 17 million steps! However, previous experience has shown that such formulas are often tested very quickly with the variable ordering changed, suggesting there may be nothing fundamentally difficult about the formulas themselves. A second point worth noting is that these extreme cases also show up in the data for formulas known to be easy in the worst case, like 2SAT. This may be accounted for by the fact DP can take exponential time on 2SAT, or by the fact that in the present implementation of DP , the unit propagation is not strictly linear time. Regardless, this issue of run time variance is one which warrants further study in and of itself, but will not be further addressed here.

Unless otherwise stated, each point on a curve represents a sample statistic calculated from 5000 formulas generated pseudo-randomly from the specified distribution. The median and proportion measures are subject to error analysis on the basis of the central limit theorems (just as the mean is) and some graphs include error bars from such analyses. When present, these error bars represent 99% confidence bounds on the points given[2].

4.1.3 Formula Distributions

Although formulas from the binomial clause length and other similar distributions are the most commonly used in practice, the analytic results on this family suggest that it may be a poor source of test problems. Since the experiments performed in (Mitchell, Selman, and Levesque 1992) bore out Franco's suggestion that the fixed clause length family may provide more challenging test sets, that family was chosen to as the primary material to study. As a reminder, random kSAT formulas are formulas constructed by selecting m clauses uniformly at random, with replacement, from the set of all possible clauses of length k defined over n variables. The range of parameter values looked at is; $n \in \{25, 50, 75\}$ and $k \in \{2, 3, 4, 5\}$, with m/n varied from 1/5 to 50. Since 3 is the smallest k for which kSAT is NP-Complete, most researchers regard this as the most interesting case, and attention is primarily limited to this case except for the final sub-section, where k is the parameter varied.

4.2 Testing random kSAT with DP

4.2.1 Varying m/n in random 3SAT

To look at the effect of clause-to-variable ratio formulas were generated from random 3SAT, with the number of variables fixed at 50, and the clause-to-variable ratio was

[2]That is, the point plotted is a statistic computed from the random sample, which provides an estimate of that same statistic for the entire population of formulas the sample was taken from. We can be 99% certain that the actual value of the population statistic is within the range given by the upper and lower segments of the error bars.

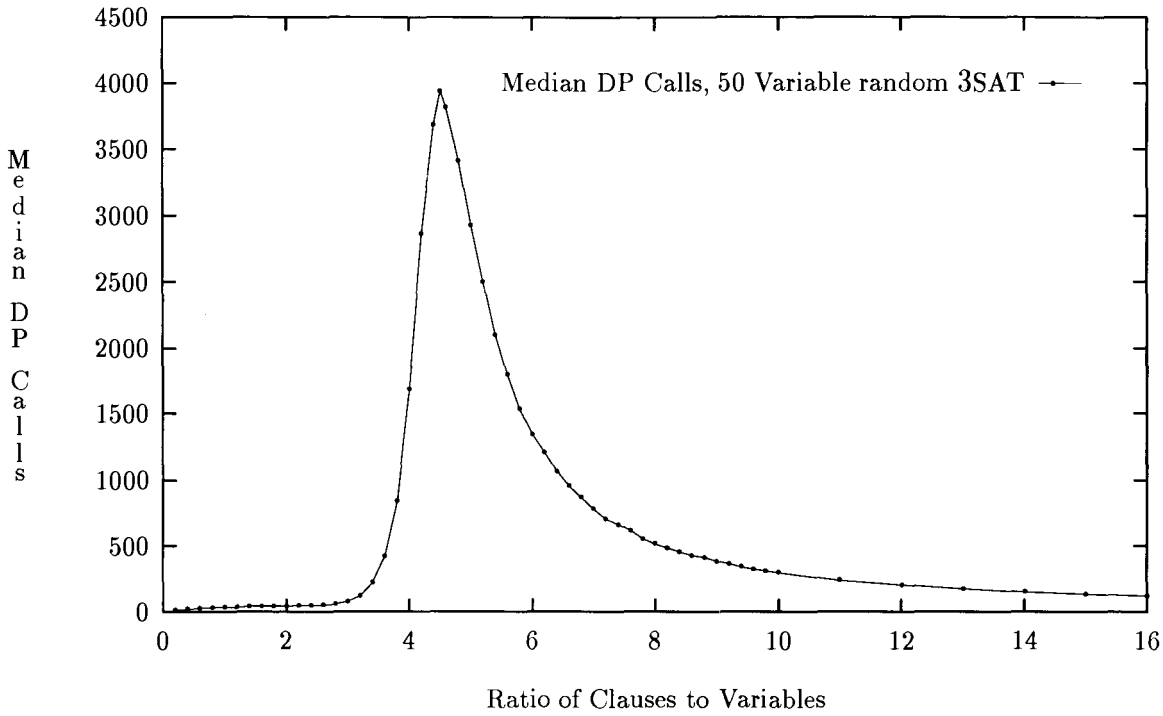

Figure 4.2: Median DP calls for 50-variable random 3SAT.

varied from 1/5 to 50. Figure 4.2 shows the median number of calls for DP to test these formulas (i.e., to find a satisfying assignment or determine that none exists), as m/n is varied from 1/5 (10 clauses) to 16 (800 clauses). The curve shows a clear unimodal pattern, with a sharp peak in average difficulty at about $m/n = 4.5$, dropping off quickly on either side of that point. This drop-off is not symmetrical about the peak, but skewed in the positive (more clauses) direction. At the peak of the curve, the median number of DP calls to test a formula is 3,942, whereas the median number of calls is about 40 for most of the region to the left of the peak, and several hundred for the region to the right of the peak. The value to the right continues decreasing monotonically beyond $m/n = 16$, dropping to below 40 at $m/n = 50$.

Thus, the most basic result is that we expect most formulas at about a ratio of 4.5 clauses-to-variables to be much harder (for DP) than those at significantly higher or lower ratios. This area, where expected difficulty increases by a factor of more

than several hundred times, will be referred to as "the hard region". The existence and location of this hard region plays a central role in the study and use of random formulas.

4.2.2 The easy-hard-easy pattern

Each clause added to a formula may be thought of as a constraint which eliminates a particular set of truth assignments as possible solutions to the instance. The ratio m/n may be thought of as a measure of the expected degree of constraint, with a low ratio indicating loosely-constrained formulas, and a high ratio indicating tightly-constrained formulas. A formula with very few clauses (relative to the number of variables) may be thought of as "under-constrained", and we might expect such a formula to have very many solutions.

Under-constrained formulas should be easy to solve, because only a very small part of the search space must be covered before a solution will be found (although expected number of solutions alone does not seem to account for the ease – see Section 4.2.5). In contrast, a formula with very many clauses, (relative to the number of variables), may be thought of as "over-constrained", and we would expect such a formula to have no solutions. For a procedure like DP , setting a small number of variables should result in unit propagation turning up a contradiction, so that the search never proceeds very far down any branch of the search tree. For formulas with a moderate number of clauses, we would expect a small number of solutions, but a very large number of near solutions. That is, given an incorrect partial solution, very many variables would have to be set before discovering that it could not be completed to a solution. Thus, a very large proportion of the search tree would have to be covered, on average, before finding a solution (or determining there isn't one).

The theoretical results show that, in the limit as n goes to infinity, formulas with extremely low clause-to-variable ratios should almost all be satisfiable, and probably be easy to solve. Formulas with very high clause-to-variable ratio should almost all be unsatisfiable, and are known to take exponential time for the Davis-Putnam Procedure (Chvátal and Szemerédi 1988). In between, there clearly must be some

sort of transition, but the exact nature of this transition (i.e., its location and shape) is not known.

Obviously, if clauses are added in a non-random manner, then the difficulty of the formula will depend on this method (i.e., adding redundant clauses should not make the formula much harder). However, one might expect that adding random clauses would result in a smooth shift from 'probably satisfiable' to 'probably unsatisfiable', and that the hard region should occur within this transition region.

4.2.3 Proportion of Satisfiable Formulas

Examination of the proportion of instances which are satisfiable at a given value of m/n confirms this intuition. The lower part of Figure 4.3 shows the proportion of 50-variable random 3SAT formulas which are satisfiable, for m/n varying from 2 to 8. The upper part of Figure 4.3 shows the same curve as Figure 4.2, but with the range of ratios restricted to 2–8, for comparison. As predicted by the argument based on intuition, for m/n ratios much lower than the hard point, almost all formulas are satisfiable, whereas for ratios much above the hard point, almost all are unsatisfiable. There is a smooth transition from almost all satisfiable to almost all unsatisfiable which corresponds closely to the hard region. This "transition region" would seem to be of some theoretical interest, as it is strictly a property of the formulas themselves, whereas the hard region is a property of the formulas in combination with the procedure used. The peak point on the "hardness" curve corresponds roughly to the point where about 40% of the formulas are satisfiable. Note that in (Mitchell, Selman, and Levesque 1992) the peak is placed at the 50%-SAT point. This difference is attributable to a small difference in the version of the Davis-Putnam Procedure used (as described in Section 4.1.1), and in general any change in procedure may be expected to shift the peak point slightly. Although the exact peak may shift small amounts, based on the intuitive explanation of the hard region given above, presumably the hard region and the transition region are inherently tied together. However, none of the theoretical work on random SAT addresses this issue directly.

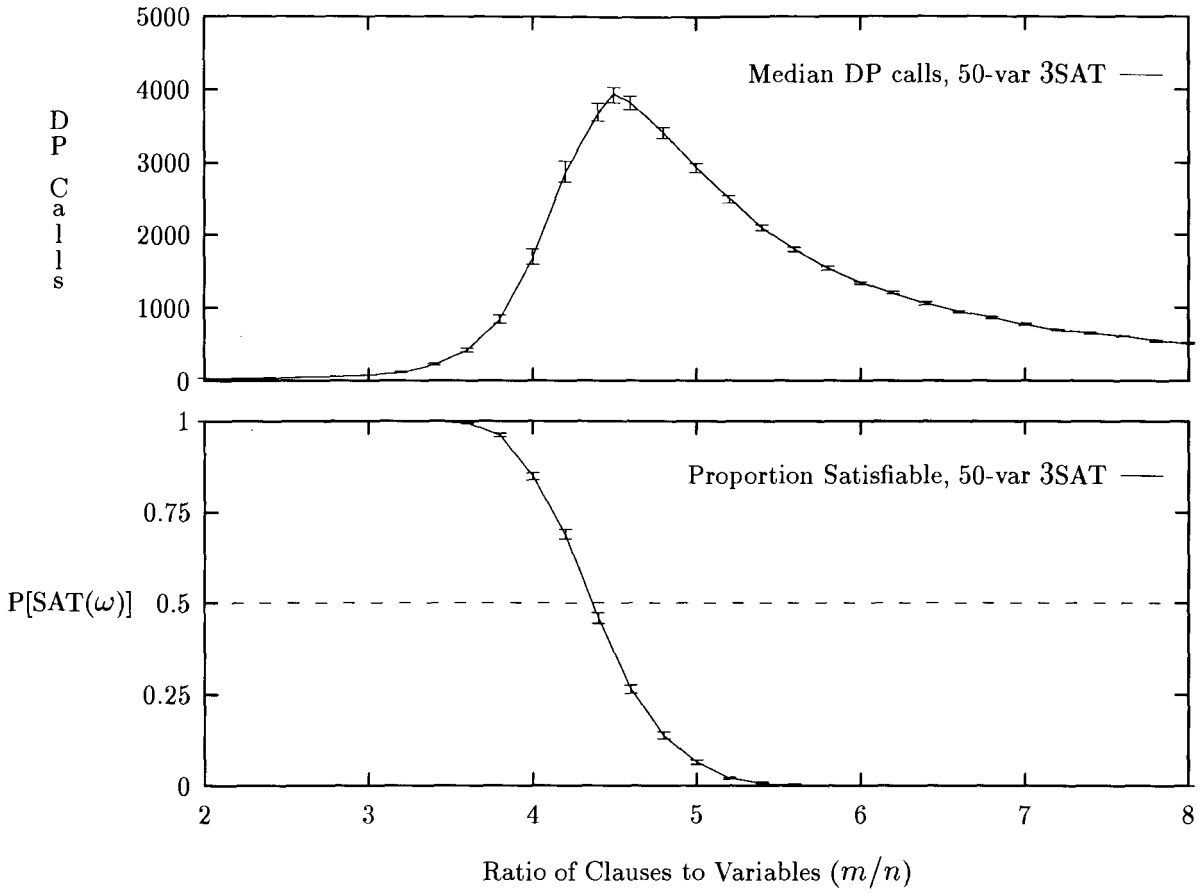

Figure 4.3: Satisfiable proportion of random 3SAT formulas.

4.2.4 Factoring Satisfiable and Unsatisfiable Subsets

The median values shown in Figure 4.2 are for *all* formulas of a given size. It seems natural to ask if there is a difference in the number of DP calls required for satisfiable and unsatisfiable cases at any given ratio – especially in light of the relation between the transition region and the hard region. The lower graph of Figure 4.4 shows three curves. One is the same curve as in Figure 4.2, while the upper curve is the median number of DP calls for unsatisfiable cases only, and the lower curve is the median number of calls for satisfiable formulas only. The composite curve is approximately the weighted average of the other two, with the weight determined by the proportion

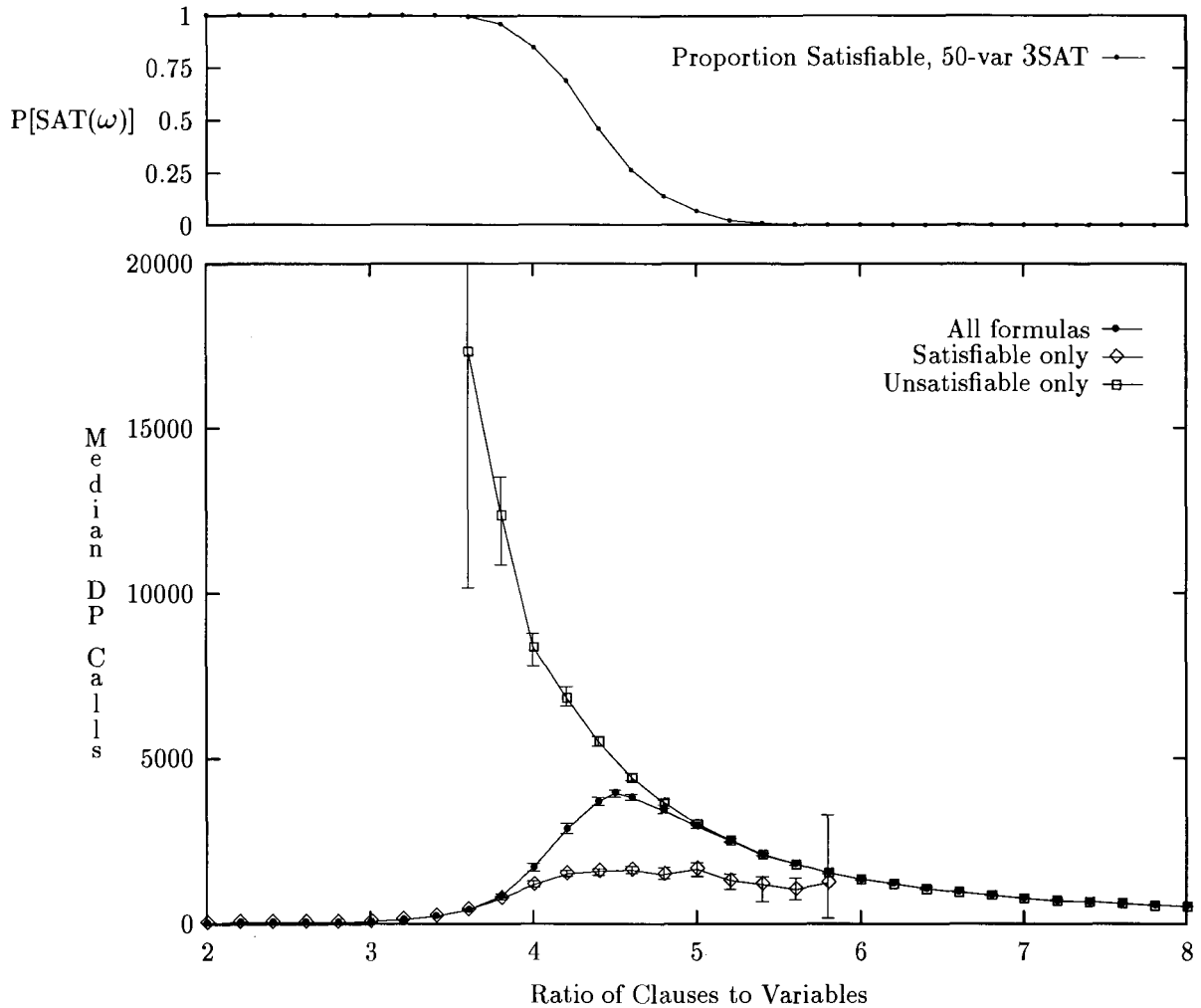

Figure 4.4: Median DP calls for random 3SAT: satisfiable *vs.* unsatisfiable.

satisfiable[3]. The points on the upper and lower curves are the medians from the satisfiable or unsatisfiable subsets of a sample of 5000, and thus represent smaller sample sizes. However the errorbars show that, although there are a couple of points with wide ranges of confidence, we can still be quite confident of the basic pattern.

The peak on the hardness curve occurs where about 60% of formulas are unsatisfiable, and these formulas are quite hard to show unsatisfiable. The hardest formulas

[3]For mean values, the composite is exactly the weighted average of the two subsets, but for medians this need not be so. For fairly smooth distributions, like the current one, the approximation is not far off.

found are those (rare) unsatisfiable formulas in the region where almost all formulas are satisfiable. Presumably these formulas are hard because they are very loosely constrained, even though unsatisfiable. Since an inconsistency would be found only after assigning values to very many variables, a very large search tree is covered before eliminating all possibilities. The satisfiable formulas in the region where almost all formulas are unsatisfiable are just slightly easier than the unsatisfiable ones at the same point, as would be expected because only a partial search is required for the satisfiable formulas. The easiest formulas are the very underconstrained satisfiable formulas, which are discussed in the following section.

4.2.5 Number of Satisfying Assignments

One might expect an under-constrained formula to be easy to solve simply because there are very many solutions and thus a high probability of stumbling over a solution early in the search, no matter what (reasonable) search strategy is used. For example, with 1 clause we have $2^n - 2^{n-3}$ solutions and 2^{n-3} non-solutions, so clearly as $n \to \infty$, the probability of a random truth assignment being a solution goes to 1. (In this example however, the ratio of clauses-to-variables is not held constant, but decreases as n increases.) To see if the ease of testing formulas below the hard region can be accounted for by large numbers of solutions, a modified version of DP was used which finds all the solutions to a satisfiable formula (rather than stopping with the first found, as is the usual practice). Since finding all solutions for extremely underconstrained 50-variable formulas was prohibitively time-consuming, data are given for 25 variable formulas only.

The interesting measure is the proportion of truth assignments which constitute solutions. The upper graph of Figure 4.5 shows the median number of DP calls to test 25-variable random 3SAT formulas (using the original procedure which stops upon finding one solution). The lower graph shows the mean proportion of the 2^{25} possible truth assignments for 25-variable formulas which satisfy a random 3SAT formula. The region with $m/n < 1$ does in fact have a very large number of solutions, so that we would expect always to find one quickly (except with a perverse search procedure).

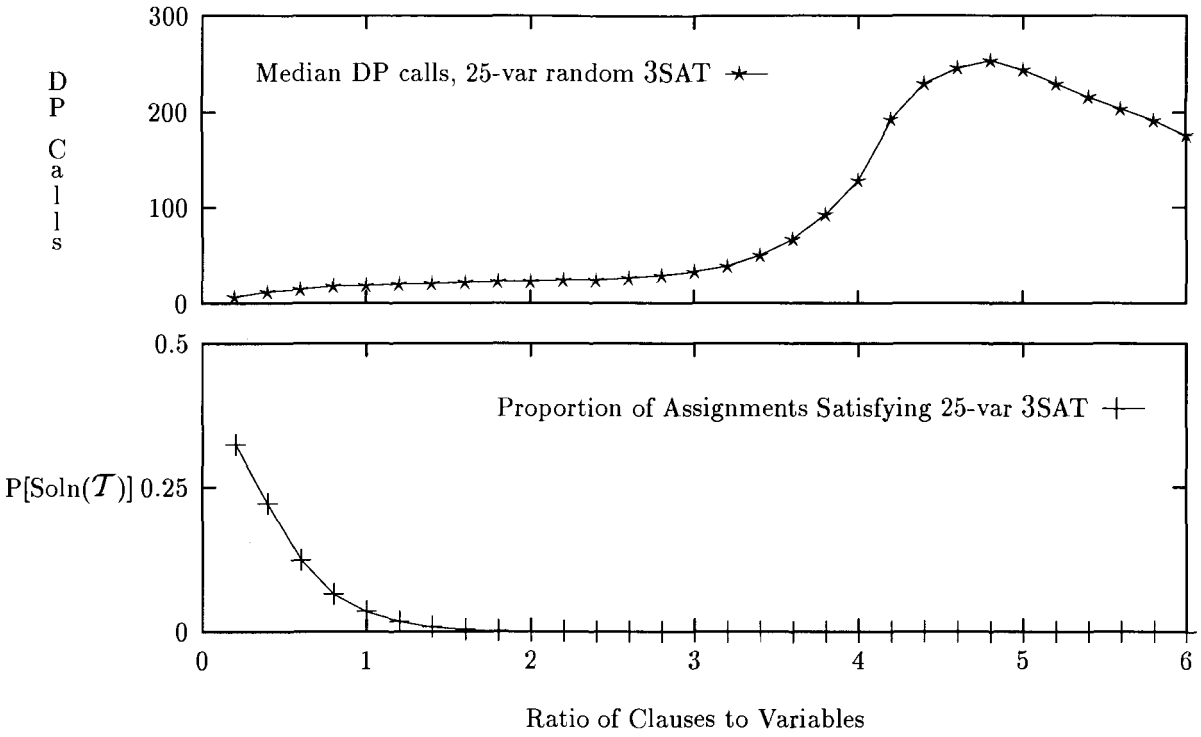

Figure 4.5: Average number of solutions for satisfiable random 3SAT.

However, 25-variable random 3SAT formulas are very easy (requiring less than 33 DP calls on average) for all $m/n < 3$. As low as $m/n = 2$, only about one in 1,000 assignments is a solution, but DP finds solutions for these formulas with less than one backtrack on average (and almost never more than 20). Thus, the ease of solving the formulas in the range $2 < m/n < 3$ cannot be accounted for by a high number of solutions alone. Preliminary experiments suggest that this also holds for $n > 25$.

From observation of traces of the execution of DP on the formulas in the range $2 < m/n < 3$, there is a great deal of unit propagation, and little backtracking. Thus, one may conjecture that although there are not many solutions (relative to the number of non-solutions – the actual number of solutions may still be quite large!) there are few variables which can take on only one value in a solution (i.e., few variables u for which ω logically implies either u or $\overline{u}$). Rather, the values assigned to the first few variables don't matter, because unit propagation succeeds in setting almost all

other variables in a manner consistent with completing that partial assignment to a solution.

4.2.6 Varying n in Random 3SAT

Formulas with 25 or 50 variables are convenient for experimentation of the kind done here, because they can be tested quickly enough to perform tests with very large sample sizes, and over a wide range of values of the other parameters. However, our real interest is in being able to solve formulas large enough to represent real problems, so we must be more concerned with the rate of growth of effort with n than with the absolute effort for any small fixed n.

To examine rate of growth for random 3SAT, tests were run on 75 variable formulas, as well as the 25 and 50 variable formulas already discussed. Figure 4.6 is a plot of the median number of DP calls for 25, 50 and 75-variable formulas, for the range of ratios 2–8. The upper graph uses a linear scale on the y-axis, showing clearly the 50-variable curve. The large peak in the 75-variable curve extends well above the graph, while the 25-variable curve is merely a small bump at the bottom. In a graph scaled to show the peak of the 75-variable curve, the 25-variable curve is indistinguishable from the x-axis. This very rapid growth rate of the peak with n creates a general problem with the presentation of this kind of data, since it is often impossible to visually compare more than two significantly different values of n directly.

The lower graph of Figure 4.6 displays the same data but with a logarithmic scale on the y-axis, so that all three curves can be seen clearly. Qualitatively, all three curves are quite similar in shape. One difference is that the peak shifts slightly to the left (lower ratios) as n is increased. A second difference, which is hard to spot in either graph, is that the peak becomes more pronounced as n is increased; for 75 variables, the peak number of DP calls is about 1000 times the number at $m/n = 2$, whereas for 25 variables this difference is only a factor of about 100. (The peak of the 75-variable curve is about 500 times the peak of the 25-variable curve).

While the graphs are helpful in getting an overall picture of the patterns, it is hard to clearly see the rate of increase of DP calls, because in one case we cannot see all

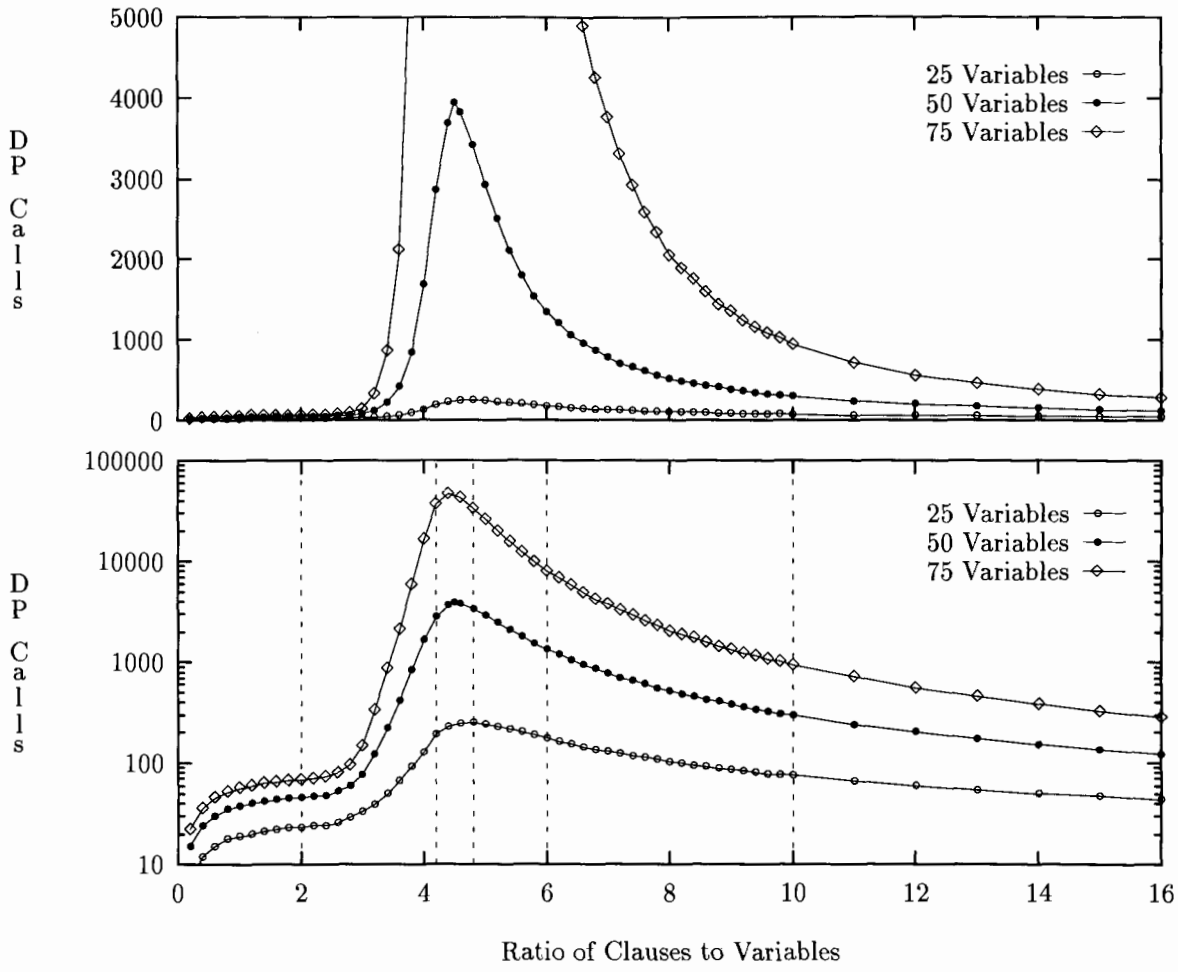

Figure 4.6: Median DP calls for random 3SAT: varying n.

the curves clearly, and in the other case perceptions are distorted by the logarithmic scale. Table 4.1, gives the numerical values for the median number of DP calls at various ratios, for the same formulas used to make up the curves just discussed. (The vertical dashed lines in the lower graph of Figure 4.6 indicate the ratios which appear in this table.) At a glance, the growth rates seem to be linear for ratios much below the transition region, and exponential at the peak. Above the transition region the numbers are less compelling. The growth of difficulty with n will be discussed more fully in Section 5.2.

Figure 4.7 shows the proportion of satisfiable formulas for random 3SAT with

ratio	25-vars	50-vars	75-vars
1	19	46	57
2	23	38	68
3	33	76	147
4.2	192	2865	37054
peak	253	3822	46915
4.8	253	3415	33124
6	176	1342	7997
10	75	299	942

Table 4.1: Effect of increasing n on DP calls in random 3SAT

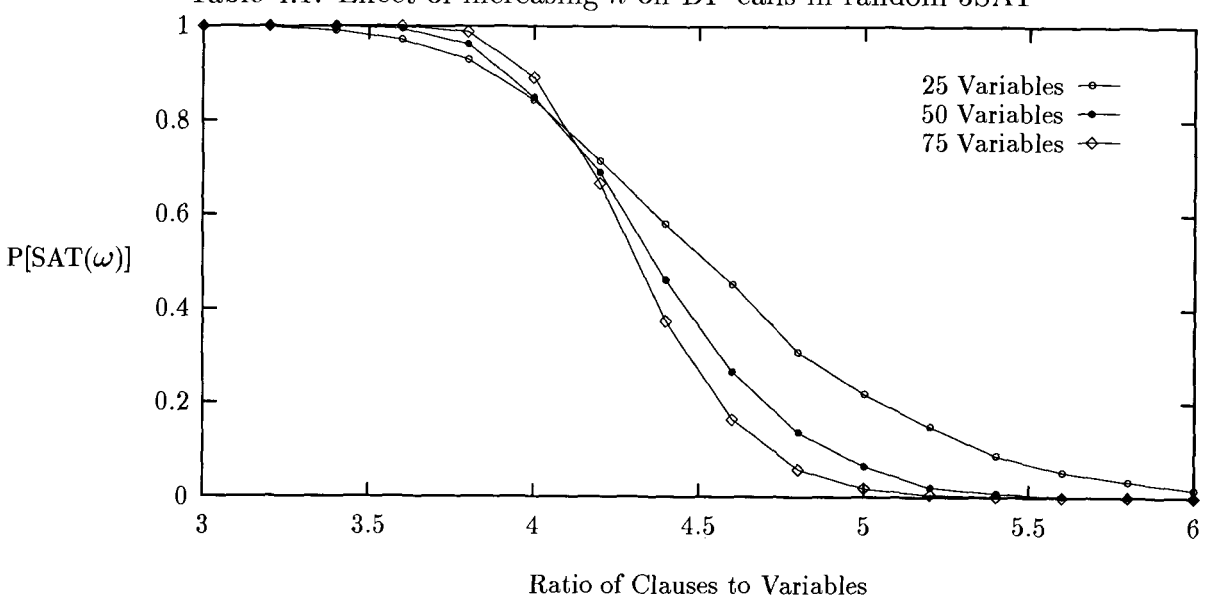

Figure 4.7: Satisfiable proportion of random 3SAT: varying n.

25, 50 and 75 variables. The transition region becomes narrower (that is, the slope of this curve in the transition region increases) with increasing n. Thus, the ratio corresponding to, say, 1/3 of formulas being satisfiable, shifts lower as n increases. This shift corresponds to the shift in the peak of the hardness curve with increasing n. There are two interesting open questions about the nature of the transition region as n goes to infinity:

- What is the exact position of the transition region?
- Does it become a step transition, or retain a finite slope?

In (Mitchell, Selman, and Levesque 1992) we suggested that the transition point ratio was 4.3. Using very large sample sizes, formulas up to 220 variables, Crawford and Auton (Crawford and Auton 1993), using linear regression, estimated the point to be 4.24 (actually, they estimated the 50%-satisfiable point to be at $4.24n + 6.21$). The graph here shows an interesting phenomenon in that the curves for different n seem all to pass through a point at about $m/n = 4.1$ and $P[SAT(\omega)] = 0.77$. Conjecturing that this "pivot point" is in fact an invariant with n, we would expect the transition point ratio to be very close to 4.1. A similar effect was also observed by (Larrabee and Tsuji 1992) with up to 140 variables, although they suggested the point was at 4.2. However, their data was based on smaller sample sizes of 500. The confidence bounds on the values from the present data are quite tight, so that it would be very surprising if the figure of 4.1 was off by more than a few per cent. Of course, this does not constitute evidence that the curves for larger n do not shift outside this range.

In terms of ratio of clauses-to-variables, the width of the transition region becomes narrower as n is increased. However, we can also calculate the width in terms of the actual number of clauses that must be added to shift from one side of the region to the other. The width of the transition region was estimated by taking the number of clauses at which 80% of formulas are satisfiable and the number at which 20% are satisfiable, using linear interpolation between the nearest points shown on the graph. The estimated widths are 24 clauses for 25 variables, 31 clauses for 50 variables and 50 clauses for 100 variables. This is not inconsistent with the transition being asymptotically a step function when in terms of clause-to-variable ratio, but does add an additional perspective to our view of the data.

It would be quite interesting to further investigate both these aspects of the transition region for formulas with much larger n and with careful error analyses.

4.2.7 Varying k in Random kSAT

Although any kCNF formula can be easily converted to an equivalent 3CNF formula, it is still interesting to look at values of k other than 3. Some problems will be more

naturally represented with other clause lengths. Even if transformation to 3SAT is assumed as a strategy for solving these other problems, the distribution of 3SAT formulas resulting from the mapping of random formulas with longer clause lengths will probably be quite different from random 3SAT. This is an issue which may well be worth investigating.

To look at the effect of clause length on the general patterns, tests were performed with $k \in \{2, 3, 4, 5\}$, for 25 and 50 variables. Figure 4.8 shows median number of calls to test the 25-variable formulas, and the proportion of them which are satisfiable, for m/n in the range 0-30. The middle graph shows the median DP calls for 25 variables, with a linear scale, so the increase in hardness as k increases can be clearly seen. (The curve for $k = 2$ is indistinguishable from the x-axis.)

The upper graph of Figure 4.8 shows the proportion of 25-variable formulas which are satisfiable for the same four values of k. Not surprisingly, in every case the transition region is in the same position as the hard region, with the peak of the hardness curve being where a little fewer than half the formulas are satisfiable. The transition point for random 2SAT is just above 1, and in fact this is known analytically to go to 1 as n goes to infinity (Goerdt 1992),(Chvátal and Reed 1992). As would be expected, the patterns are the same for 50-variable formulas, with attendant increases in difficulty.

The manipulation of k has two main effects. Increasing k shifts the transition region and the hard region to higher clause-to-variable ratios, and also increases the number of DP calls to test formulas at ratios near and above the transition region. Both the peak difficulty and the position of the transition region appear to change exponentially with k. That the shift of transition region is exponential in k follows from the lower bound given by (Chao and Franco 1990).

The lower graph shows these same curves plotted with a log scale on the y-axis. In this graph all four curves can be seen and compared, and in particular one can observe an interesting aspect at the low clause-to-variable ratios. That is, the four curves all converge, so that for ratios below $m/n = 2.5$ they are the same. The "shoulder" at the left side, where number of steps drops quickly to near zero, can be accounted for by the fact that for ratios much below 1 we do not expect all the variables to occur

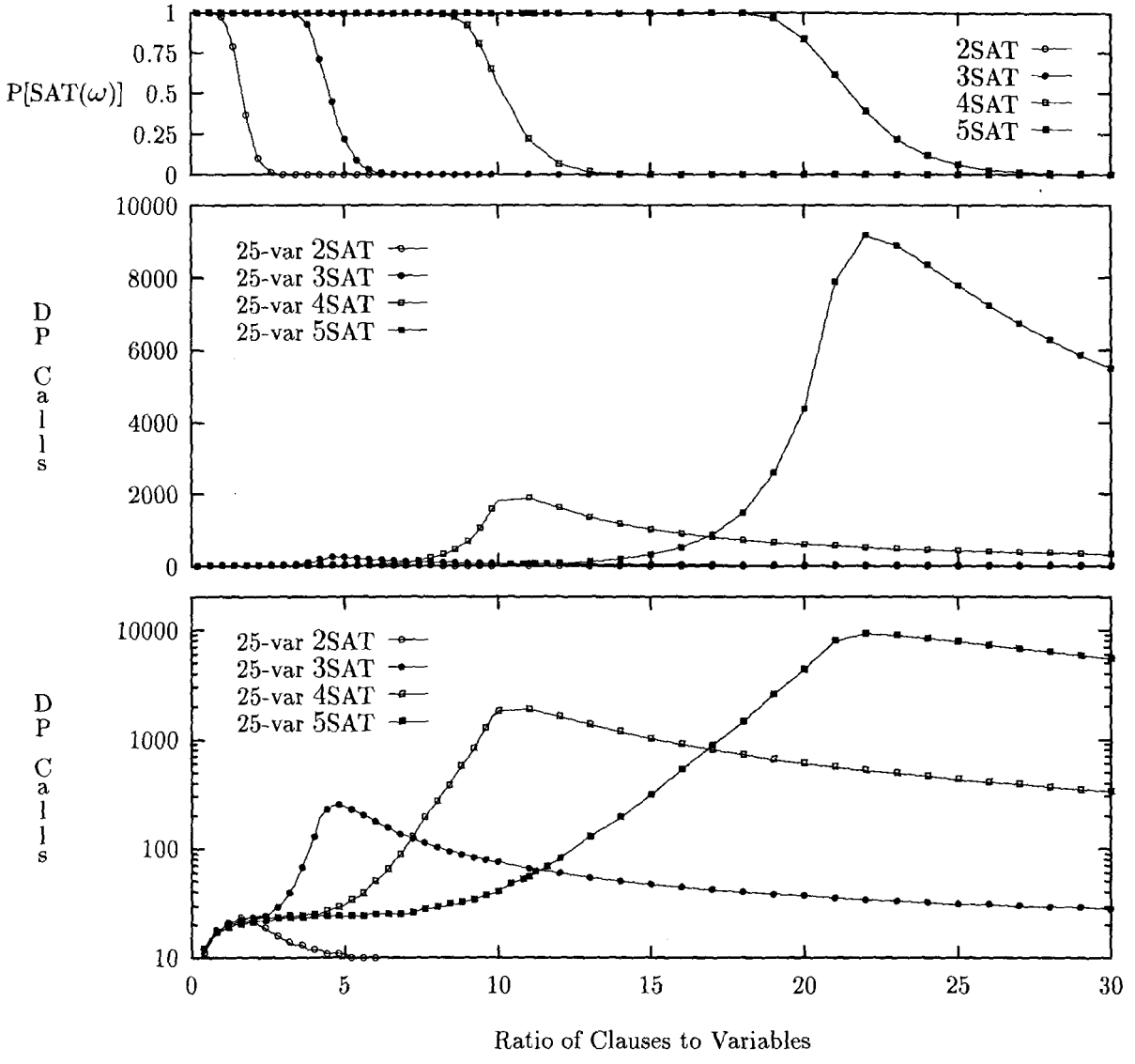

Figure 4.8: DP Performance on random kSAT formulas: varying k

in a random formula. A m/n decreases, fewer and fewer variables will be mentioned, so in fact these formulas should be quite easy. Another interesting aspect is that, for all of the curves except $k = 2$, there is a "plateau" to the right of the shoulder, which leads up to the peak. In the case of $k = 2$, which is the only case which is not NP-Complete, the plateau and the peak are both missing, and above the shoulder difficulty drops off quickly to near-zero as m/n is increased.

This correspondence of the left parts of the curves seems to hold also for larger

numbers of variables. This correspondence indicates that these formulas are of essentially the same average-case difficulty for DP . Since 2SAT, and therefore random 2SAT, is solvable in linear time in the worst case, it would seem to follow that all the formulas in this region are easy.

Chapter 5

Issues and Further Experiments

Chapter 4 looked at how changing the parameters to the random kSAT family of formulas affected the average number of DP calls to test these formulas and the proportion of the formulas which are satisfiable. This chapter extends this investigation to additional distributions, discusses the issue of hardness as n increases, and reviews previous experimental work in light of the current findings. Section 5.1 confirms that the same kinds of patterns shown for random kSAT also hold for random $\mathcal{E}k$SAT, the most used family of distributions in empirical work. It also substantiates the claim, made in (Mitchell, Selman, and Levesque 1992) but based on very small experiments, that the random $\mathcal{E}k$SAT formulas are too easy to be suitable as test instances to demonstrate the efficient performance of SAT testing programs. Section 5.2 attempts to strengthen this argument, directly comparing the performance of DP on random kSAT and random $\mathcal{E}k$SAT, and looking at this performance for formulas with up to 500 variables. Section 5.3 reviews many previous reports of experiments aimed at demonstrating effective SAT testing programs. This section is organized by distribution family, for each distribution showing experiments where either very easy formulas were used, or where failure to appreciate the properties of random SAT formulas lead to unwarranted inference from the data provided, or showing how this makes interpretation of data difficult.

5.1 Testing Random $\mathcal{E}k$SAT with DP

The formulas most frequently used and studied in the literature are variations of Goldberg's binomial clause length family, in spite of the warning presented by analytic results such as those in (Franco and Paull 1983) and later. As was argued in Section 3.3, from an empirical point of view random $\mathcal{E}k$SAT is the most interesting family of distributions based on the method of generating formulas used by Goldberg. As (Hooker and Fedjki 1989) show, a very simple analysis demonstrates that some form of transition region must exist, and a reasonable argument can be made that an easy-hard-easy pattern should also be observed. The objective of this section is to confirm these predictions and to see how the patterns for random $\mathcal{E}k$SAT compare with those observed in Chapter 4 for random kSAT. Formulas from random $\mathcal{E}k$SAT are examined with parameters ranging over the same sets of values that were used in Chapter 4 for random kSAT. With the random $\mathcal{E}k$SAT distributions, k can take on non-integer values. However, 3 is the smallest integer value of k for which the problem remains NP-Complete[1]. As with the random kSAT family, most attention is paid here to the case where k is set to 3. Data points again represent the median number of DP calls from a sample of 5000 pseudo-randomly generated formulas.

The lower graph in Figure 5.1 shows the median number of calls required by DP to test 50 variable random $\mathcal{E}$3SAT formulas (i.e., to find a satisfying assignment or determine that none exists). As with the fixed clause length formulas the curve shows a clear uni-modal easy-hard-easy pattern. The upper graph of Figure 5.1 shows the proportion of formulas which are satisfiable for 50-variable random $\mathcal{E}$3SAT. The transition from almost all satisfiable to almost all unsatisfiable again occurs over a relatively narrow region, and as expected this transition region corresponds to the hard region in the lower curve. In this case, the peak of the DP calls curve occurs at about the same ratio for which 1/3 of the formulas are satisfiable.

Despite the general similarity of the patterns for corresponding curves for random 3SAT and random $\mathcal{E}$3SAT, there are several differences. The peak of the hard area

[1]Since we allow no clauses shorter than 2, if $k = 2$ we have 2SAT, which is solvable in a linear time (Aspvall, Plass, and Tarjan 1979; Monien and Janssen 1977)

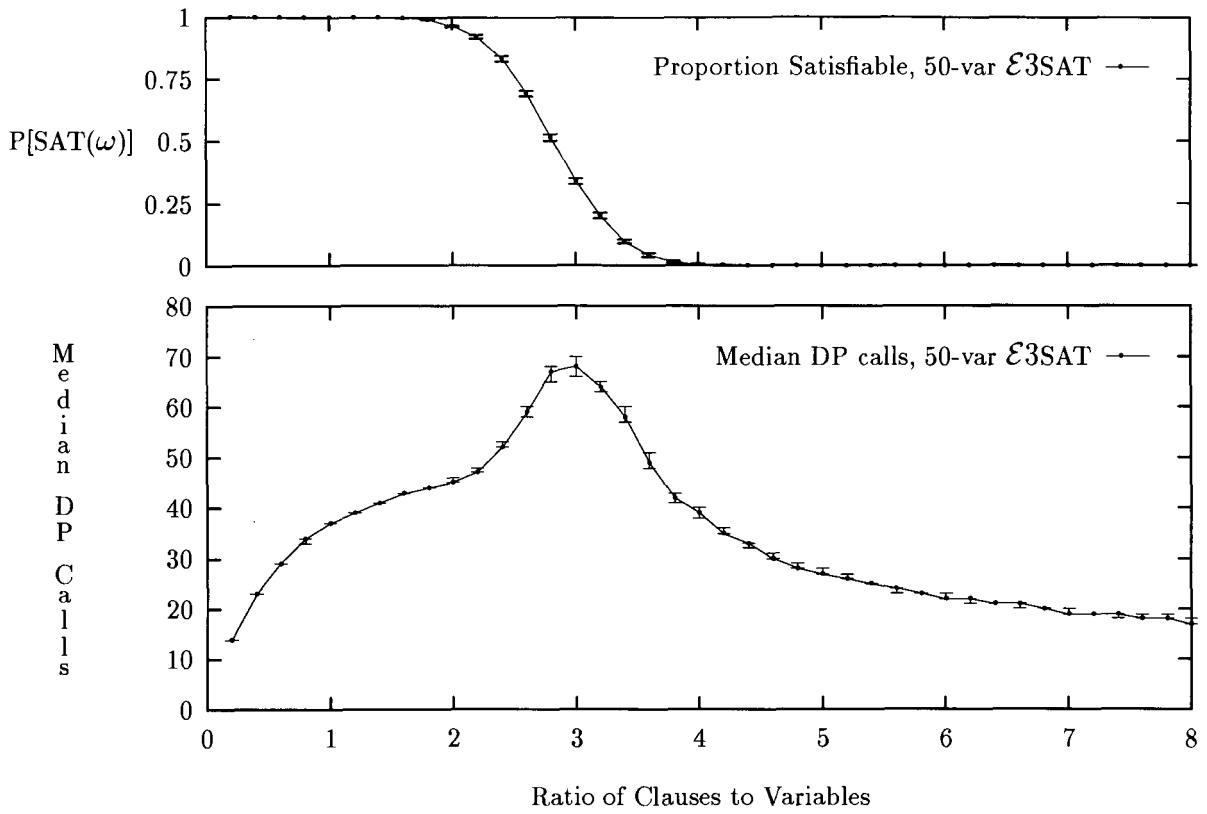

Figure 5.1: Median DP calls for 50 variable random $\mathcal{E}$3SAT.

occurs at a lower ratio of clauses-to-variables; at about $m/n = 3$ rather than $m/n = 4.5$. The number of calls at the peak of this curve is only about twice the number of the areas adjacent to but clearly outside of the hard area, whereas for random 3SAT, this difference was a factor of nearly 100. Also, for the region to the right of the hard area the number of DP calls drops off very quickly to below the level of the "plateau" area to the left of the peak. For random 3SAT, the difficulty to the right of the peak remains substantial until quite high ratios are reached. Most significantly, the actual numbers of DP calls for the two distributions are vastly different, with a 50-variable random $\mathcal{E}$3SAT peak of about 68 in contrast to about 3900 for random 3SAT. Based on the peak values for these graphs, we might suspect that these formulas are too easy, on average, to provide reasonable test materials. To determine if this is merely an effect of very small n, Section 5.2 examines the rate of growth of difficulty of random

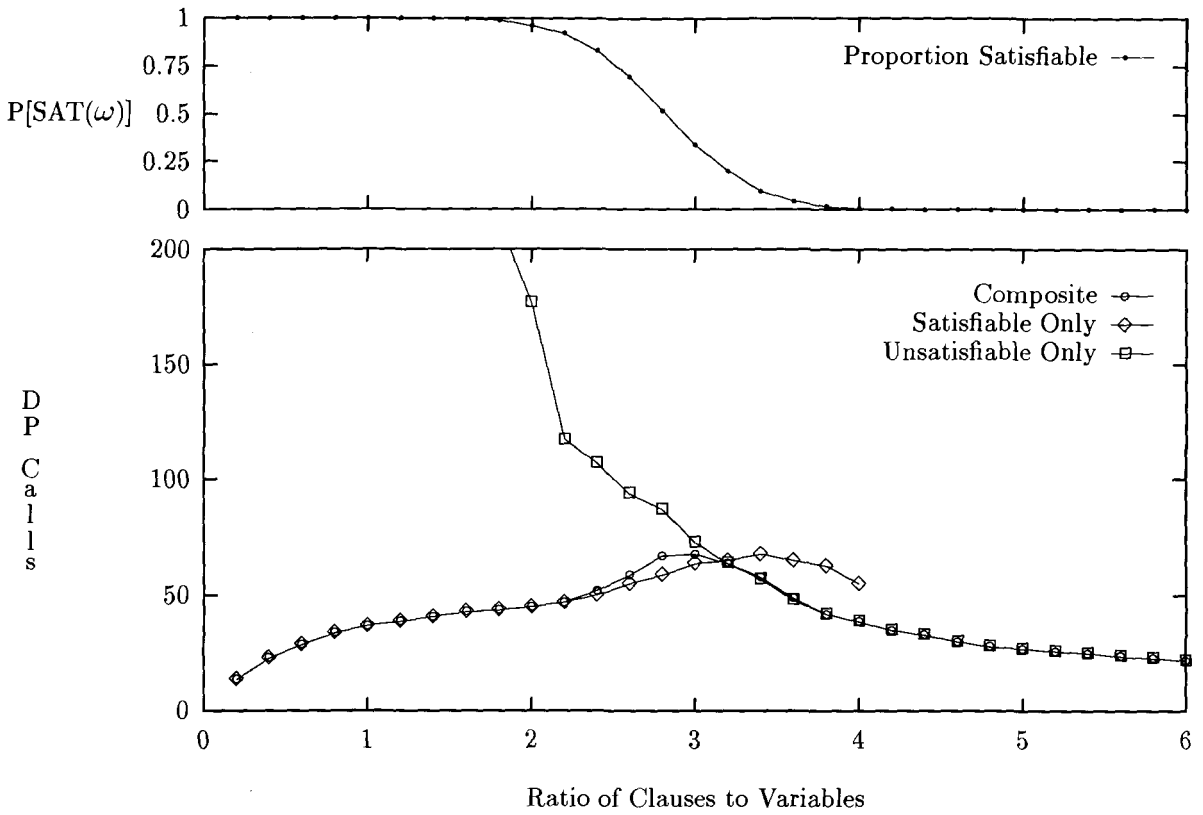

Figure 5.2: 50 variable random $\mathcal{E}$3SAT, satisfiable *vs.* unsatisfiable.

$\mathcal{E}$3SAT and random 3SAT.

Figure 5.2 shows the median number of DP calls for 50-variable random 3SAT, with satisfiable and unsatisfiable sub-sets factored out. The composite curve, for all formulas, is the same curve shown in the bottom section of figure 5.1, and is again approximately the weighted mean of the values for the satisfiable formulas alone and the unsatisfiable formulas alone. As with random 3SAT, the hardest formulas are those rare unsatisfiable formulas occurring at relatively low clause-to-variable ratios. However, the hardest satisfiable formulas occur in the region where almost all formulas are unsatisfiable, unlike random 3SAT where they fell near the peak on the composite curve. These hardest satisfiable formulas they are harder than the unsatisfiable formulas at the same ratio, which is somewhat counter-intuitive. Since the satisfiable and unsatisfiable formulas at a given ratio would be expected to be

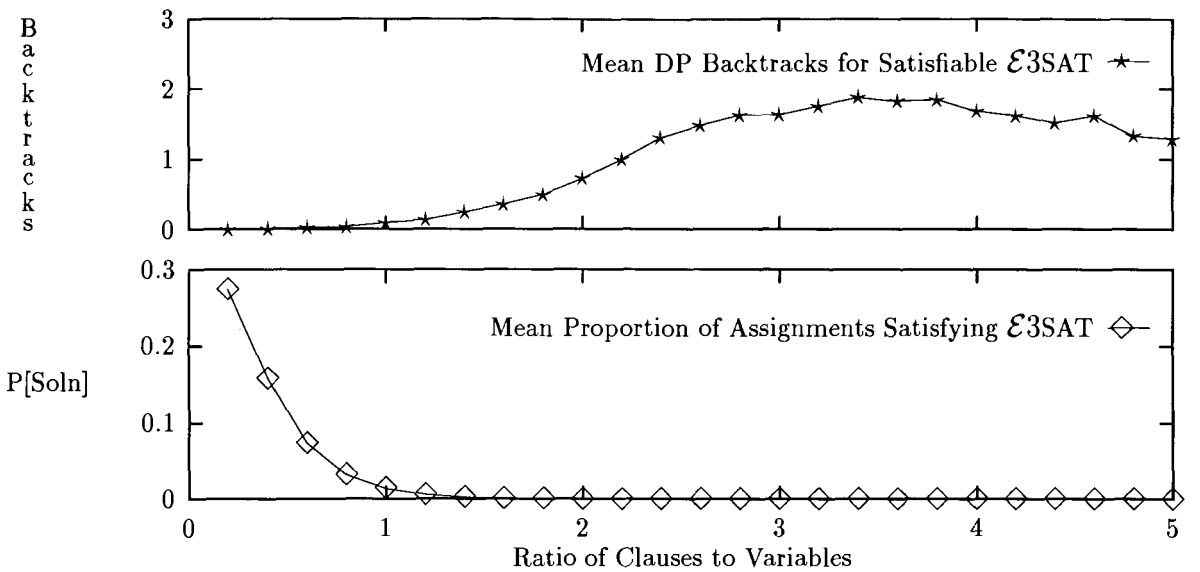

Figure 5.3: Average number of solutions for satisfiable random $\mathcal{E}$3SAT

quite similar in other respects, one would normally expect the satisfiable formulas to be somewhat easier, since only a partial search of the assignment is required to find a solution whereas a complete search is required to show there isn't one.

In (Hooker and Fedjki 1989) it was argued that the formulas from this distribution with ratios below the transition region would be very easy because the expected number of solutions for these problems is exponential in n. However, since the number of truth assignments also grows exponentially, this argument is not convincing alone. The proportion of truth assignments which satisfy satisfiable 25-variable random $\mathcal{E}$3SAT formulas is shown in the lower graph of figure 5.3. For this distribution, the proportion of assignments satisfying a random formula is quite high at ratios below $m/n = 1$, but low at ratios much above that. At $m/n = 2.2$ the proportion has fallen below 1 in 10,000. However, these formulas require almost no search to find a satisfying assignment. The upper graph of figure 5.3 shows the mean number of backtracks required by DP to find a satsifying assignment to a satisfiable random $\mathcal{E}$3SAT formula. As can be seen, it almost always require less than 2 backtracks on average to show these formulas are satisfiable. Thus, the ease of solving them cannot

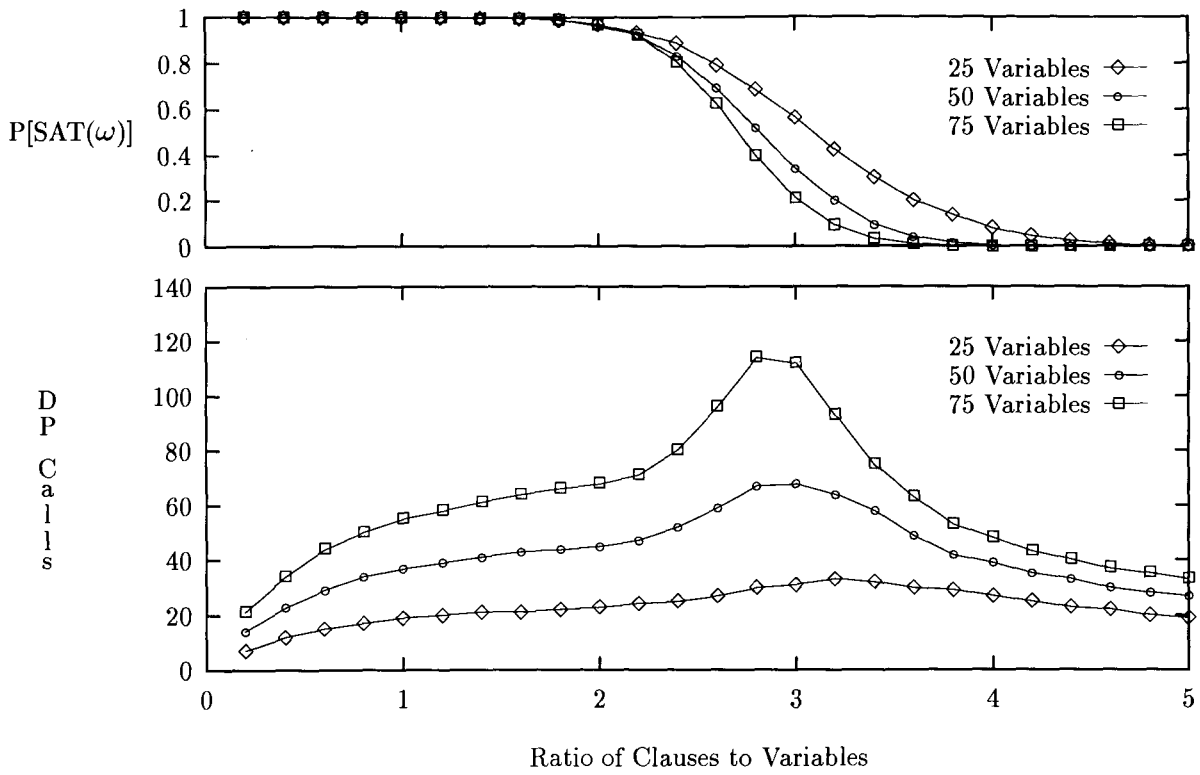

Figure 5.4: Median DP calls for random $\mathcal{E}$3SAT: varying n.

be attributed merely to their having a large number of satisfying assignments, except perhaps when $m/n < 1$.

Again, the effect of varying n is considered by comparing the median DP calls curves for 25, 50 ad 75 variable formulas. The lower graph of figure 5.4 shows the three curves of median number of DP calls. The qualitative shapes of the curves are again very similar, with the peak shifting slightly to the left as n increases. The peak does become more pronounced with 75 variables, but it is still only twice as hard as the easy region to the left of the peak, in marked contrast to the random 3SAT case, where the difference between the peak and adjacent regions becomes much greater as n is increased. The upper graph of figure 5.4 shows the proportion of satisfiable formulas for the three values of n. Again, as with random 3SAT, the slope of the transition region increases with increasing n. In this case, the left-shifting of the peak in the hardness curves corresponds to the shift in the region where about 1/3 of

formulas are satisfiable. In this graph, there is no evidence of the "pivot point" that seemed apparent in the corresponding graph for random 3SAT (figure 4.7).

Actual values from the curves of figure 5.4 are shown in table 5.1. Again, at very low ratios the increase in number of DP calls as n is increased appears very much to be linear (at least for such small n). For random 3SAT, though, the values at the

ratio	25-vars	50-vars	75-vars
1	19	37	55
2.8	30	67	114
peak	33	68	114
3.2	33	64	93
5	19	27	33
10	11	15	18

Table 5.1: Effect of increasing n on DP calls in random $\mathcal{E}$3SAT

peaks were obviously changing at very much above a linear rate. However, here there is no real evidence for faster than linear increase of calls with n, even at the peak or at higher ratios.

The remaining examination is for the effect of k on the random $\mathcal{E}k$SAT family. This is shown in figure 5.5. The upper graph shows the proportion of satisfiable formulas, and the lower graph the median DP calls, as k is varied from 2 to 5. The random $\mathcal{E}$2SAT curve is, of course, the same as the random 2SAT curve, because no clauses shorter than length 2 are allowed. As with the fixed clause length formulas, position of the peaks and transition points shift right with increased k, and the peak hardness also increases with k. The shift right is consistent again with change exponential in k. The increase in the peak hardness is less compelling.

5.2 Empirical Hardness of Random SAT

The previous section showed that, with some differences, the two families of distributions random kSAT and random $\mathcal{E}k$SAT exhibit quite similar qualitative patterns.

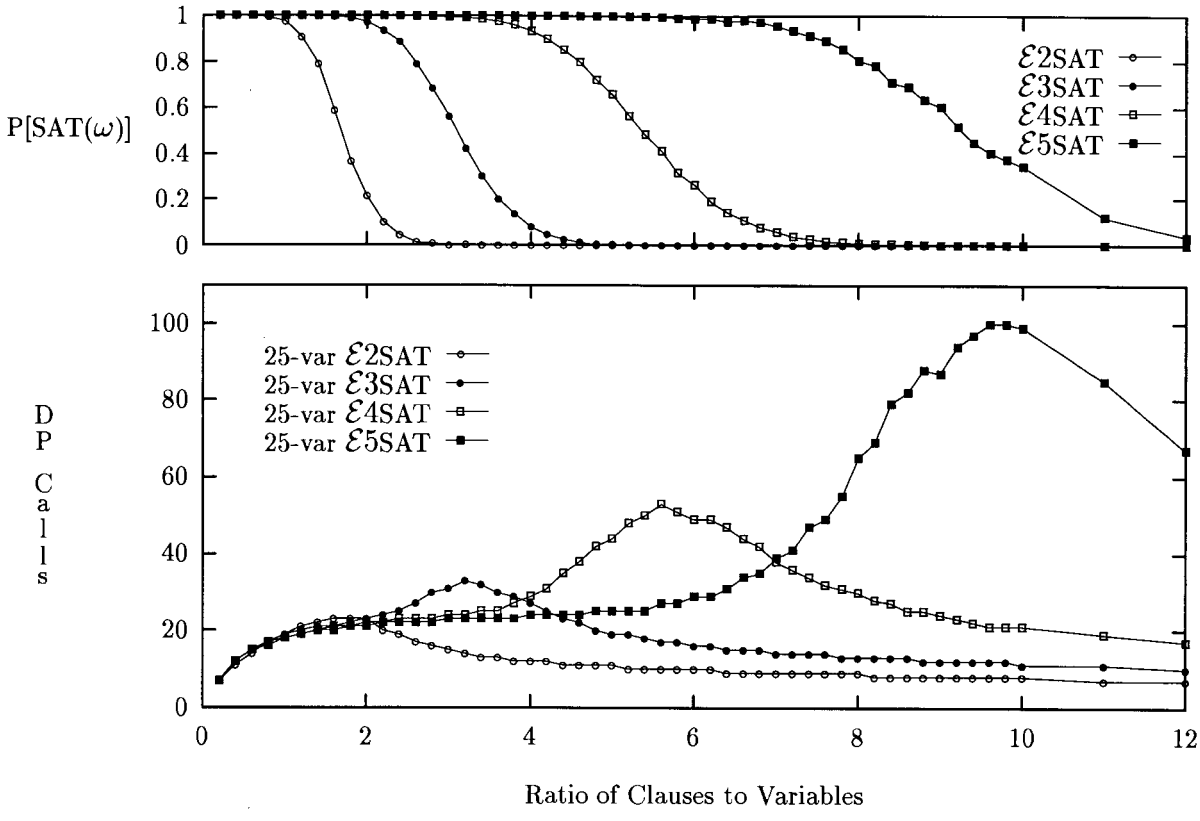

Figure 5.5: Median DP Calls for random $\mathcal{E}k$SAT formulas: varying k.

However, the random $\mathcal{E}k$SAT formulas are much easier (for similar numbers of variables, and similar clause lengths) than the fixed clause length formulas. In (Mitchell, Selman, and Levesque 1992), it was argued that the random $\mathcal{E}k$SAT formulas were probably not hard enough to be useful as test instances in evaluating SAT testing procedures. This argument, however, was based on experiments with small numbers of variables. In this section, further effort is made to substantiate this argument. The difficulty of random 3SAT and random $\mathcal{E}$3SAT are compared directly, for similar numbers of variables, and with additional attention to the increase in difficulty with increasing n. In discussing growth with n, I use a $\sim$ character to indicate functions where I ignore constants. For example, when I say "this curve is consistent with $\sim n^2$ growth", I mean that a function of the form $c_1 + c_2(n^2)$ can be fit to the curve with small error.

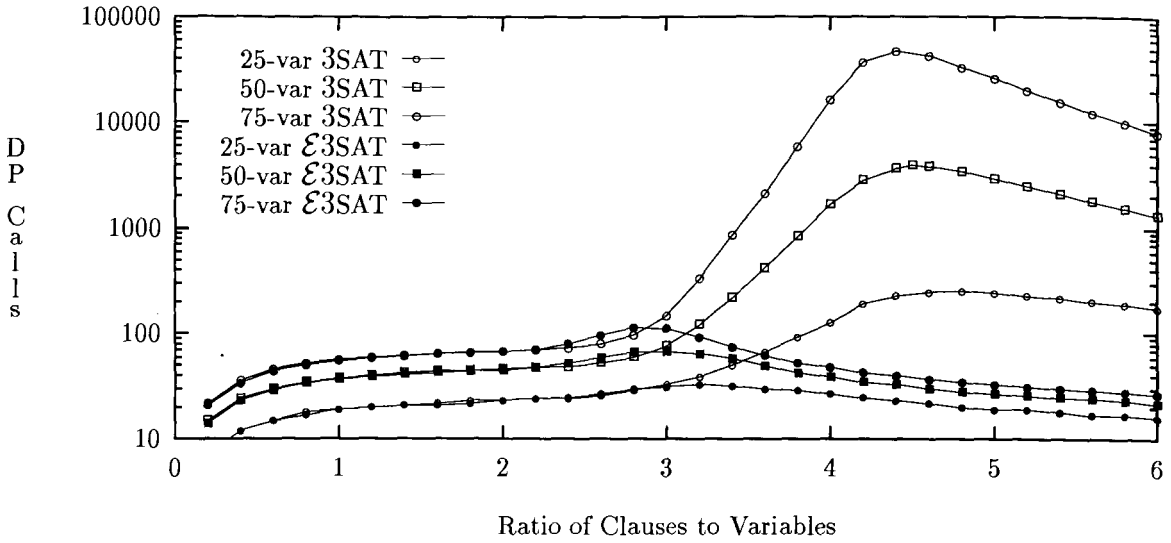

Figure 5.6: Comparing random 3SAT and random $\mathcal{E}$3SAT: DP calls with varying n.

Figure 5.6 shows the median calls curves for both families , for $n \in \{25, 50, 75\}$, with the clause-to-variable ratio varying from 1/5 to 6. The y-axis of this graph has a logarithmic scale to make it possible to display all the curves in one graph. Checking the scale on this axis, it is clear that the random 3SAT formulas are much harder than the random $\mathcal{E}$3SAT formulas, with the peak of the 75-variable random 3SAT curve being about 900-times that of the 75-variable random $\mathcal{E}$3SAT curve. Also, peak values for random 3SAT are increasing much faster. The most striking aspect of this graph, though, is that the curves for random 3SAT and random $\mathcal{E}$3SAT formulas with the same numbers of variables are essentially identical for all ratios below 2.4. In the range of 2.4 to 3.0, where the peak of the random $\mathcal{E}$3SAT curves appears, the curves deviate somewhat, but the random $\mathcal{E}$3SAT curves appear to be growing at no greater a rate than the random 3SAT curves at this point. At ratios above 3, the random $\mathcal{E}$3SAT formulas become much easier. The peak of the random $\mathcal{E}k$SAT curves correspond (both in terms of ratio and the median number of calls) to a region of random kSAT which is still quite clearly in the easy area. The difficulty of the random 3SAT formulas soars much higher above this pont, then decreases only gradually.

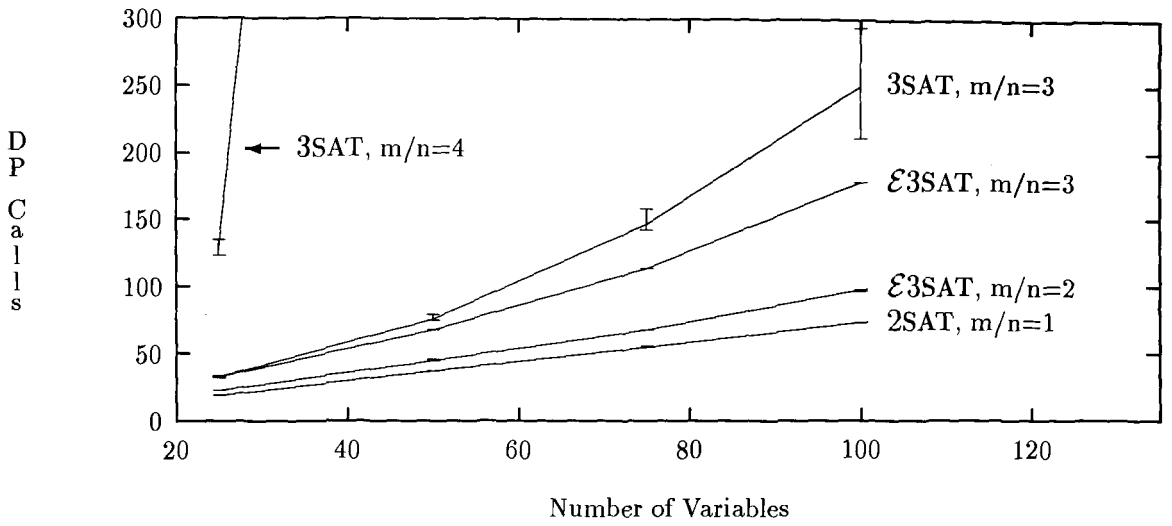

Figure 5.7: Growth of DP calls with n for various distributions

Figure 5.7 shows how the median number of DP calls changes as n is varied from 25 to 100, for particular clause-to-variable ratios with the different distributions. The bottom curve is for random 2SAT with $m/n = 1$. This is near the hardest ratio for random 2SAT, and since 2SAT is solvable in linear worst-case time, provides a kind of baseline for comparison. Although there are 2SAT instances on which DP will take exponential time, the curve for random 2SAT does appear to be linear. The correspondence of other curves to the random 2SAT curve will be used later in this chapter to argue that the formulas in question are also easy. This argument holds in essence even if the DP performance on random 2SAT is super-linear. DP performs quite well on random 2SAT, and if DP performs at least as well on the formulas in question they must also be easy (but not necessarily solvable in linear time).

The curve for random 3SAT with $m/n = 4$ is also plotted in Figure 5.7 as a second line for comparison, although only a small part of it appears at the upper left of the graph. The second curve from the bottom is for random $\mathcal{E}$3SAT with $m/n = 2$. These formulas are only very slightly harder to test than the 2SAT formulas. The curve for random $\mathcal{E}$3SAT with $m/n = 1$ is not shown, but is nearly identical to the random 2SAT curve. The two remaining curves are for 3SAT and $\mathcal{E}$3SAT with $m/n = 3$,

very near the peak ratio for $\mathcal{E}$3SAT. The $\mathcal{E}$3SAT formulas are easier than the 3SAT formulas, but only a little. These two curves are clearly not linear, but at the same they provide no evidence of exponential growth. Of course, this doesn't show that exponential growth does not occur, since this could be the tail of an exponential curve with a very small constant in the exponent. To make this clearer, Table 5.2 gives the actual values plotted in Figure 5.7, with additional figures for $n = 500$, based on smaller sample sizes, of about 200 except for 3SAT at $m/n = 3$, which is based on 30 samples, and 3SAT with $m/n = 4$, which is an estimate based on extrapolating from data points with smaller n. The growth rate for random $\mathcal{E}$3SAT with $m/n = 3$ is

Distribution	25 vars	50 vars	75 vars	100 vars	500 vars
2SAT, $m/n = 1$	19	37	55	74	363
$\mathcal{E}$3SAT, $m/n = 2$	23	45	68	98	441
$\mathcal{E}$3SAT, $m/n = 3$	31	68	112	164	824
3SAT, $m/n = 3$	33	76	147	250	996,111
3SAT, $m/n = 4$	128	1687	16544	174339	10^{22}(est)

Table 5.2: Effect of n on DP Calls: comparison of distributions

actually perfectly consistent with a growth rate of $\sim n^2$, and not with exponential or linear growth. This rate of growth is dwarfed by that of the numbers for random 3SAT with $m/n = 4$, which clearly increases by an order of magnitude for each increment of n.

Perhaps the most important question about a random SAT distribution is the average-case complexity. As has been remarked, finding this analytically appears to be a formidable task. Although, unfortunately, it is not possible with the kind of empirical data presented here to distinguish clearly between polynomial and exponential behaviour, the data shown in Tables 4.1, 5.1 and 5.2 is surprisingly compelling. The growth for $m/n \leq 2$ is easily seen to be essentially linear; each constant increase in the number of variables results in a constant increase in DP calls, for both families of distributions. The random $\mathcal{E}$3SAT formulas at $m/n = 3$ appear to be of $\sim n^2$ average-case complexity. For random 3SAT, the story is quite different at the peak.

Here, for each constant increase in the number of variables we have a greater than factor of 10 increase in the number of calls. This looks very much like exponential behaviour. For $m/n \geq 6$, things are not so clear. Although the growth is significant, it is much less dramatic than for the peak. Preliminary experiments, and those of others (for example, (Crawford and Auton 1993)) suggest that these trends continue to much higher values of n (the highest that current technology makes it practical to test). Thus, it would be somewhat surprising to discover that the actual average-case behaviour for random 3SAT is anything but linear for very low ratios, or that the actual average-case complexity of DP on random 3SAT was less than exponential for ratios near the peak. Interestingly, the case of $m/n \geq 6$, for which the empirical data are not categorical, is the only case for which we know the average case complexity of DP on random 3SAT is exponential (Chvátal and Szemerédi 1988). Empirically, though, these formulas are do not appear nearly as challenging as those at the peak, so the result of exponential *asymptotic* performance is only moderately informative for moderate-sized problems.

Using a version of the Davis-Putnam Procedure with additional heuristics, and tuned for optimum performance on random 3SAT formulas, (Crawford and Auton 1993) have recently investigated the rate of growth of average testing time for their program "tableau" as n was varied up to 1000 variables at low ratios, up to 260 in the hard region, and to 600 at $m/n = 10$. Their data suggests strongly that for $m/n \leq 3$, their program tests random 3SAT in time linear in n, and that above the transition region it takes time exponential in n. For n much greater than the transition ratio, the exponent appeared to be much smaller. This data is highly suggestive that the actual average-case complexity will turn out to be linear for ratios below 3, and with respect to this question is stronger than the data presented here. The data presented above are stronger in the following sense. The procedure used in (Crawford and Auton 1993) was optimized on random 3SAT, and shows that it is possible to test these formulas, with small ratios, in linear time at least for small n. Although the value of their heuristics is not restricted to random 3SAT, it is possible that there is some tradeoff of performance on some other sets of formulas that would not be desirable in practice. The procedure DP is a very simple procedure, not optimized at all for performance

on these particular formulas. The performance of DP on these formulas, apparently about $\sim n^2$, thus suggests the minimum level of performance that would be accepted in a practical program.

Finally, these data provide empirical evidence that, as Franco's asymptotic results suggested, that the random $\mathcal{E}k$SAT family of formulas is too easy to be very useful for testing SAT procedures. An intuitive explanation for the ease of the random $\mathcal{E}k$SAT formulas is that for small k they contain quite large numbers of clauses of length 2. Further, as n increases, the proportion of clauses of length 2 increases, so that in the limit they will consist mainly of these clauses. A procedure which cannot solve these almost certainly must be regarded as a poor one, since the simplest reasonable procedure for SAT has no trouble with them. However, success in solving these formulas probably cannot be regarded as a demonstration of a good procedure. Solving these formulas quickly would seem to be a necessary criterion for calling a procedure good, but far from a sufficient one.

5.3 Critique of Previous Experiments

In this chapter, previous experimental work with random formulas will be reviewed in light of the results of Chapter 4 and the previous sections of the current chapter. The implications of these results for evaluating SAT testing programs will be demonstrated, and some additional distributions considered, with further data presented as needed. Two additional informal terms will be used in this section, which are described with reference to the top curve in Figure 5.8. The region to the left (below $m/n = 1$ for this curve) where the number of steps drops very quickly as m/n is decreased, is referred to as the "shoulder" of the curve. The nearly flat region between the shoulder and the point near the hard region where difficulty begins to rapidly increase (from about $m/n = 1$ to $m/n = 2.5$ here), is referred to as the "plateau". These regions are convenient for reference because they show up, to one degree or another, in almost all the curves we are concerned with. In the graphs in this chapter, each data point is computed from a sample size of 1000 unless otherwise indicated.

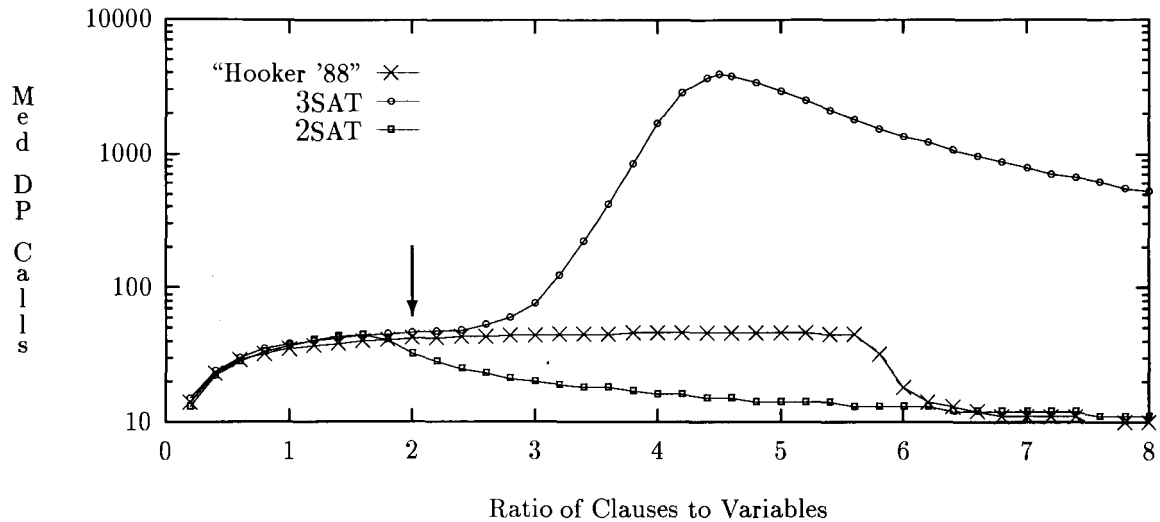

Figure 5.8: Median DP calls for formulas in Hooker (1988) *vs.* random 3SAT

5.3.1 Random $\mathcal{E}k$SAT and Related Distributions

In (Hooker 1988c), a version of the binomial clause length family of formulas was used in which unit clauses, but not empty clauses were allowed. Formulas of this type were used as test instances to compare the performance of two SAT testing procedures, one based on integer programming and cutting planes, the other using the resolution principle. Formulas with expected clause length of 5 were generated from 10, 20, 30 and 50 variables, all with a clause-to-variable ratio of 2, and from 20 variables at 6 ratios in the range of 1 to 32, with 20, 40, 60, 80, 120 and 640 clauses. In Chapter 3 it was remarked that formulas like these were too easy to be useful in evaluating SAT testing programs, and this lead to introducing the restriction of not permitting unit clauses. To demonstrate this claim, 50-variable formulas were generated with an expected clause length of 5, for clause-to-variable ratios varying from 1/5 to 50. The median DP calls to test these formulas is shown in Figure 5.8, along with the corresponding curves for 50-variable random 3SAT and random 2SAT for comparison. An arrow highlights the point at which Hooker's tests were made. The median number of DP calls to test these formulas is less than 50, indicating that the procedure merely had to set the variables and did almost no backtracking.

The graph shows that the "plateau" in the curve for these formulas is at the same level as the easy random 3SAT formulas, and also the same level as the random 2SAT peak. The correspondence to the random 2SAT curve also holds at 100-variables, suggesting that these formulas are no harder to solve than the linear-time 2SAT.

Hooker later dissallowed unit clauses, as did others using related formula distributions. For example, (Hooker and Fedjki 1989) constructed formulas by generating clauses by Goldberg's method, setting $p = 0.5/n$, but rejecting all clauses of length less than 2. This distribution is very similar to random $\mathcal{E}$5SAT, but with average clause lengths slightly greater than 5. They used formulas with 50 and 100 variables, which have average clause lengths of 5.15 and 5.16 respectively. Being aware of the existence of a transition region, they tested at several ratios in and around that region. Figure 5.9 shows curves of the median number of DP calls for these formulas, for m/n in the range 0 to 16. For comparison it also shows the curves for 50-variable random 3SAT and for the formulas used by (Hooker 1988c), shown above in Figure 5.8. (Kamath, Karmarkar, Ramakrishnan, and Resende 1990) used random $\mathcal{E}$5SAT with 50 and 100 variables, at a clause-to-variable ratio of 2. The actual curves for random $\mathcal{E}$5SAT are not shown, because they are almost the same as those for the formulas used by Hooker and Fedkji, the only difference being a slight shift to the left of the hard region, which does not affect the arguments made here. Interestingly, as far as can be discerned from the small sample sizes (10 or 20 formulas for each point), and small number of data points, the hard point for their program seems to be very nearly at the same ratio as is the hard area for DP .

Each point tested in (Hooker and Fedjki 1989) and (Kamath, Karmarkar, Ramakrishnan, and Resende 1990) is indicated in Figure 5.9 by an arrow. The two arrows at ratio 2 indicate the points used by (Kamath, Karmarkar, Ramakrishnan, and Resende 1990). At the ratio of $m/n = 2$, which (Kamath, Karmarkar, Ramakrishnan, and Resende 1990) used, the curves correspond in difficulty with those of the formulas used by (Hooker 1988c) and with random 2SAT. This correspondence also holds for 100-variable formulas in this region, leading to the conclusion that these formulas are probably solvable in linear time. The formulas in the hard region are easier, for the same number of variables, than random 3SAT, but only a little, and the rate of growth

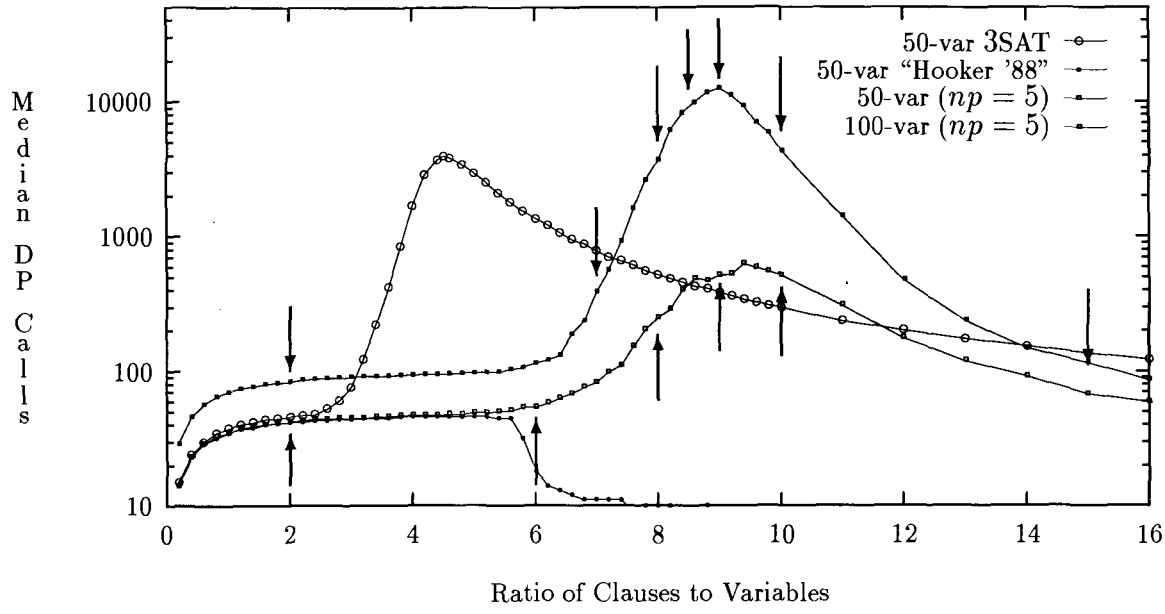

Figure 5.9: Median DP calls for random $\mathcal{E}5$SAT and formulas from Hooker & Fedjki (1989)

of the peak for these curves is significant, with the increment from 50 to 75 variables producing an order of magnitude increase in the peak. These formulas, then, do give some appearance of being reasonable test material, provided they are tested near the hard region. However, just as with the random $\mathcal{E}3$SAT formulas, which the previous sections showed were quite easy, in these formulas shorter and shorter clauses will have to dominate as n is increased, to keep the average clause length fixed. Thus, as n is increased they may become progressively poorer test material.

Chapter 4 looked in depth at the case of $k = 3$. Use of these formulas to evaluate SAT testing programs has been reported by (Gu 1992) and (Kamath, Karmarkar, Ramakrishnan, and Resende 1990). Gu used random $\mathcal{E}3$SAT with 50, 500 and 1000 variables, all at $m/n = 2$, and with sample size of 30 formulas for each of the parameter settings. Kamath et al. (1990), also used random $\mathcal{E}3$SAT with 1000 variables at $m/n = 2$. We have seen already that the random $\mathcal{E}3$SAT formulas are extremely easy for small n, but it remained to be determined if n as large as 500 or 1000 produces challenging instances. The upper graph in Figure 5.10 shows curves of the median

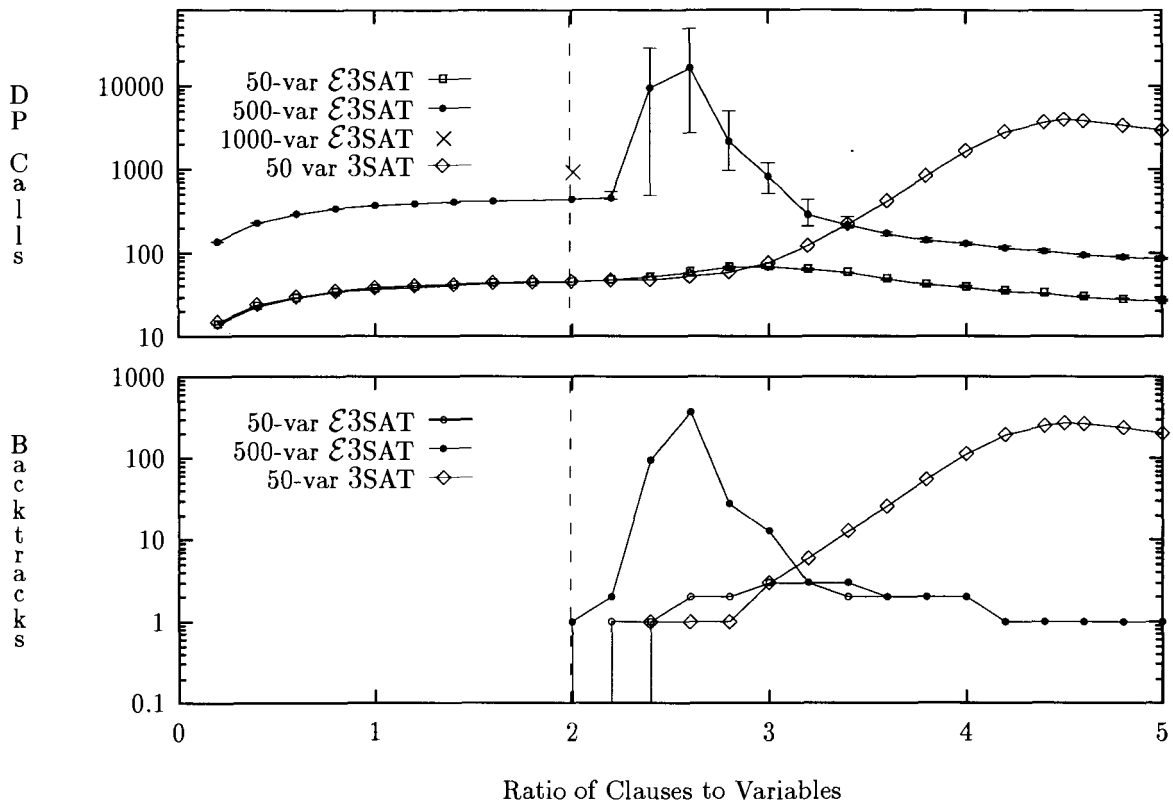

Figure 5.10: Median DP Calls for random $\mathcal{E}$3SAT

number of DP calls for random $\mathcal{E}$3SAT with 50 and 500 variables. A single data point at that ratio is shown for $n = 1000$, based on a small test sample of 37 formulas. Also shown, for comparison, is the curve for 50-variable random 3SAT. The location of the tests by Gu and Kamath et al. ($m/n = 2$) is indicated by a vertical dashed line.

Earlier in this chapter it was argued that random $\mathcal{E}$3SAT formulas are too easy to provide useful test instances, yet the 500-variable formulas do not appear to be trivial. The formulas at the peak of the 500-variable curve are a little harder than those at the peak of the 50-variable random 3SAT curve, typically taking more than 10,000 DP calls to test. Thus, it could be argued that these formulas should be considered reasonable test instances. However, the fact that much larger numbers of variable are required to reach this level of difficulty is telling, giving that formulas from the hard

region of random 3SAT with 500 variables apparently cannot be tested with success by any known sound and complete procedure. It would seem likely that formulas that can be solved in $\sim n^2$ on average by a simple procedure like DP are structurally different from the kinds of formulas we should use to show effective performance of a procedure on very hard problems.

This said, the hardest 500- and 1000-variable random $\mathcal{E}$3SAT formulas still require non-negligible effort to test, and may be regarded as at least a moderately good testing tool. However, the upper graph of Figure 5.10 shows clearly that the experiments using random $\mathcal{E}$3SAT are to the left of the hard region. The number of steps required to test these is much less than at the peak, but it is still not near zero, and one might still suspect some value in their use. As a further challenge to their value, the lower graph of Figure 5.10 shows curves of the median number of backtracks by DP in testing the same sets of formulas. The median number of backtracks is 1 at $m/n = 2$ for both the 500 and 1000-variable formulas. That is, for most of these formulas the number of DP calls is just that number required to set enough variables to satisfy the formula, and almost no backtracking is required. Given that the DP relies only on luck and unit propagation to find correct variable assignments, this near-absence of backtracking is difficult to interpret in any way other than demonstrating that the formulas are trivial.

Formulas from random $\mathcal{E}$10SAT were used in (Gu 1992; Kamath, Karmarkar, Ramakrishnan, and Resende 1990). Gu tested formulas with 1000 variables at ratios of 1, 2, 4, 8, 10 (i,e., 1000,...,10000 clauses), and with 2000, 3000, 4000 and 5000 variables all at ratio of 10. He took 30 samples at each point. Kamath et al. tested formulas with 50, 400, and 500 variables at $m/n = 2$, and with 1000 variables at $m/n \in \{2, 4, 8, 16, 32\}$. Based on the experimental results from Section 5.1, we would expect the formulas in the hard area to be quite challenging (although no claim is made about the rate of growth of difficulty for these), but also that the hard region should occur at quite a high ratio of clauses-to-variables. Two tests were carried out with these formulas. First, to estimate the transition region, 50-variable formulas were generated and tested with m/n up to 50. The same "shoulder" region occurs at the lower ratios as for all the other distributions looked at, but extending up to

about $m/n = 8$. From $m/n = 8$ to $m/n = 50$ the curve is nearly flat, increasing very slowly with the increasing ratio, but showing no evidence whatsoever of approaching the hard region. At $m/n = 50$ not a single formula out of 1000 was unsatisfiable. Additionally, essentially no backtracking is required to test these formulas at any ratio below 30, where the mean number of backtracks is about 1. To confirm that increasing the number of variables does not produce hard formulas, tests were run on 1000 variable formulas. These were tested with m/n ranging only up to 10. At 10, the curve produced by the median DP steps for these formulas has only just passed the usual "shoulder" found at very low ratios. About 90% of the formulas as $m/n = 10$ required zero backtracks to test, and out of 1,000 formulas only one required as many as three. The formulas at lower ratios required fewer backtracks.

It is only possible to conclude that these formulas also are trivial to test and of very little interest. This said, it is worth remarking that tests for 1000-variable random $\mathcal{E}$3SAT formulas at above $m/n = 10$ were prohibitively expensive to carry out, because of the time spend in building up the data structures which in smaller, but more challenging by usual measures, formulas are extremely helpful in speeding up execution time. This suggests that, while formulas such as random 3SAT might be most useful for finding good procedures, a wide variety of formulas should be used for testing an actual implementation. It also suggests that some experimenters may have believed they were testing logically, or structurally, hard formulas when they were not because of implementation problems like this, or like that illustrated by Hooker's resolution program that could not solve very easy problems of even modest size.

5.3.2 Random kSAT Formulas

Until recently, few researchers have used random kSAT formulas for empirical tests. The earliest example appears to be (Blair, Jeroslow, and Lowe 1986), which will not be discussed here. Somewhat more recently is (Gallo and Urbani 1989), discussed below.

d'Anjou and his colleagues (d'Anjou, Granã, Torrealdea, and Hernandez 1993) report a method for solving SAT instances with Boltzmann Machines. Like the local

search methods discussed in Section 2.2.3, the Boltzmann Machine method is a sound but incomplete SAT testing procedure. They report experiments to demonstrate its performance both in terms of speed and correctness, using formulas from random kSAT with $k \in \{2, 5, 6, 7\}$ and from UD[2,7] (see Section 5.3.3). They suggest that their experiments show their procedure solves SAT in time linear in the number of variables, and that it is insensitive to the number of clauses or the distribution of clause sizes. Here I consider the clause size issue; the others are addressed in Section 5.3.3.

To show insensitivity of their procedure to clause size distribution, they generate and test formulas from random kSAT with $k \in \{2, 5, 6, 7\}$, and for $n \in \{10, 15, 20, 25\}$, in every case with 25 clauses. For each value of n, their procedure executes in essentially the same time for all values of k tested. This could seem to be an impressive result, given our knowledge that difficulty for the Davis-Putnam Procedure increases dramatically with k. However, note that the largest clause-to-variable ratio in this set of parameter values is 2.5. Thus, in every case except 2SAT (which is easy anyway), they are well below the hard region. Moreover, as k increases, the hard region shifts rapidly to higher ratios. In these experiments, they increase k while holding the number of clauses constant. As can be seen in Figure 4.8, the curves to the left of $m/n = 2.5$ are essentially the same for DP also, so that their data appears to be an artifact of the formulas used, and does not inform us about the effect of clause length on the performance of their procedure in the more general case.

In (Gallo and Urbani 1989), random 3SAT formulas with 10, 20, 30, 40 and 50 variables and 50, 100, 140, 170 and 200 clauses respectively, were used. A sample of 10 formulas was tested at each of the 5 points in the parameter space. Table 5.3 gives the points at which they generated random 3SAT formulas, along with the clause-to-variable ratio, the transition ratio (i.e., the ratio at which half of the formulas are satisfiable), the number of formulas they found satisfiable, and the average time for their implementation of the Davis-Putnam Procedure to test the formulas. Since the exact position of the peak may vary with the procedure used it is not possible to determine exactly where each of these tests lies on the hardness curve for Gallo and Urbani's procedures. However, the m/n ratio for these tests range from 5 to 4. Since all these tests are fairly near the hard area, and since the same formulas

variables	10	20	30	40	50
clauses	50	100	140	170	200
ratio	5.0	5.0	4.67	4.25	4.0
transition	4.86	4.55	4.45	4.40	4.36
satisfiable	7	5	2	6	8
time (s)	0.3	1.2	5.1	13.5	32.7

Table 5.3: Locations of Tests in Gallo & Urbani (1989)

were used to compare the different procedures they look at, their data are useful for that comparison. However, as they increase n in this experiment, they shift their choice of m/n from somewhat above the transition point (5 as compared with 4.86) to somewhat below the transition point (4 as compared with 4.36). It would be surprising if this shift maintained the points at the same position on the hardness curve, given that the data presented in Chapter 4 suggest the peak shifts with the transition point, and that they make no mention of having taken the hardness pattern into account. Thus, not even a crude estimate of the rate of growth of difficulty for their procedures can be made from this data. Further, it is not possible to compare their results with any new data, because there is no way to match up the ratios relative to the hard points. An additional confound in interpreting their data is that, with such a small sample size, the proportion of satisfiable formulas they have varies considerably from the proportion of satisfiable formulas in the population. Since the satisfiable and unsatisfiable instances can have very different expected running times at or just below the transition point, the average times from this experiment may not be very good indicators of the actual average.

The fact that they not only maintain a nearly linear ratio, but shift it in the correct direction to account for the shift with transition point with n, shows that Gallo and Urbani did have some awareness that the number of clauses would have to grow approximately linearly with the number of variables (although this appears to have been motivated by their desire to have even proportions of satisfiable and unsatisfiable instances, rather than by awareness of the hard region). The fact that

they failed to shift their test point in accordance with the actual shift of the 50%-satisfiable point can be attributed to the small sample sizes used, from which they could not possibly have made a precise estimate of the transition point.

Gallo and Urbani also tested 3SAT for 30, 40 and 50 variables with $m/n = 5.$ with manipulation of the proportion of Horn clauses (50, 70, 80 and 90 per cent). Since the manner of controlling percent Horn is not specified, no comparison can be made. (We might assume that at 50 per cent Horn, we have random 3SAT).

In (Selman, Levesque, and Mitchell 1992) a new local search procedure for solving SAT problems was introduced. In this paper, the procedure is tested on instances of N-queens, graph colouring, boolean induction, and random 3SAT. The random 3SAT problems used were at $m/n = 4.3$, and had from 50 to 500 variables. Performance was compared with the version of the Davis-Putnam Procedure used in (Mitchell, Selman, and Levesque 1992). At the time, $m/n = 4.3$ was the best estimate of the transition point and of the hardest region, but since in fact these points shift below 4.3 with increasing n, this data also probably involves a confounding shift with respect to the hardest point. However, the largest difficulty with evaluating this and other similar procedures is that there was (and still is) no sound and complete SAT testing program available capable of testing the larger formulas. Since the procedure presented there is sound but incomplete, there is no way of knowing if it solved all the satisfiable formulas given, and thus no way of knowning if the reported times are correct.

Hooker (1988) made a single experiment with 20 variables and 640 clauses of length 5 (to make a quick comparison with the Chvatal & Szemeredi results). Since (Hooker 1988c) provided no other condition to compare the data with, this seems rather uninformative.

5.3.3 Uniformly Distributed Clause Lengths

As well as the random 3SAT formulas discussed above, (Gallo and Urbani 1989) tested formulas with clause lengths uniformly distributed over the range [1,7] (ie., on average, these formulas have equal numbers of clauses of each length 1–7). For convenience, I call these formulas UD[1,7] formulas. Figure 5.11 shows the median DP calls and

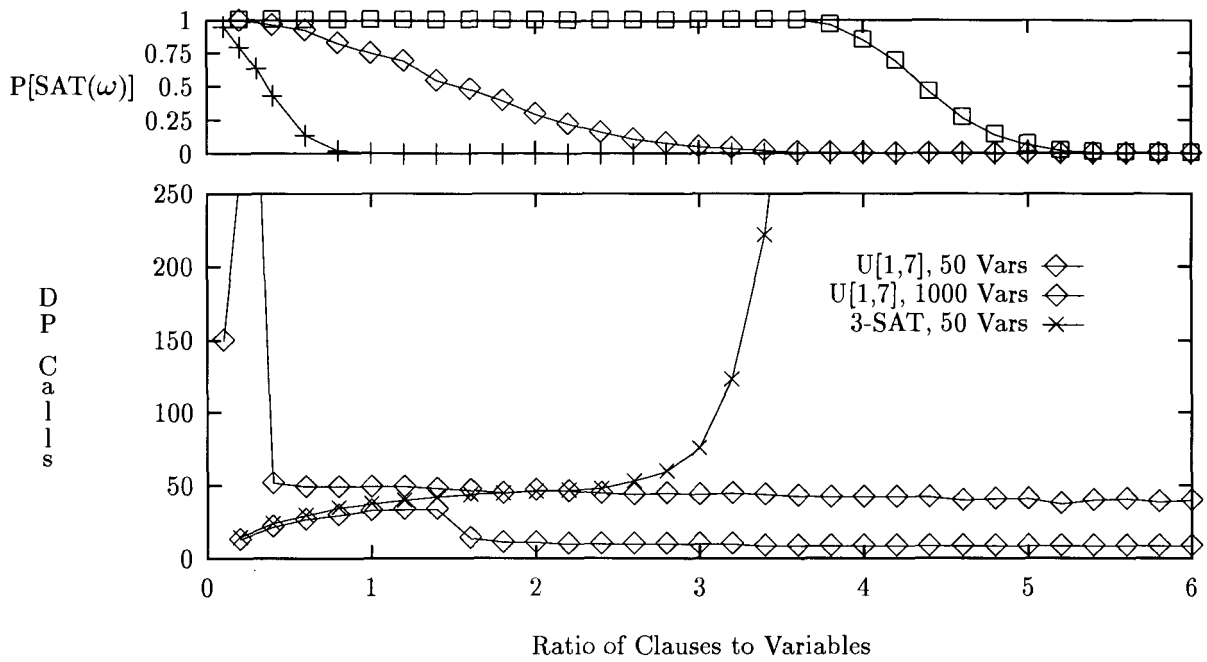

Figure 5.11: Median DP calls for UD[1,7]

proportion satisfiable curves for UD[1,7] formulas with 50 and 1000 variables, and for comparison the corresponding curves for random 3SAT. The UD[1,7] formulas exhibit essentially the same behaviour as random 2SAT, with the usual "shoulder" pattern at the left, but no real hard region. The large peak at the left of the 1000-variable curve corresponds to the shoulder, and only looks substantial because some few hundred variables need to be assigned values. Like the other trivial formula sets, virtually no backtracking is required to test these formulas.

With formulas based on Goldberg's generation method, the exclusion of unit clauses was required to obtain non-trivial formulas. It was hypothesized that unit clauses may also be the reason the UD[1,7] formulas are trivial, and to test this UD[2,7] formulas, in which unit clauses were not permitted, were generated and tested. Figure 5.12 shows the resulting curves. The UD[2,7] formulas exhibit the now-familiar easy-hard-easy pattern, although they are not as hard as the random 3SAT formulas. Presumably they are easier than 3SAT because of the large number of clauses of length 2, since excluding these will almost certainly produce formulas at least as hard

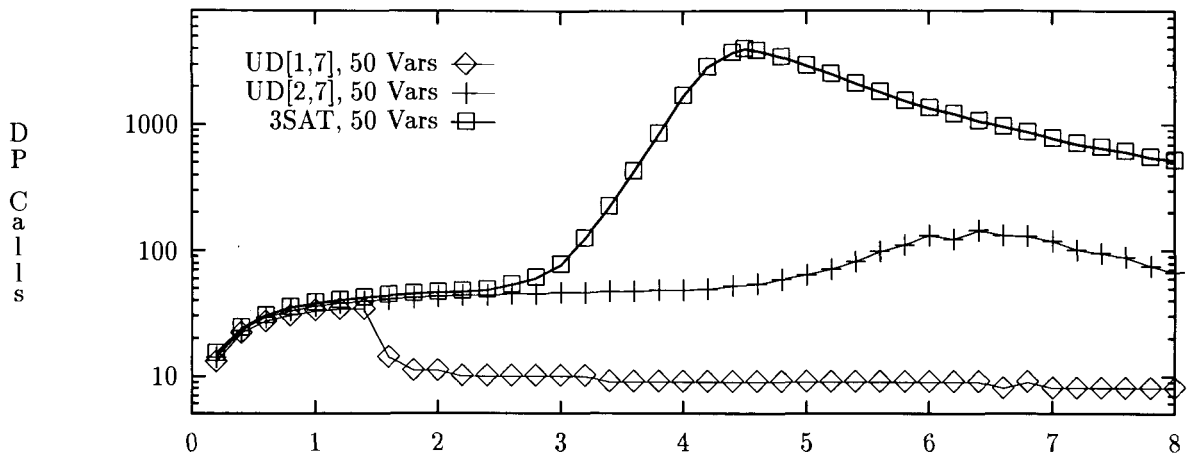

Figure 5.12: Median DP calls for UD[2,7]

as 3SAT.

The distribution UD[2,7] was used in the experiments reported in (d'Anjou, Granã, Torrealdea, and Hernandez 1993) (see Section 5.3.2 for a brief introduction). They based their main result of linear time performance on testing formulas from UD[2,7] with $n \in \{10, 15, 20, 25, 30, 35, 40, 100\}$, in every case with 25 clauses. The point based on 100 variables is suspect a priori, since with 25 clauses the expected number of literals in a formula is 112, so that with 100 variables (and thus 200 literals to choose from) we do not even expect all the variables to be mentioned in a formula. That aside, note that the clause-to-variable ratio here shifts from 2.5 down to 0.25 as the number of variables is increased. These formulas are quite easy at $m/n = 2.5$, and as they increase the number of variables, they move their formulas to even easier ratios, as can be seen in Figure 5.12. Moreover, their procedure took about twice as long to test 50-variable formulas as it did the 25-variable formulas, whereas DP took only marginally longer (26 *vs.* 18 calls) for the 50 variable formulas. At the same time, the curve they show of performance on these formulas is remarkably near to a perfect line. It would seem that their linear performance is worse that the difficulty of the formulas requires, and is probably the result of the rate of growth in the size

of their data structures.

An additional secondary claim made in (d'Anjou, Granã, Torrealdea, and Hernandez 1993) is that the performance of their procedure is insensitive to the number of clauses in a formula. To demonstrate this, they test formulas from UD[2,7] with 20 variables for m varying from 25 to 55. The average time of their procedure is almost independent of the number of clauses, with only a slight increase in time with m (although the variance in times increases with m). However, the range of clause-to-variable ratio here is from 1.25 to 2.75, and as Figure 5.12 shows, the time for DP to test these formulas also does not increase as the ratio varies across this range. Again, the data seems to be an artifact of the formulas chosen, and not indicative of positive performance results for the procedure.

Chapter 6

Discussion

Random formulas have been used by many researchers for evaluating the performance of SAT testing programs. Similar distributions of formulas have been used both for empirically evaluating SAT testing programs and in performing average-case analyses of SAT testing procedures. How informative results based on these random formulas are depends crucially on the "reasonableness" of the formula distributions chosen. Somewhat surprisingly, since this point is presumably understood by all workers in the field, little effort has been made in the past to understand the properties of the random formulas being used in empirical testing. The primary objective of this thesis was to investigate this apparent shortcoming.

Typically, a family of random formula distributions is defined by giving a method for generating instances from a set of parameters. Typical parameters for a distribution family are the number of variables, the number of clauses per formula, and number of literals per clause. Workers normally test formulas from several points in the parameter space, but generally this has been done with little knowledge of the general character of the formulas produced by the generation method, or of the the effects of the parameters on the formulas under consideration. In this thesis, formulas from the most commonly used distribution families were generated methodically using a wide range of parameter values, and tested for satisfiability with a simple version of the Davis-Putnam Procedure, here called DP. It was shown that for these distributions certain predictable patterns emerge as a result of varying the parameters.

The patterns were demonstrated in terms of the number of steps required for a simple version of the Davis-Putnam Procedure to test the formulas and the proportion of formulas which were found to be satisfiable. Most significantly, each distribution exhibited a characteristic "easy-hard-easy" pattern at a particular ratio of clauses to variables, independent of the number of variables. At the same ratio where the hard region occurs, there is a transition from almost all formulas being satisfiable to almost all formulas being unsatisfiable.

It was also shown that some formula distributions are of very questionable value for providing test instances for evaluating SAT testing program performance, because they are simply too easy to test. A set of test problems which any reasonable procedure can solve very easily cannot be used to distinguish programs which perform well on both easy and hard problems from those which perform well on easy problems but not on hard ones.

Previous reports of experimental work with random SAT were reviewed in light of the data presented here, to establish to what degree these results may affect the practice of experimental evaluation of SAT programs. This review showed that many studies examining performance of SAT programs used formulas of dubious value for the purpose. It was also pointed out that failure to understand the patterns of effects of the parameters can lead to misinterpretation of data, even when challenging formula sets are selected. As a consequence it is claimed that understanding of these patterns is in fact elementary and critical to this work.

The primary results presented here are based on testing randomly generated formulas from the two most used families of distributions. In the first of these, formulas are generated by mentioning, or not mentioning, each variable in a clause with some given probability. In experimental work, the most commonly used variants of this method involve making this probability a function of the number of variables such that the average clause length remains constant regardless of the number of variables. The second distribution family is based on random clauses of fixed length. These two families were called random $\mathcal{E}k$SAT and random kSAT, respectively. For both of these distribution families, as well as the others that were looked at, it was found that the distributions can usefully be viewed in terms of three parameters: the number of

variables (n), the ratio of clauses to variables (m/n), and average clause length (k).

Formulas with very low clause-to-variable ratios are almost always satisfiable, and formulas with very large clause-to-variable ratio are almost always unsatisfiable. Simple counting arguments can be used to show that this must be so, and thus that some transition must happen between these states as the clause-to-variable ratio is increased. Such a transition presumably holds for all non-trivial families of random formulas in which the number of clauses can be varied independently. However, the exact nature of this transition has only recently become a subject of active study. The data presented here show that, at least for the distributions looked at, the shift from almost always satisfiable to almost never satisfiable as the clause-to-variable ratio is increased does not occur slowly over a very wide range. Rather, the transition happens smoothly but quickly over a narrow range of ratios, and for all clause-to-variable ratios below this range almost all formulas are satisfiable, and for all values above it almost all formulas are unsatisfiable. This narrow region, over which the proportion of formulas which are satisfiable shifts smoothly from 0 to 1 as the ratio is increased, was called the *transition region.* For a given distribution family, and fixed value of the clause-length parameter, this transition region occurs at approximately the same ratio of clauses to variables for all values of n. An interesting open question about the transition region is whether, in the limit as n goes to infinity, it remains a region or becomes a step transition. The ratio where exactly half the clauses are satisfiable has sometimes been called the threshold or flip point. For moderate numbers of variables, this point shifts to slightly lower ratios as n increases. Another open question is the asymptotic ratio for this point. The data here, although based on values of n too small to be very convincing, suggest that the threshold point for random 3SAT is near 4.1, which is a smaller value than previous predictions.

The expected number of steps for DP to test a random formula also varies with the clause-to-variable ratio. For the more interesting distributions this measure goes from very low values at low ratios, to much higher values, and then becomes low again. This "easy-hard-easy" pattern appears to be inherently related to the transition region. The peak values of number of DP steps occur at ratios where 1/3 to 1/2 of the formulas are satisfiable, and the number of steps drops off rapidly to lower levels on

either side of the peak. The area near the peak was called the *hard region.* Although similar patterns occurred for all the families considered, the individual distributions vary greatly both in difficulty at particular values of n, and in the increase of difficulty as n is increased. This is especially so near the hard region.

Formulas were generated from each of the two main distribution families, for a wide range of parameter values. The effect of each parameter was considered independently, and illustrated by graphs of curves showing the median number of DP steps required to test the formulas, and the proportion of the formulas which are satisfiable. Not surprisingly, the effect of increasing n, in every case, is to increase the number of DP steps required for testing. However, the rate of increase in difficulty is very dependent upon both the kind of distribution of formulas used and the particular values of the other distribution parameters. The effect of m/n has already been described, in terms of the hard-easy-hard pattern and the transition region. The effect of increasing k is to shift the transition region and the hard region together to higher clause-to-variable ratios, and to make testing harder for DP , especially increasing the height of the peak of the DP calls curve. This shift in ratio is apparently exponential in k, and the increase in difficulty may also be exponential in k for some distributions.

The consequences of these patterns for experimental (and, to a lesser degree, analytical) studies of random formulas are significant, the main consequence being that it is very easy to end up testing one's procedure only on quite easy formulas. A review of many of the reported empirical investigations using random SAT was carried out in light of the patterns shown by the initial study of random kSAT and random $\mathcal{E}k$SAT. This review demonstrates that understanding of the basic properties of random formulas illustrated here is essential to effective empirical study. There are three basic ways in which failure to understand these properties can lead to flawed experiments. The first is to use a method of formula generation which produces an overwhelming preponderance of formulas which are easy to test. The others can occur even with a generation method which is capable of producing challenging test sets. The second is to fail to recognize the existence of easy-hard-easy patterns, and thus generate formulas only in the easy region. The third possibility arises from considering viewing the parameter space in terms of numbers of variables and clauses, as if these

were orthogonal parameters.

In three experiments reviewed, formulas were used which were shown here to be trivial to find solutions for. In two of these, one using a variant of random $\mathcal{E}$5SAT and one using uniformly distributed clause lengths, unit clauses were permitted in the distributions. These formulas, for the parameter values at which they were used, could almost always be shown satisfiable with no backtracking (and thus in well under n steps). These distributions were also shown to be easy to test at other clause-to-variable ratios, and it is conjectured that they can be tested in linear time on average. Similar distributions with unit clauses dissallowed were shown to be somewhat less easy. In the third case, random $\mathcal{E}$10SAT was used. It is possible that at the hard region these formulas are not easy, but this occurs at a very high clause-to-variable ratio. The ratios used to perform the reported test were so far below the transition ratio that the formulas were trivially found satisfiable, again with no backtracking.

Early analytical results suggested that formulas generated by mentioning variables in clauses with some given probability are very easy, at least asymptotically. The formulas like this used most often in empirical studies, which are called here random $\mathcal{E}k$SAT would appear not to be as easy as those used in the analytical work, since empty and unit clauses are eliminated. The experiments here, though, show that these formulas also are generally quite easy for DP . Even at the hard region, they appear to be solved in $\sim n^2$ time on average by DP . The random kSAT formulas, on the other hand, appear very much to take exponential time on average to test at ratios in the hard region. So, for some parameter settings the random $\mathcal{E}k$SAT formulas are not trivial, but they are still quite easy compared to random kSAT formulas. If our interest is the test formulas on the most challenging problems that may arise in practice, then these formulas would seem to be a poor choice. Of course, failure of a procedure to test these problems as efficiently as DP would bring into the question the usefulness of the procedure.

With random kSAT formulas (for $k > 2$), very challenging problems can be generated with modest numbers of variables, if formulas are generated at clause-to-variable ratios near the hard region. Failure to recognize the existence of the hard region could lead to testing formulas at easy ratios, again bringing into question the value of the

resulting data. An additional pitfall is that manipulating the numbers of variables and clauses independently leads to confounding of the effects of the two parameters on difficulty. For example, formulas at ratios less than 2 are easy to test for small values of n, and appear to be solvable in polynomial time on average. Tests made in this regions may not be very valuable, just as those made with random $\mathcal{E}k$SAT are not. Further, if the number of variables is then increased while keeping the number of clauses constant, then as n is increased the formulas are shifting toward easier parts of the curve, thus making the growth rate look less than it really is. The opposite error could also occur, if ratios above the hard region were used, so that the growth of difficulty with n would be overstated.

In conclusion, the formula distributions similar to random $\mathcal{E}k$SAT are shown to be of little use, and random kSAT more useful, in the following sense. The procedure DP is arguably the simplest "reasonable" SAT testing procedure. Yet this procedure performs very well on the hardest problems in the random kSAT; much better than on even the easy formulas from random kSAT except at very low ratios. Thus random $\mathcal{E}k$SAT would seem to present a minimum requirement – any procedure that does not solve these effectively cannot readily be considered useful. A distribution of problems which are so hard that no known procedure can solve them, except possible for extremely small numbers of variables, also wouldn't be very useful. If all procedures find them unmanageable, then they cannot distinguish good from bad procedures. The random kSAT formulas would appear to be a reasonable compromise. The simple procedure used here, when carefully implemented, can solve them for modest numbers of variables, and so can provide a "base case" data for comparison with better methods. Yet, they are hard enough that moderate-sized instances are challenging for the best known procedures. Selecting this, or another distribution which can be shown to produce challenging problem sets, is not enough though. The worker still needs to be aware of the patterns of effects of all of the parameters to a distribution family. Failure to understand the fundamental patters can lead to experiments in which effects of different parameters are confounded, and to misinterpretation of the resulting data.

Bibliography

Aspvall, B., M. F. Plass, and R. E. Tarjan (1979, March). A linear-time algorithm for testing the trurh of certain quantified boolean formulas. *Information Processing Letters 8*(3), 121–123.

Beckman, R. and R. Cook (1983). Outlier.....s. *Technometrics 25*(2), 119–149. See also the discussion articles in the same volume.

Blair, C., R. Jeroslow, and J. Lowe (1986). Some results and experiments in programming techniques for propositional logic. *Comput. & Ops. Res. 13*(5), 633–645.

Brown, C. A. and P. W. J. Purdom (1981). An average time analysis of backtracking. *SIAM Journal on Computing 10*(3), 583–593.

Bugrara, K. M., Y. Pan, and P. W. Purdom, Jr. (1989, April). Exponential average time for the pure literal rule. *SIAM Journal on Computing 18*(2), 409–418.

Buro, M. and H. Büning (1992). Report on a sat competition. Technical Report 110, Universität Paderborn.

Chandru, V. and J. Hooker (1988). Extended horn sets in propositional logic. Working Paper 88-89-39, Graduate School of Industrial Adminitration, Carnegie Mellon University, Pittsburgh, PA 15213.

Chandru, V. and J. Hooker (1991, January). Extended horn sets in propositional logic. *Journal of the ACM 38*(1), 205–221.

Chao, M. and J. Franco (1986). Probabilistic analysis of two heuristics for the 3 Satisfiability problem. *SIAM Journal on Computing 15*, 1106–1118.

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

Cheeseman, P., B. Kanefsky, and W. M. Taylor (1991). Where the really hard problems are. In *Proc. IJCAI-91.*

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

Chvátal, V. and E. Szemerédi (1988). Many hard examples for resolution. *Journal of the ACM 35*(4), 759–768.

Cook, S. (1971). The complexity of theorem proving procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, New York, pp. 151–158. Association for Computing Machinery.

Crawford, J. and L. Auton (1993). Experimental results on the cross-over point in satisfiability problems. In *Proc. AAAI-93.*

d'Anjou, A., M. Granã, F. Torrealdea, and M. Hernandez (1993, May). Solving satisfiability via boltzmann machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*(5).

Davis, M. (1963). Eliminating the irrelevant from mathematical proofs. In *Proc. Symposium in Applied Math,*, Volume 15, pp. 15–20.

Davis, M., G. Logemann, and D. Loveland (1962). A machine program for theorem-proving. *Journal of the ACM 5*, 394–397.

Davis, M. and H. Putnam (1960). A computing procedure for quantification theory. *Journal of the ACM 7*, 201–215.

Dowling, W. F. and J. H. Gallier (1984). Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming 1*(3), 267–284.

Doyle, J. and R. Patil (1991). Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence 50*, 261–297.

DuBois, O. (1991). Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science 81*, 49–64.

Even, S., A. Itai, and A. Shamir (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing 5*(4).

Ferguson, F. and T. Larrabee (1991). Test pattern generation for realistic bridge faults. In *Proc. IEEE 1991 International Test Conference*, pp. 492–499.

Franco, J. (1984). Probabilistic analysis of the pure literal heuristic for the satisfiability problem. *Annals of Operations Research 1*, 273–289.

Franco, J. (1986a, August). On the probabilistic performance of algorithms for the satifiability problem. *Information Processing Letters 23*, 103–106.

Franco, J. (1986b). Probabilistic analysis of algorithms for np-complete problems. Annual Scientific Report, United States Air Force, Air Force Offic of Scientific Research, Grant No. AFOSR 84-0372.

Franco, J. and Y. C. Ho (1988). Probabilistic performance of a heuristic for the satifiability problem. *Discrete Applied Math 22*, 35–51.

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

Frieze, A. (1993). Personal Communication.

Frieze, A. and S. Suen (1992). Analysis of trow simple heuristics on a random instance of k-SAT. In preparation.

Gallo, G. and M. G. Scutella (1988). Polynomially solvable satisfiability problems. *Information Processing Letters 29*, 211–227.

Gallo, G. and G. Urbani (1989). Algorithms for testing the satisfiability of propositional formulae. *Journal of Logic Programming 7*, 45–61.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman.

Genesereth, M. and N. Nilsson (1987). *Logical Foundations of Artificial Intelligence.* Los Altos: Morgan Kaufmann Pub., Inc.

Gent, I. and T. Walsh (1993). Towards an understanding of hill-climbing procedures for SAT. Submitted.

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

Goldberg, A. (1979). On the complexity of the satisfiability problem.

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

Haralick, R. M. and S.-H. Wu (1987). An approximate linear time propogate and divide theorem prover for propositional logic. *Internaltional J. of Pattern Rec. and AI 1*(1), 141–155.

Hooker, J. (1988a). Generalized resolution and cutting planes. *Annals of Operations Research 12*, 217–239.

Hooker, J. (1988b). A quantitative approach to logical inference. *Decision Support Systems 4*, 45–69.

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

Hooker, J. and C. Fedjki (1989, December). Branch-and-cut solution of inference problems in propositional logic. Working Paper 77-88-89, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213.

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

Johnson, D., C. Aragon, L. McGeoch, and C. Schevon (1991). Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and

number partitioning. *Operations Research 39*(3), 378–406.

Kamath, A., N. Karmarkar, K. Ramakrishnan, and M. Resende (1990, May). Computational experience with an interior point algorithm on the satisfiability problem. In *Integer Programming and Combinatorial Optimization*, pp. 333–349. Mathematical Programming Society.

Kamath, A., N. Karmarkar, K. Ramakrishnan, and M. Resende (1991). A continuous approach to inductive inference. Submitted for Publication.

Karmarkar, N., M. Resende, and K. Ramakrishnan (1988). An interior-point algorithm for zero-one integer programming. In *Proc., 13th International Sympo. on Mathematical Programming, Tokyo*. Mathematical Programming Society.

Kautz, H. and B. Selman (1992). Planning as satisfiability. In *Proceedings, ECAI-92*, Vienna.

Kirousis, L. and C. Papadimitriou (1985). The complexity of recognizing polyhedral scenes. In *Proceedings, The 26th Annual Symposium on the Foundations of Computer Science*, Portland, Oregon.

Larrabee, T. (1992, January). Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 6–22.

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

Lee, S.-J. and D. A. Plaisted (1990). New applications of a fast propositional calculus decision procedure. In *Proc. CSCSI-90*, pp. 204–211. CSCSI.

Loveland, D. (1978). *Automated Theorem Proving: A Logical Basis*. North Holland.

Mackworth, A. (1991a). Constraint satisfaction. In *The Encyclopedia of Artificial Intelligence*. New York: John Wiley.

Mackworth, A. (1991b). The logic of constraint satisfaction. Technical Report 91-26, Department of Computer Science, University of British Columbia.

Mitchell, D., B. Selman, and H. Levesque (1992). Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, San Jose, CA.

Monien, B. and D. Janssen (1977). Uber die komplexitat der fehlerdiagnose bei systemen. *Ang. Mathem. Mech.*. T315-T317.

Monien, B. and E. Speckenmeyer (1985). Solving satisfiability in less than 2^n steps. *Discrete Applied Math 10*, 287–295.

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

Purdom, Jr., P. W. (1983). Search rearrangement backtracking and polynomial average time. *Artificial Intelligence 21*, 117–133.

Purdom, Jr., P. W. and C. Brown (1987). Polynomial-average-time satisfiability problems. *Information Sciences 41*, 23–42.

Schaefer, T. (1978). The complexityof satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on the Theory of Computing*, New York, pp. 216–226. Association for Computing Machinery.

Selman, B., H. Levesque, and D. Mitchell (1992). A new method for solving hard satisfiability problems. In *AAAI-92*, San Jose, CA.

Tseitin, G. (1968). *On the complexity of derivation in propositional calculus.* New York: Consultants Bureau. In; *Studies in Constructive Mathmatics and Mathematical Logic – Part II, A.O. Slisenko (ed.).*

Tukey, J. (1960). *A Survey of Sampling From Contaminated Distributions.* Palo Alto, CA: Stanford University Press. I. Olkin, S. Ghurey, W. Hoeffdine, W. Madow and H. Mann (Eds.).

Van Gelder, A. (1988, October). A satisfiability tester for non-clauses propositional calculus. Volume 79.

Waltz, D. (1975). *Understanding Line Drawings of Scenes with Shadows.* New York: McGraw-Hill. Patrick H. Winston, Ed.

Yamasaki, S. and S. Doshita (1986). Resolution deduction to detect the satisfiability for another class including non-horn sentences in propositional logic. *Information Processing Letters 23*, 201–207.

Zabih, R. and D. McAllester (1991). A rearrangement search strategy for determining propositional satisfiability. In *Proc. IJCAI-91*, pp. 155–160.