Why solutions can be hard to find

A featural theory of cost for a local search algorithm on random satisfiability instances

Josh Singer

Ph.D.
University of Edinburgh
2001

Abstract

The local search algorithm WSAT is one of the most successful algorithms for solving the archetypal NP-complete problem of satisfiability (SAT). It is notably effective at solving RANDOM-3-SAT instances near the so-called 'satisfiability threshold', which are thought to be universally hard. However, WSAT still shows a peak in search cost near the threshold and large variations in cost over different instances. Why are solutions to the threshold instances so hard to find using WSAT? What features characterise threshold instances which make them difficult for WSAT to solve?

We make a number of significant contributions to the analysis of WSAT on these high-cost random instances, using the recently-introduced concept of the *backbone* of a SAT instance. The backbone is the set of literals which are implicates of an instance. We find that the number of solutions predicts the cost well for small-backbone instances but is much less relevant for the large-backbone instances which appear near the threshold and dominate in the overconstrained region. We undertake a detailed study of the behaviour of the algorithm during search and uncover some interesting patterns. These patterns lead us to introduce a measure of the *backbone fragility* of an instance, which indicates how persistent the backbone is as clauses are removed. We propose that high-cost random instances for WSAT are those with large backbones which are also backbone-fragile. We suggest that the decay in cost for WSAT beyond the satisfiability threshold, which has perplexed a number of researchers, is due to the decreasing backbone fragility. Our hypothesis makes three correct predictions. First, that a measure of the backbone robustness of an instance (the opposite to backbone fragility) is negatively correlated with the WSAT cost when other factors are controlled for. Second, that backbone-minimal instances (which are 3-SAT instances altered so as to be more backbone-fragile) are unusually hard for WSAT. Third, that the clauses most often unsatisfied during search are those whose deletion has the most effect on the backbone.

Our analysis of WSAT on RANDOM-3-SAT threshold instances can be seen as a *featural theory* of WSAT cost, predicting features of cost behaviour from structural features of SAT instances. In this thesis, we also present some initial studies which investigate whether the scope of this featural theory can be broadened to other kinds of random SAT instance. RANDOM-2+p-SAT interpolates between the polynomial-time problem RANDOM-2-SAT when $p = 0$ and RANDOM-3-SAT when $p = 1$. At some value $p = p_0 \approx 0.41$, a dramatic change in the structural nature of instances is predicted by statistical mechanics methods, which may imply the appearance of backbone fragile instances. We tested NOVELTY$^+$, a recent variant of WSAT, on RANDOM-2+p-SAT and find some evidence that growth of its median cost changes from polynomial to superpolynomial between $p = 0.3$ and $p = 0.5$. We also find evidence that it is the onset of backbone fragility which is the cause of this change in cost scaling: typical instances at $p = 0.5$ are more backbone-fragile than their counterparts at $p = 0.3$. NOT-ALL-EQUAL (NAE) 3-SAT is a variant of the SAT problem which is similar to it in most respects. However, for NAE 3-SAT instances no implicate literals are possible. Hence the backbone for NAE 3-SAT must be redefined. We show that under a redefinition of the backbone, the pattern of factors influencing WSAT cost at the NAE RANDOM-3-SAT threshold is much the same as in RANDOM-3-SAT, including the role of backbone fragility.

Acknowledgements

First of all many thanks are due to my two encouraging supervisors. Alan Smaill has supervised this work as principal supervisor, through thick and thin, since October 1997. Ian Gent of St. Andrews University has very kindly acted as second supervisor for the second half of my time at Edinburgh. Thanks to EPSRC for funding me through studentship 97305799.

I have also had a lot of support from the APES cross-university research group of which I have been a member and would like to thank them very much for the many discussions on this work as it has developed and for the encouraging and open atmosphere within the group. In particular Toby Walsh and Pat Prosser have contributed a lot of useful comments. Thanks are also due to Joe Culberson for valuable discussions about NOT-ALL-EQUAL 3-SAT during and after his visit to Scotland. Also thanks to Adam Beacham for contributing to this discussion.

I would like to extend thanks to my erstwhile office-mate Andrew Tuson of City University for a lot of discussion, feedback and ideas especially in the early stages and for inviting me to City for a month-long research visit there in June 1999. I would like to thank for their feedback and encouragement the anonymous reviewers of a rejected submission to the Journal of Automated Reasoning, of the article in the Journal of Artificial Intelligence Research and of the paper in ECAI-2000. Also, thanks are due to Holger Hoos for useful feedback over email and when we met at ECAI and to Thomas Stützle and Edward Tsang for their feedback while we were at ECAI. Thanks to Allen van Gelder for making his MODOC system available and to Steve Prestwich for some interesting suggestions about non-random backbone fragile instances. Thanks to fellow Ph.D student Ben Curry for encouragement and feedback and to Rachel Steven for encouragement and proofreading.

Finally, thanks to my two examiners Chris Williams (internal) and Barbara Smith (external) for a very thorough and constructive examination of the thesis.

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Contents

List of Figures

Chapter 1

Introduction

In Artificial Intelligence (AI) combinatorial search is a major theme. It is often necessary to configure an agent's environment so as to achieve some goal. Typically, certain combinations of elementary configurations are impossible or undesirable because of restrictions imposed by the nature of the goal. Therefore a search of the space of configurations is necessary to find a configuration which satisfies the goal.

In this thesis a *problem* is an abstract, though unambiguous definition of a task to be solved. The problem description specifies the form the input will take and what properties the output should have. A *problem instance* is a concrete instantiation of the problem i.e. a particular piece of input in the form specified in the problem description. For combinatorial search problems, a *solution* to a problem instance is a particular configuration which satisfies the goal.

This thesis concerns the study of certain existing algorithms developed by AI researchers to find solutions to instances of a simple combinatorial search problem: satisfiability. Satisfiability is interesting because of its generality: many tasks of interest can be naturally encoded as instances of satisfiability. It is also in a sense representative of the difficult NP-complete class of problems.

There are various *generation methods* by which instances of satisfiability can be created. Some recent research has focused on *stochastic generation methods.* These are procedures (often extremely simple) for creating instances randomly. Using stochastic generation methods, the experimenter sets some parameters governing the overall size and shape of the instance to be generated but then the details of the instance are set

randomly.

Stochastic generation methods have been devised and investigated for both satisfiability and a number of other important combinatorial problems. Stochastic generation methods have exhibited some interesting behaviour. Under a given set of parameter settings, some properties of instances may occur more or less frequently. It is interesting to study the distribution of instance properties under different parameter settings. Suppose some of the parameters are held constant but one of them, the *control parameter*, is varied. For some cases, it has been shown that as this parameter is varied the frequency of instances which have solutions changes rapidly from all to none. This change in the frequency of instances with solutions is known as a *threshold* phenomenon. The sudden change is said to occur in the *threshold region* of the control parameter's range.

A fascinating further discovery was that in many problems, the frequency of instances which are difficult to solve is higher in the threshold region than elsewhere in the control parameter range. While some algorithms can quickly solve instances from the non-threshold region, none has yet been found which can efficiently deal with the threshold instances. As a consequence of this, it has been conjectured that in the satisfiability problem the instances from the threshold region are *inherently* hard in the typical case. This is taken to mean (given $P \neq NP$) that no algorithm can exist which typically finds solutions to instances from this region with a reasonable amount of computational effort. In summary, whereas the NP-completeness of a problem demonstrates that there must be some instances where solutions are hard to find, the discovery of the threshold phenomenon actually locates these instances.

One type of algorithm which experiences these difficulties in the threshold region but which performs well away from the threshold is the *local search* class of algorithms. These techniques, which share a particular common architecture, are the algorithms studied in this thesis.

The aim of this thesis is this: to extend our knowledge of what *characteristics* of threshold region instances, absent in non-threshold instances, cause the performance of local search to degrade. In other words, to answer the question *Why are threshold*

instances hardest for local search? Another objective is to elucidate the mechanism by which the characteristics of the threshold instances cause the poor performance of local search.

The thesis relies exclusively on empirical methods. Cohen (1995) proposes that empirical work in AI should lead to *featural theories* concerning the behaviour of AI systems. These are not formal mathematical theories. They are statements which characterise the behaviour of an AI system given the features of its architecture and of the task that is to be performed. Featural theories are obtained by careful observation of the combinations of features which coincide with a particular kind of behaviour. This leads to a *predictive* featural theory of the form:

> *If an AI system's architecture/task has features* $F_1, \ldots, F_i$ *then its behaviour will have features* $G_1, \ldots, G_i$.

A featural theory may optionally have an explanatory clause giving the mechanism by which the behavioural features are produced by the combination of architecture and task features. This is an *explanatory* featural theory.

The aim stated above will therefore be satisfied by way of a featural theory: an evidence-based statement of which properties are responsible for the hardness of threshold instances.

What is the significance of this project in the broader field of research into search algorithms? If the threshold region is as inherently and as universally hard as is conjectured, then the poor performance of local search is not merely a failing of the local search architecture. Rather, the properties of the threshold region are responsible. Hence, by understanding the threshold region and how its properties affect the operation of local search algorithms, we can move towards understanding why solutions can be hard to find in more general terms.

We now outline the structure of the thesis to show how the featural theory will be developed.

- Chapter 2, *Satisfiability and Local Search*, gives the key elements of background knowledge, definitions and accomplishments of previous work which are required

for the rest of the thesis. Topics such as the threshold region and the architecture of local search algorithms are covered in detail here.

- In Chapter 3, *Local Search on* RANDOM-3-SAT, we analyse in detail the behaviour of the local search algorithm WSAT/SKC on threshold instances of the RANDOM-3-SAT generation method. We confirm some patterns of behaviour which have been revealed by past research. We also discover a number of other patterns, leading to a richer characterisation of WSAT/SKC behaviour in the threshold region. In particular we investigate in detail the relationship between performance and certain other instance properties (the *backbone* and the number of solutions) which have been studied in isolation before. We also undertake a "microscopic" study of the internal state of the algorithm during search on threshold instances. By the end of Chapter 3 we have a detailed picture of the performance and internal state behaviour of WSAT/SKC on threshold instances. However, some key behavioural patterns remain unexplained.

- The aim of Chapter 4, *The Backbone Fragility Hypothesis*, is to explain these unexplained patterns. We introduce a new property of SAT instances – *backbone fragility*. We then propose a hypothesis, based on this property, which endeavours to account for the unexplained behaviour of WSAT/SKC. In the remainder of Chapter 4 we test experimentally a number of predictions of the hypothesis, with positive results. We therefore have a consistent and relatively complete account of why the performance of WSAT/SKC degrades in the threshold region but is good elsewhere. The featural theory predicts detailed features of WSAT/SKC behaviour on threshold RANDOM-3-SAT instances.

According to Cohen, the utility of a featural theory is extended if its *scope* (i.e. the range of possible AI systems and tasks which it covers) is broadened. While Chapters 3 and 4 form the heart of the thesis, Chapters 5 and 6 contain more preliminary and peripheral results in which we investigate broadening the scope of the theory to encompass different kinds of instances.

- In Chapter 5, *Local Search on* RANDOM-2+p-SAT, we look at the behaviour of a WSAT variant on threshold instances of the RANDOM-2+p-SAT generation

method. This generation method allows interpolation between the RANDOM-2-SAT problem, where the threshold region is not hard and RANDOM-3-SAT which contains a threshold region which is conjecturally universally hard. We report initial experiments which are evidence that a hard threshold for local search emerges during the interpolation between RANDOM-2-SAT and RANDOM-3-SAT and that this is because of the emergence of backbone fragility.

- In Chapter 6, *Local Search on* NOT-ALL-EQUAL 3-SAT, we extend our theory to a combinatorial search problem which, though identical to SAT in most respects, has one key difference. A central concept in the featural theory is that of the backbone, which is a property of instances. In NOT-ALL-EQUAL 3-SAT the backbone is fundamentally different in nature from the backbone in SAT. We show that a number of results, including the key predictions relating performance and backbone fragility, carry over to the behaviour of WSAT/SKC in the threshold region of NOT-ALL-EQUAL RANDOM-3-SAT.
- In Chapter 7, *Conclusions and Further Work*, we summarise the featural theory which has been developed. We conclude with a number of suggestions for both extending the theory and exploiting it.

Chapter 2

Satisfiability and Local Search

In this chapter the aim is to familiarise the reader with the concepts which are used in the rest of the thesis. Section 2.1 defines the satisfiability problem and various related concepts. Section 2.2 discusses the study of hard instances of the satisfiability problem, particularly highlighting the RANDOM-k-SAT stochastic generation method. Section 2.3 introduces local search: an algorithmic paradigm used to solve the satisfiability problem. This section also gives details of certain important variants of local search which we and other authors have analysed. Section 2.4 focuses on some particular cases of previous work which aimed at understanding the operation of local search procedures on RANDOM-k-SAT instances and which were a bedrock on which our central ideas were then constructed. Section 2.5 reviews the Davis-Putnam-Logemann-Loveland algorithm for satisfiability which is used extensively for the calculation of instance properties in subsequent chapters. A brief summary is given in Section 2.6.

2.1 Propositional satisfiability

The logic variant important to this thesis is *propositional logic*. Formulas of propositional logic are written using a set of *propositional variables* V and *logical connectives* $\neg$, $\wedge$, $\vee$ and $\rightarrow$, known respectively as negation, conjunction, disjunction and implication. We will use the letters x, y and z, possibly subscripted, for variables. The syntax of propositional logic formulas is as follows:

- Any propositional variable in V is a formula.

- Where A is a formula, $\neg A$ is a formula.
- Where A and B are formulas, $A \wedge B$, $A \vee B$ and $A \rightarrow B$ are formulas.

In addition to these symbols, parentheses will be used to resolve syntactic ambiguity where necessary. In this thesis, as with much of the work on the computational aspects of propositional logic, we will be wholly concerned with *clausal* formulas, which we now define.

A *literal* is a propositional variable x where $x \in V$ or the negation of one: $\neg x$ where $x \in V$. l, possibly subscripted, will be used for literals. Let $l_1, l_2, \ldots, l_k$ be literals. A k-clause, or clause of length k with literals $l_1, l_2, \ldots, l_k$ is the formula $l_1 \vee l_2 \vee \ldots \vee l_k$. The letter c, possibly subscripted, will range over clauses. A *clausal formula* with m clauses $c_1, c_2, \ldots, c_m$ is the formula $c_1 \wedge c_2 \wedge \ldots \wedge c_m$. The letter C, possibly subscripted, will range over clausal formulas. Arbitrary formulas of propositional logic may be converted into equivalent clausal formulas by a series of rewriting operations. Although the details of these operations are not relevant here, this is one important property of the clausal form.

Generally, the *semantics* of a logic governs the meaning of sentences with respect to a *domain* : the portion of the world in which we are interested. The object which gives a sentence its meaning is an *interpretation.* In propositional logic we make a minimal ontological commitment about the domain. We simply insist that there are propositions in the domain and that each of these is either true or false. Therefore, in propositional logic, an interpretation $\mathcal{T}$ is simply a *truth assignment* mapping V to the set of *truth values* {**true**, **false**}. Hence $\mathcal{T}(x)$ denotes the truth value which the variable x is mapped to under $\mathcal{T}$. Given an interpretation, the truth value of a formula is calculated recursively on its syntactic structure using the standard truth tables for the connectives.

A truth assignment which makes the formula **true** is a *solution* to the formula[1]. If all possible truth assignments for a formula are solutions, the formula is *valid.* If there exist any solutions to a formula then it is *satisfiable.* If every truth assignment gives the value **false** for the formula (i.e. there are no solutions) then the formula is *unsatisfiable.*

[1] Elsewhere such a truth assignment is called a model or a satisfying assignment.

A clause of a clausal formula C is *satisfied* under a truth assignment $\mathcal{T}$ if at least one literal evaluates to **true** and *unsatisfied* otherwise. Note that a truth assignment $\mathcal{T}$ is a solution to a clausal formula C iff all clauses of C are satisfied under $\mathcal{T}$.

SAT refers to the decision problem of satisfiability: given a clausal formula C as input, decide whether there exists any solution to C. Hence a SAT instance is a clausal formula. k-SAT is the special case of SAT where the clauses of the input formula are of length at most k. A more useful but in important ways an equivalent problem to these decision problems is that of finding a solution if there is one. In this thesis we focus on algorithms for finding solutions.

The use of SAT algorithms is now routine in several different areas of AI. These include finite mathematics (Slaney *et al.* 1995; Zhang *et al.* 1996), planning (Kautz and Selman 1992, 1996), electronic design automation (Guerra e Silva *et al.* 1999) and network design and routing (Kautz *et al.* 1997). The general approach is as follows:

1. Encode the various constraints from the problem domain as a propositional formula A, such that A is satisfiable if the task is solvable and so that a solution to A encodes a solution to the task.
2. Convert A into an equivalent clausal formula C.
3. Use a generic SAT procedure which operates on clausal formulas to find a solution to C or prove its unsatisfiability.
4. If a solution to C was found, construct a solution to the task from it.

This simple methodology (the SAT-encoding approach) has been found to be quite effective in practice, often competitive with domain-specific methods e.g. Kautz and Selman (1996). Due to the NP-completeness of SAT, any instance of a problem in NP can be encoded as a SAT instance and this is often although certainly not always a reasonable approach.

2.2 The hardness of satisfiability

In this section we motivate the study of the "hard instances" predicted by NP-completeness theory and review work on stochastic instance generation methods for SAT.

2.2.1 What are hard instances and why should we study them?

The research question of this thesis is : *Why can solutions to* SAT *instances be hard to find for local search algorithms?* We now discuss hard instances of the SAT problem.

The NP-completeness of SAT implies (assuming P $\neq$ NP) that there can be no deterministic polynomial-time decision procedure. Alternatively, the implication is that any decision procedure must take superpolynomial time in the worst case. In principle we could collect these worst cases together in a class of their own. So, for any algorithm, there must be a subset of all SAT instances, containing some instances of every size, such that the best run time of the algorithm on instances from the subset grows superpolynomially with the size of the instances. This subset can be termed a "hard class of instances" of the algorithm.

It is now a standard research technique to investigate hard instances for different algorithm classes. Cook & Mitchell, introducing their survey article of this area (Cook and Mitchell 1997) describe the motivation for this kind of research:

> "the hard inputs for a class of algorithms characterize the limitations of those algorithms, and point up where additional research is needed."

Underlying the work presented in this thesis is the idea that the nature of hard instances for current algorithms and the mechanisms by which current algorithms are caused to fail on hard instances should be understood before substantial improvements to the algorithms can be made.

2.2.2 Stochastic SAT instance generation methods

There are special cases of the SAT problem which are in P: the best known are 2-SAT and Horn-SAT (where each clause has at most one negation). In these cases, by using

an appropriate algorithm, hard instances never occur. However, it is often the case that our task cannot be encoded as an instance of one of these special cases and so the possibility of hard instances is a reality.

The NP-completeness of SAT is a *worst case* analysis. We may be interested in whether the instances generated by a particular method will cause problems for our algorithm. The NP-completeness may not be a practical problem in SAT applications unless a hard class of instances for our algorithm (as defined above) forms a significant part of the class of instances generated by our instance generation method. There are two ways this could happen. The mean cost to run the algorithm on the instances could grow superpolynomially with n. In this case we say the class is "hard in the average case". Another possibility is that the median cost could be superpolynomial. In this case we say the class is "hard in the typical case". Whether or not an instance generation method is hard in the average or typical case is relative to a particular algorithm.

Researchers have looked at the cost to solve randomly generated SAT instances. These instances can be generated in different ways. According to experiments, some random generation methods give a class of instances which are hard in the average or typical case for an algorithm, while others give classes of instances which are not hard. This is of theoretical and practical interest. The theoretical interest arises because generating instances in some ways can lead to instances which appear to be hard in the average/typical case for many algorithms. Other generation methods result in easy instances. This helps shed light on what properties cause the cost to be high. The practical interest is as an unlimited source of test inputs for comparing algorithms. In fact the use of these hard random instances for evaluating SAT algorithms is now routine and has been used to identify several techniques which were later used successfully on practical AI tasks.

Early work generating random instances used stochastic generation methods which were later proven by Franco and Paull (1983) not to be hard in the average case. More recent experimental work (Cheeseman *et al.* 1991; Mitchell *et al.* 1992; Selman *et al.* 1996) identified the much more interesting RANDOM-k-SAT generation method, which is the subject of the remainder of this section.

```
procedure RANDOM-k-SAT(n, m)
    V := a set of n propositional variable symbols {x_1, ..., x_n}
    for i = 1 to m
        Select k distinct variables y_1, ..., y_k at random from V
        for j = 1 to k
            with probability 1/2
                l_j := ¬y_j
            otherwise
                l_j := y_j
            end with
        end for
        c_i := l_1 ∨ ... ∨ l_k
    end for
    return c_1 ∧ ... ∧ c_m
```

Figure 2.1: The RANDOM-k-SAT generation method.

2.2.3 The RANDOM-k-SAT generation method

RANDOM-k-SAT instances are generated using two parameters, the number of variables n and the number of clauses m. Pseudocode[2] for the procedure defining the RANDOM-k-SAT generation method is given in Figure 2.1.

We have a set of n variable symbols V. To generate an instance, each clause is independently chosen by selecting as its literals a subset of k distinct variables from V. Each such subset has equal probability of selection. We then independently negate each literal in the clause with probability $\frac{1}{2}$. Note that there is no guarantee that all variables are mentioned in the instance or that it will not contain duplicate clauses. Most work has studied RANDOM-3-SAT, since 3 is the smallest value of k for which k-SAT is NP-complete (Cook 1971).

2.2.4 The threshold phenomenon in RANDOM-k-SAT

RANDOM-k-SAT is often analysed by fixing n and varying the ratio m/n, the control parameter. The probability of generating a satisfiable instance then depends on the

[2] One non-standard control flow feature added to the standard pseudocode in this thesis is **with probability** p ... **otherwise** ... **end with** to allow stochastic non-determinism, which is a feature of many of the relevant computational procedures.

value of clauses to variables m/n. At low values of m/n (from 0 to around 4 for $k = 3$, $n = 100$ for example) a satisfiable instance is almost certain to be generated. This region of the control parameter's range is known as the *underconstrained* region. Then, over a region known as the *threshold region* (from about 4 to 5 for $k = 3$, $n = 100$) the probability of satisfiability falls sharply to almost zero. The point at which 50% of the instances are satisfiable is often used as a marker of the location of the threshold region. Thereafter, for higher values of m/n, in the *overconstrained* region, an unsatisfiable instance is almost certain to be generated. Crawford & Auton carried out an extensive experimental analysis of this phenomenon in RANDOM-3-SAT (Crawford and Auton 1996). The threshold region narrows in terms of m/n as n is increased. The location of the threshold region appears to converge to a constant value for m/n of about 4.25 for RANDOM-3-SAT. It is unknown analytically whether the location of the threshold region converges to a constant in the asymptotic limit and if so, what the value of the constant is. The threshold region appears to be very difficult to analyse. However, certain analytic results have been possible. The asymptotic location of the threshold region has been bounded by constants: Achlioptas and Sorkin (2000) have now shown a strict lower bound of 3.26 while Dubois *et al.* (2000) have very recently shown a strict upper bound of 4.506. Friedgut (1999) has also recently shown that the threshold is indeed sharp: the width of the threshold region narrows to nothing in terms of m/n as n tends to infinity. Friedgut's result allows for the location to vary with n (within the bounds) but it is widely conjectured to converge to a constant.

2.2.5 The hardness of RANDOM-k-SAT threshold instances

The threshold phenomenon relates directly to the hardness of the instance generation method. Early experimental work by Mitchell *et al.* (1992) examined the average search cost for the DPLL[3] decision procedure (Davis and Putnam 1960; Davis *et al.* 1962), on RANDOM-3-SAT instances. Mitchell *et al.* found that solving these instances with DPLL is easiest in the average case in the underconstrained region. Interestingly, the average cost to solve then rises as m/n is increased and peaks very near to the

[3] Although often referred to as the DP (Davis-Putnam) procedure, the algorithm incorporates techniques due to both Davis and Putnam (1960) and to Davis, Logemann & Loveland (Davis *et al.* 1962) and so we refer to it as DPLL. This algorithm is reviewed in detail in Section 2.5.

point of 50% probability of satisfiability. Thereafter the average cost decays again as we move into the overconstrained region. A similar pattern was found in the cost of two substantially different decision procedures by Larrabee and Tsuji (1992). Cook and Mitchell (1997) conjecture that the peak will occur for all reasonable decision procedures. Due to the cost peak, RANDOM-k-SAT instances from the threshold region are often the object of study. Hence the term "threshold RANDOM-k-SAT" will refer to the RANDOM-k-SAT method with m/n set such that the probability of generating a satisfiable instance is as close to 50% as can be. Threshold RANDOM-k-SAT has just one parameter n which is proportional to the number of clauses in the formula since at the threshold m/n is $O(1)$.

There have also been some rigorous results concerning this pattern of hardness. Frieze and Suen's lower bound (Frieze and Suen 1996) on the threshold location for RANDOM-3-SAT was shown by exhibiting an algorithm which with probability approaching 1 as $n \to \infty$ returns a solution in polynomial time when $m/n < 3.003$. Hence there is an algorithm for which solution finding in this region is provably easy in the typical case. Solving satisfiable RANDOM-3-SAT problems is also provably easy if we move far enough into the overconstrained region. Gent (1998), for example, gave a simple linear time algorithm which, in all but an exponentially small proportion of cases returns a solution to satisfiable instances when m/n is $\Omega(\log n)$. Since the transition occurs when m/n is $O(1)$ this refers to the overconstrained region, where satisfiable instances are vanishingly rare. Unsatisfiable instances from the threshold region are provably hard in the average case for any procedure based on the resolution inference system. This was demonstrated by Chvátal and Szemerédi (1988) based on the exponential minimum length of resolution proofs. Beame *et al.* (1998) prove lower and upper bounds on the lengths of resolution proofs of unsatisfiable RANDOM-3-SAT, including a proof that once there are $n^2/\log n$ clauses, there is an algorithm which can almost certainly find an unsatisfiability proof in polynomial time. This confirms the eventual disappearance of hard unsatisfiable instances as m/n is increased.

2.3 Local search for satisfiability

An algorithm is *complete* if it is guaranteed to solve any problem instance (by returning a solution or proving none exists). The fact that simple SAT generation methods such as RANDOM-3-SAT are apparently universally hard has motivated researchers to relax their insistence that algorithms must be complete. Instead, in an attempt to find algorithms which were more efficient on the hard instances, *incomplete* algorithms, which do not carry the guarantee, have been investigated.

Local search describes a particular class of (typically) incomplete algorithms. In this section we review the motivation for and the details of these algorithms, while postponing their analysis until Section 2.4 of this chapter. We give details only of the local search algorithms of direct relevance to this thesis.

2.3.1 Pseudocode preliminaries

Certain sub-procedures which will be useful in the discussion of the algorithms are defined in Figure 2.2. We also adopt the convention that sub-procedures terminated with an asterisk (SELECT-VARIABLE-$*$ for example) can be instantiated in different ways, allowing a family of variant algorithms. The different possible instantiations will be denoted by replacing the asterisk with different labels, for example SELECT-VARIABLE-GSAT will refer to the GSAT instantiation for SELECT-VARIABLE-$*$.

2.3.2 What is local search?

Figure 2.3 describes the most basic architecture shared by local search algorithms for satisfiability.

The search begins by selecting an initial truth assignment. This can be done in various ways according to the instantiation of SELECT-TRUTH-ASSIGNMENT-$*$. In all the local search variants discussed in this chapter SELECT-TRUTH-ASSIGNMENT-$*$ is instantiated by SELECT-TRUTH-ASSIGNMENT-RANDOM.

The technique proceeds by selecting one of the variables and reversing its truth value in the assignment $\mathcal{T}$. This step is called a *flip* and is generally considered to be the

SELECT-TRUTH-ASSIGNMENT-$*(C)$ returns a truth assignment to the variables mentioned in the formula C.

SELECT-TRUTH-ASSIGNMENT-RANDOM(C) returns a truth assignment to the variables mentioned in the formula C. Each variable is mapped to **true** with probability $1/2$ and **false** otherwise.

SELECT-VARIABLE-$*(\mathcal{T}, C)$ returns a variable mentioned in the formula C.

FLIP$(\mathcal{T}, v)$ returns a truth assignment which is equal to $\mathcal{T}$ except that in the returned truth assignment, v is mapped to the opposite truth value to $\mathcal{T}(v)$.

SOLUTION$(\mathcal{T}, C)$ returns true if $\mathcal{T}$ is a solution to C, false otherwise.

SATISFIED-CLAUSES$(\mathcal{T}, C)$ returns the sub-formula of C consisting of the clauses which are satisfied under truth assignment $\mathcal{T}$.

UNSATISFIED-CLAUSES$(\mathcal{T}, C)$ returns the sub-formula consisting of the clauses of C which are unsatisfied under truth assignment $\mathcal{T}$.

SELECT-CLAUSE-AT-RANDOM(A) returns a clause from the formula A with equal probability of each.

SELECT-VARIABLE-FROM-CLAUSE-$*(c, \mathcal{T}, C)$ returns one of the variables which appears in the clause c.

SELECT-VARIABLE-AT-RANDOM(S) returns one of the variables in the set S at random, with equal probability of each.

$A \cap B$ returns the formula consisting of the clauses which appear in both formula A and formula B.

$|A|$ returns the number of clauses in the formula A.

Figure 2.2: Sub-procedures used in definitions of local search algorithms.

```
procedure LOCAL-SEARCH(C, Max_flips)
    T := SELECT-TRUTH-ASSIGNMENT-*(C)
    for j = 1 to Max_flips
        v := SELECT-VARIABLE-*(T, C)
        T := FLIP(T, v)
        if SOLUTION(T, C)
            return T
        end if
    end for
    return "no solution found"
```

Figure 2.3: LOCAL-SEARCH, the basic architecture shared by local search SAT algorithms.

```
procedure LOCAL-SEARCH-WITH-RESTART(C, Max_tries, Max_flips)
    for i = 1 to Max_tries
        Local_search_output := LOCAL-SEARCH(C, Max_flips)
        if SOLUTION(Local_search_output, C)
            return Local_search_output
        end if
    end for
    return "no solution found"
```

Figure 2.4: LOCAL-SEARCH-WITH-RESTART, the local search SAT algorithm architecture incorporating a restart mechanism.

basic unit of cost for the algorithm. If $\mathcal{T}$ becomes a solution, we return it, otherwise we iterate, making more flips until a solution is found or some maximum number of flips is reached, in which case we abandon the search. The method used to select the variable at each point can also vary from algorithm to algorithm according to the instantiation of SELECT-VARIABLE-$*$.

In the early local search algorithms, on certain instances it was found that the search could become trapped such that $\mathcal{T}$ would never become a solution. The early algorithms avoided this by incorporating a *restart mechanism.* Pseudocode for local search with restart is given in Figure 2.4.

LOCAL-SEARCH-WITH-RESTART calls LOCAL-SEARCH as a sub-procedure. Each call to LOCAL-SEARCH is called a *try* and is limited in length to Max_flips. At most Max_tries tries are made. If any one of these finds a solution, it is returned.

If, for SELECT-TRUTH-ASSIGNMENT-$*$, we use some procedure which successively enumerates all possible truth assignments, LOCAL-SEARCH-WITH-RESTART is guaranteed to find a solution or (if 2^n tries fail) show that no solution exists. Such complete local search variants were investigated by Gent and Walsh (1993b). However, in most cases (and for all instantiations of local search relevant to this thesis) we use SELECT-TRUTH-ASSIGNMENT-RANDOM. Another example of a complete local search procedure is FILL due to Morris (1993) but this is mainly of theoretical interest as its space requirements are unfeasible. So, typically, local search is incomplete both because it cannot typically show unsatisfiability and because on satisfiable inputs it is not typically guaranteed to find a solution after a bounded number of steps.

```
procedure SELECT-VARIABLE-GSAT(T, C)
    V := the set of variables appearing in C
    for each variable x ∈ V
        Score[x] := | SATISFIED-CLAUSES(FLIP(T, x), C) |
    end for
    Possible_moves := the set of variables x ∈ V such that Score[x] is maximal
    v := SELECT-VARIABLE-AT-RANDOM(Possible_moves)
    return v
```

Figure 2.5: SELECT-VARIABLE-GSAT: the greedy variable selection strategy.

2.3.3 Why use local search?

Why sacrifice the guarantees given by completeness? The benefits of local search for satisfiability were discovered relatively recently by Selman *et al.* (1992) and independently by Gu (1992). They found that local search procedures were able to find solutions to certain large instances, including some from threshold RANDOM-3-SAT, which good complete methods failed to solve within a reasonable time. Therefore in any system where completeness may be sacrificed, local search procedures have an important role to play and this is why they have generated so much interest in recent years.

2.3.4 GSAT

GSAT, the local search procedure introduced by Selman *et al.* (Selman *et al.* 1992) was for some time the baseline algorithm and has been the subject of much investigation. GSAT stands for *greedy* SAT algorithm, which refers to its instantiation of SELECT-VARIABLE-$*$, given in Figure 2.5.

SELECT-VARIABLE-GSAT returns the variable which when flipped will result in the greatest increase (or smallest decrease) in |SATISFIED-CLAUSES($\mathcal{T}$, C)|. In the case of more than one variable having this property, the variable to flip is picked randomly.

2.3.5 WSAT

One property of GSAT and similar variants is that they consider all variables before choosing one to flip. By reducing the number of variables considered, the calculations

```
procedure SELECT-VARIABLE-WSAT(T, C)
    A := UNSATISFIED-CLAUSES(T, C)
    c := SELECT-CLAUSE-AT-RANDOM(A)
    v := SELECT-VARIABLE-FROM-CLAUSE-*(c, T, C)
    return v
```

Figure 2.6: The architecture for SELECT-VARIABLE-* which is shared by WSAT variants.

required to evaluate each flip can be reduced, so that more flips can be made in a given time, allowing the possibility of a successful search in less time. One possibility is to consider only those variables which occur in unsatisfied clauses under the current assignment. After flipping one of these variables, at least one clause must go from being unsatisfied to being satisfied and so flips are thought to be in a sense directed towards repairing the flaws in the current assignment (Selman *et al.* 1994; Gent and Walsh 1995).

A LOCAL-SEARCH variant using this approach proposed by Papadimitriou (Papadimitriou 1991, 1994) was called "the random walk algorithm". In Papadimitriou's random walk algorithm, the method used to select the variable to be flipped was a two stage process, first selecting at random an unsatisfied clause and then a variable mentioned in that clause. This two stage approach to SELECT-VARIABLE-* was adopted by Selman *et al.* and named the WSAT architecture (the W stands for *walk*). Pseudocode for SELECT-VARIABLE-WSAT is given in Figure 2.6. In Papadimitriou's random walk algorithm SELECT-VARIABLE-FROM-CLAUSE-*$(c, \mathcal{T}, C)$ returned a variable at random from those appearing in the clause c but generally in the WSAT family this sub-procedure may be instantiated in different ways.

Selman *et al.* used what was later called the SKC strategy (referring to the three authors Selman, Kautz and Cohen) for SELECT-VARIABLE-FROM-CLAUSE-*. Some preliminary experiments (Selman *et al.* 1994) with WSAT/SKC (the WSAT architecture with the SKC variable selection strategy described below) indicated that the WSAT approach could be significantly faster (in run time terms) than GSAT variants and so later research has tended to concentrate on the WSAT family.

Research has yielded some variable selection strategies for WSAT which form the pow-

```
procedure SELECT-VARIABLE-FROM-CLAUSE-SKC(c, T, C)
    Clause_vars := the set of variables appearing in c
    for each variable x ∈ Clause_vars
        Broken_clauses :=
                SATISFIED-CLAUSES(T, C) ∩ UNSATISFIED-CLAUSES(FLIP(T, x), C)
        Breaks[x] := |Broken_clauses|
    end for
    Breakless_vars := the set of variables v ∈ Clause_vars such that Breaks[v] = 0
    if Breakless_vars is not empty
        return SELECT-VARIABLE-AT-RANDOM(Breakless_vars)
    else
        with probability noise
            return SELECT-VARIABLE-AT-RANDOM(Clause_vars)
        otherwise
            Breakmin_vars := the set of variables v ∈ Clause_vars
                                such that Breaks[v] is minimal
            return SELECT-VARIABLE-AT-RANDOM(Breakmin_vars)
        end with
    end if
```

Figure 2.7: The SKC strategy for selecting a variable from a clause.

erful local search variants which will we will study in this thesis. We now give details of these variants: WSAT/SKC, WSAT/NOVELTY and WSAT/NOVELTY$^+$.

2.3.6 WSAT/SKC

The strategy SELECT-VARIABLE-FROM-CLAUSE-SKC which was first used by Selman *et al.* (1994) (the details were later given by Parkes and Walser (1996)) is presented in Figure 2.7.

The strategy is based on choosing a variable which minimises damage to the set of satisfied clauses. A clause which becomes unsatisfied due to a flip is said to be *broken* by the flip. If a variable appearing in c (the selected unsatisfied clause) can be flipped without breaking any satisfied clauses, one such variable is chosen. If there is no such variable, the behaviour is randomised between two strategies. It may select a variable from c which minimises the number of broken clauses or it may ignore this heuristic and make a purely random walk move, selecting a variable appearing in c at random. The SKC strategy has an additional parameter *noise* ($0 \leq$ *noise* ≤ 1) governing the

frequency of the random walk moves.

2.3.7 WSAT/NOVELTY

The local search variants discussed above are *memory-free*: each move takes no account of previous moves or previous assignments visited. It is thought that this property may sometimes cause local search algorithms to revisit non-solution assignments repeatedly. Gent and Walsh (1993b) showed that HSAT, a variant of GSAT which incorporated a simple memory mechanism outperformed its memory-free predecessor on a number of SAT instance classes including satisfiable threshold RANDOM-3-SAT. Various other authors have also suggested local search algorithms with memory features (Selman and Kautz 1993b; Morris 1993; Frank 1996b; Mazure *et al.* 1997) and these generally compare favourably with their memory-free counterparts.

NOVELTY is a quite recent WSAT variant. Its instantiation of SELECT-VARIABLE-FROM-CLAUSE-$*$ incorporates a simple memory mechanism. It is due to McAllester *et al.* (1997). The procedure is given in Figure 2.8. The basic selection criterion is to increase the number of satisfied clauses, without re-flipping the same variable in a row too often[4]. McAllester *et al.* found that WSAT/NOVELTY outperformed a number of other WSAT variants including WSAT/SKC on large threshold RANDOM-3-SAT instances.

2.3.8 Achieving probabilistic approximate completeness

If for a local search variant L there exists a satisfiable clausal formula such that if the initial assignment is set to a particular truth assignment, L will never find a solution without restarting, we say that L is *essentially incomplete*. If on the other hand when L is run on any satisfiable instance, beginning at any initial assignment the probability of L finding a solution tends to 1 as Max_flips tends to infinity, we say that L is *probabilistically approximately complete* (PAC). These terms are due to Hoos (1999a). Culberson and Gent (1999a) proved that WSAT/SKC is PAC for the 2-SAT case.

[4] McAllester *et al.* did not specify how to break ties between variables with equal *Score* when neither variable had been flipped in the current try. In our implementation we broke ties randomly in this situation, but for clarity's sake we have omitted this detail from the pseudocode.

```
procedure SELECT-VARIABLE-FROM-CLAUSE-NOVELTY(c, T, C)
    Clause_vars := the set of variables appearing in c
    for each variable x ∈ Clause_vars
        Score[x] := |SATISFIED-CLAUSES(FLIP(T, x), C)|
    end for
    Variable_ranking := a list of the variables of c sorted by decreasing Score[x],
                        breaking ties by placing the least recently flipped variable
                        first in the list
    if the first variable of Variable_ranking is not the most recently flipped variable
        return the first variable of Variable_ranking
    else
        with probability noise
            return the second variable of Variable_ranking
        otherwise
            return the first variable of Variable_ranking
        end with
    end if
```

Figure 2.8: The NOVELTY strategy for selecting a variable from a clause.

Hoos (1998) observed that his data suggested WSAT/SKC could be PAC. In every one of our experiments with WSAT/SKC on satisfiable instances a solution was found without restart, which is further evidence (though not proof) that WSAT is PAC.

Hoos (1998, 1999a) proved that certain WSAT variants including NOVELTY are essentially incomplete. Furthermore, Hoos found that some NOVELTY tries on RANDOM-3-SAT instances failed to find a solution after a very large number of flips. So, although NOVELTY is an efficient WSAT strategy for RANDOM-3-SAT instances, often finding a solution in much fewer flips than other variants, there are some tries where it fails to find a solution within a very long time, possibly due to its essential incompleteness.

Hoos (1999a) proposed a simple remedy to this problem: the addition of pure random walk with a small probability. The NOVELTY$^+$ strategy (the + indicates the addition of pure random walk) is given in Figure 2.9. The strategy has a parameter wp, which Hoos set to 0.01. This governs the probability of a random walk flip occurring. Note that NOVELTY$^+$ acts exactly like NOVELTY except that every so often it takes a pure random walk flip. Modifying any WSAT variant in this way guarantees probabilis-

```
procedure SELECT-VARIABLE-FROM-CLAUSE-NOVELTY+(c, T, C)
    with probability wp
        Clause_vars := the set of variables appearing in c
        return SELECT-VARIABLE-AT-RANDOM(Clause_vars)
    otherwise
        return SELECT-VARIABLE-FROM-CLAUSE-NOVELTY(c, T, C)
    end with
```

Figure 2.9: The NOVELTY+ strategy for selecting a variable from a clause.

tic approximate completeness[5]. Hoos (1998) demonstrated that NOVELTY+ did not have the incompleteness problems of NOVELTY, making it one of the best local search algorithms for this class to date.

Implementation note

All local search runs reported in this thesis were made using our own C implementations of WSAT/SKC and WSAT/NOVELTY+.

2.4 Understanding local search cost

We now review a number of recent previous studies whose results are relevant to our research area. We focus particularly on empirical studies of the behaviour of GSAT and WSAT variants on RANDOM-3-SAT instances. The discussion of some other studies of this type which are more closely related are postponed until later chapters so as to show more clearly the relationship between them and the thesis work.

Since local search is unable to show unsatisfiability, it is usually only run on satisfiable instances. Commonly, the method used is to generate RANDOM-3-SAT instances and filter out the unsatisfiable instances using a complete procedure. We will refer to instances generated in this way as satisfiable RANDOM-3-SAT.

[5] Observe that for any clause c currently unsatisfied under $\mathcal{T}$, there must be a variable x appearing in c such that for some solution $\mathcal{T}_{sol}$, $\mathcal{T}_{sol}(x)$ is the inverse truth value to $\mathcal{T}(x)$, since c must be satisfied under $\mathcal{T}_{sol}$. By repeatedly flipping the variable with this property $\mathcal{T}$ will become a solution. Hence any WSAT strategy which has a non-zero probability of flipping each variable in each unsatisfied clause is PAC.

2.4.1 The two phases of GSAT's search

Note that GSAT as described above allows *sideways moves*: flips which do not change the number of satisfied clauses. Where possible, GSAT will make such a move if no *upward move* (one which strictly increases the number of satisfied clauses) is possible. Selman *et al.* (1992) found early on that the sideways moves were in fact crucial to GSAT's success: if tries are abandoned when no upward moves are possible the performance degrades significantly.

Gent and Walsh (1993b) used a similar methodology: they varied other features of GSAT and found some interesting effects. Gent & Walsh found that greediness was not important. For example a variant CSAT (*cautious* SAT algorithm) is similar to GSAT except that if there are upward moves, it simply selects one at random from all possible upward moves. This is in contrast with GSAT which greedily maximises the increase in the number of satisfied clauses. If there are no upward moves, but there are sideways moves it selects one of these, otherwise it selects a *downward move* (decreasing the number of satisfied clauses) at random. CSAT and GSAT performed similarly on a range of problem instances including satisfiable threshold RANDOM-3-SAT. So, the variable selected does not need to be the one which increases the number of satisfied clauses the most. However, some kind of hill-climbing is required for a successful algorithm. Gent and Walsh also examined ISAT (*indifferent* SAT algorithm), which does not distinguish between sideways and upward moves: it selects a non-downward move at random from those available if there are any and a downward move otherwise. ISAT did not perform well in comparison to GSAT and CSAT, demonstrating that hill-climbing (the distinction between upwards and sideways moves) is important.

These observations are consistent with a *two-phase* model of GSAT's search behaviour on satisfiable threshold RANDOM-3-SAT- observed and discussed both by Selman and Kautz (1993a) and by Gent and Walsh (1993b), who also analysed this model in detail. Initially, there is a very rapid *hill-climbing phase*, where the proportion of satisfied clauses climbs from around 87.5% (under the initial random assignment, 7 out of 8 of all clauses will be satisfied on average) to nearly 100%. From here onwards the proportion of satisfied clauses climbs only very slowly. This is the *plateau phase*, so-called because

during most of this phase, only sideways moves are possible, with upwards moves becoming increasingly infrequent. When they do occur, upward moves almost always increase the number of satisfied clauses by only one. In threshold RANDOM-3-SAT several authors report that the plateau phase is much longer than the hill-climbing phase and constitutes most of the search e.g. Gent and Walsh (1993a).

The existence of the two phases suggests that incorporating some kind of hill-climbing is necessary so that the plateau phase is reached and so that during the plateau phase, possible upward moves are exploited when they arise. Sideways moves are necessary to allow exploration during the plateau phase. Greediness is not important because it can only reduce the length of the hill-climbing phase which only constitutes a small part of the search.

2.4.2 Local optima and how to avoid them

There exist SAT instances such that there is a set of assignments S where for each assignment $\mathcal{T} \in S$ there are no upward moves from $\mathcal{T}$, but such that there are sideways moves to other members of S. Such a set of assignments is a local optimum[6]. Moreover, as Frank *et al.* found, local optima occur quite frequently in RANDOM-3-SAT when the number of unsatisfied clauses is low (Frank *et al.* 1997). A simple consequence of the hill-climbing features of GSAT and CSAT is that any try which visits a local optimum containing more than one assignment will never succeed in finding a solution. Selman *et al.* recognised this, which is what led them to propose the restart mechanism for GSAT discussed in Section 2.3.2.

Other techniques have been proposed to avoid the problem of local optima: for example random noise and random walk (Selman *et al.* 1994), tabu lists (Mazure *et al.* 1997; McAllester *et al.* 1997) and clause weights (Morris 1993; Selman and Kautz 1993b; Frank 1996a). Each of these variants make what would be considered by GSAT downward moves under certain conditions and thus are in principle able to avoid being trapped, at least by traps of this kind.

[6] The term *local optimum* here is intended to correspond to a local minimum as defined formally by Frank *et al.* (1997). In other contexts the term local optimum has been used to denote a different concept e.g. a single assignment whose function value is strictly better than its neighbours but worse than the global optimum.

The re-randomisation of the truth assignment using the restart mechanism limits the search steps wasted when the algorithm is trapped in a local optimum. This way, the probability of finding a solution then approaches 1 as the number of tries approaches infinity. The question which then arises is when to restart the search or equivalently, how to set the Max_flips parameter.

Both Gent and Walsh (1993b, 1995) and Parkes and Walser (1996) investigated the effect of varying the Max_flips parameter in local search variants employing the restart technique. In all cases the measure of cost was the number of flips (totalled over possibly multiple tries) to solve satisfiable threshold RANDOM-3-SAT instances (setting Max_tries infinite). They looked at various local search variants. For the more greedy algorithms such as GSAT and a greedy WSAT variant, the effect on the cost was as follows: for low values of Max_flips the cost is very high. As Max_flips is increased there is a broad cost minimum after which cost begins to climb again, apparently linearly. However, for the variants which are not strictly greedy and also employ some other mechanism promoting occasional downward moves (including WSAT/SKC) the dependence of the cost on the value of Max_flips is somewhat different as follows. For low values of Max_flips the cost is very high. As Max_flips is increased, the cost falls to a certain level, and then stabilises. As long as Max_flips is high enough, the cost is insensitive to the Max_flips setting.

Hoos and Stützle (1998) found that many variants which are not strictly greedy and also employ a mechanism promoting occasional downward moves have another property of interest. Suppose several tries of such a local search algorithm L are run on a large instance from the RANDOM-3-SAT threshold region. The number of flips taken for L to find a solution is known as the *run length*. Hoos & Stützle observed that for these algorithms the distribution of the run length is consistent with an exponential distribution. Amongst other implications, along with the insensitivity of the algorithms to the setting of the Max_flips parameter, this result implies that the restart mechanism is not worthwhile in the presence of such other search control mechanisms. If the algorithm is PAC and run lengths are distributed exponentially, it is as good to run a single long try to completion than to randomly restart at any point. Since we use algorithms with these properties, in this thesis we do not employ the random

restart mechanism.

2.4.3 Local search cost growth

The interest in local search stems from its apparent superiority in terms of cost growth over complete procedures such as DPLL, especially on threshold RANDOM-3-SAT instances. In this section we focus on work relating to this aspect of local search : how the cost to find a solution grows with the instance size n.

Some analytic worst case results are known for GSAT and CSAT. Hirsch (2000), by exhibiting a hard instance which trapped the algorithms on all but an exponentially small fraction of initial assignments, proved a lower bound on the worst case of the expected run time on 3-SAT formulas of $\Omega(2^{0.0817n})$. Hirsch also proved an upper bound of $2^{(1-\frac{1}{(k-1)d+1})n}$ on the expected run time of CSAT on k-SAT formulas where each of the variables occurs at most d times.

To our knowledge the only analytic average-case result of applying local search algorithms to random instances of 3-SAT is that of Koutsoupias & Papadimitriou (Koutsoupias and Papadimitriou 1992). However, their proof applies to a variant which does not make sideways moves, running on satisfiable instances with $\Omega(n^2)$ clauses. Such instances must be in the overconstrained region, where satisfiable instances are vanishingly rare, since the threshold occurs when the number of clauses is $O(n)$. This result is not therefore applicable to algorithms which do make the important sideways moves when run on instances from the threshold region.

Knowledge of the behaviour of local search cost growth comes mainly from experimental work. On threshold RANDOM-3-SAT, average cost for DPLL variants (averaged over both satisfiable and unsatisfiable instances) appears to scale as a simple exponential (Crawford and Auton 1996; Gent *et al.* 1997). For local search algorithms, Gent *et al.* (1997) observed that average GSAT cost growth on satisfiable threshold RANDOM-3-SAT was consistent with a polynomial (at most quartic) in the underconstrained region. Parkes and Walser (1996) studied larger instances in the threshold region and found that WSAT/SKC average cost growth was not consistent with any function of the form an^b, but that certain functions which were superpolynomial but which grew

more slowly than a simple exponential, did fit the data.

It is important to stress that when comparing empirical cost scaling of local and complete search techniques, the local search case is derived only from satisfiable instances, whereas the complete search case is also based on unsatisfiable instances.

2.4.4 The cost peak and instance structure

As with complete procedures, there is a peak in the cost for local search procedures to solve the instances from the threshold region using several instance generation methods for constraint-like problems. Clark *et al.* (1996) showed a peak in the median and other percentiles for GSAT cost on RANDOM-3-SAT at $n = 100$, with m/n varying from 4.2 to 4.6. Both Clark *et al.* Hogg and Williams (1994) also found analogous easy-hard-easy patterns using different local search algorithms on other constraint satisfaction problems. If we increase m/n from 0 and measure the average cost for local search to solve all satisfiable instances, as we move through the underconstrained region, the average cost increases. In the threshold region, the cost peaks, and in the overconstrained region, the cost decreases again. As well as the apparent efficiency of local search techniques on these instances, this cost peak phenomenon has also of been of interest to researchers and is the main object of study in this thesis. As mentioned in Section 2.2.5 the threshold region is thought to be hard for any reasonable complete algorithm: apparently it is the hardest for the local search class.

Some research on the causes of the cost peak seen in the threshold region has focused on the nature of threshold instances. We now review work which identifies certain structural patterns in the nature of RANDOM-3-SAT instances which are related to instance hardness. Chapters 3 and 4 of this thesis can be seen as extending this work.

Implicates and backbones

Definition A clause c is an *implicate* of a formula C iff $C \rightarrow c$ is a valid formula. This means that the clause c is an implicate when its truth follows from assuming the truth of C. □

In the context of search for solutions to satisfiable formulas, understanding implicates

is important. This is because all implicates of a formula C are **true** under any solution to C. Therefore a search procedure which searches a region of truth assignments under which an implicate is **false** is searching in an unsatisfiable part of the space of assignments.

Definition The *backbone*[7] of a satisfiable formula C is the set of literals (i.e. clauses of length 1) which are implicates of C[8]. □

The backbone may equivalently be thought of as the set of literals which are **true** under all solutions. The *backbone size* (which we will sometimes denote *bsize*) is the number of literals in the backbone.

Schrag and Crawford (1996) investigated the evolution of implicates in RANDOM-3-SAT as m/n is increased. One of their discoveries was the fact that short implicate clauses (such as literals) only begin to appear near the threshold region. Parkes (1997) made a detailed experimental study of the evolution of the backbone in satisfiable RANDOM-3-SAT. Parkes' study demonstrated that in the underconstrained region, for most instances, only a very small fraction of the variables, if any, appear in the backbone. However, as m/n is increased, in the threshold region, a subclass of instances which have large backbones, mentioning around 75-95% of the variables, rapidly emerges. Interestingly, at no point are instances with medium-sized backbones particularly numerous; as random clauses are added, the size of the backbone tends to jump quickly from small to large. Also the fraction of variables mentioned in the backbones of the emergent large backbone subclass appears to be insensitive to n.

Soon after the control parameter is increased into the overconstrained region these large-backbone instances account for all but a few of the satisfiable instances. Once a formula has gained a large backbone, unsatisfiability is likely to follow after the addition of a few more clauses, since there is then a significant chance that all literals of an added clause are contradicted by literals in the backbone.

[7] Here, our use of the term "backbone" follows Monasson *et al.* (1999a,b). Although they do not formally define the backbone, their definition of the backbone fraction is equal to our definition of backbone size/n for satisfiable instances.

[8] Note that some other authors have found it more convenient to define the backbone of satisfiable instances as the set of negations of implicate literals.

In Figure 2.10 we present some experimental data to illustrate this transition from small-backbone instances in the underconstrained region to large-backbone instances in the overconstrained region (similar, more extensive data is presented by Parkes (1997)). Data was collected on satisfiable RANDOM-3-SAT with $n = 100$ using various values of m/n near the threshold. These ranged from the point of 90% satisfiability (4.03) to the point of 20% satisfiability (4.49). The backbone size was calculated for each of 5000 instances at each point. Figure 2.10 shows the change in frequencies of different backbone sizes as m/n is varied.

Parkes also showed that in the threshold region, the cost for the local search procedure WSAT/SKC is very strongly influenced by the size of the backbone: the hardest instances are those with large backbones. Parkes' finding is particularly important in understanding this thesis. This is because it suggests that the onset of local search hardness near the threshold region as m/n is increased may be due to the emergence of large-backbone instances at this point. This is the view of local search on RANDOM-3-SAT which we will develop in Chapters 3 and 4.

2.5 The Davis-Putnam-Logemann-Loveland (DPLL) procedure

In this section we review the DPLL algorithm for satisfiability because implementations of it are used as components of our system to calculate certain properties of SAT instances such as the backbone. In Section 2.5.1 we define some concepts related to the SAT problem which are relevant to this discussion of DPLL and in Section 2.5.2 we cover the details of variants of DPLL which were used.

2.5.1 DPLL preliminaries

Definition A *unit clause* is a clause of length 1: a clause containing one literal. □

Definition An *empty clause* is a clause of length 0: a clause containing no literals. If all literals are removed from a clause, it becomes an empty clause. Empty clauses evaluate to **false** under all interpretations. □

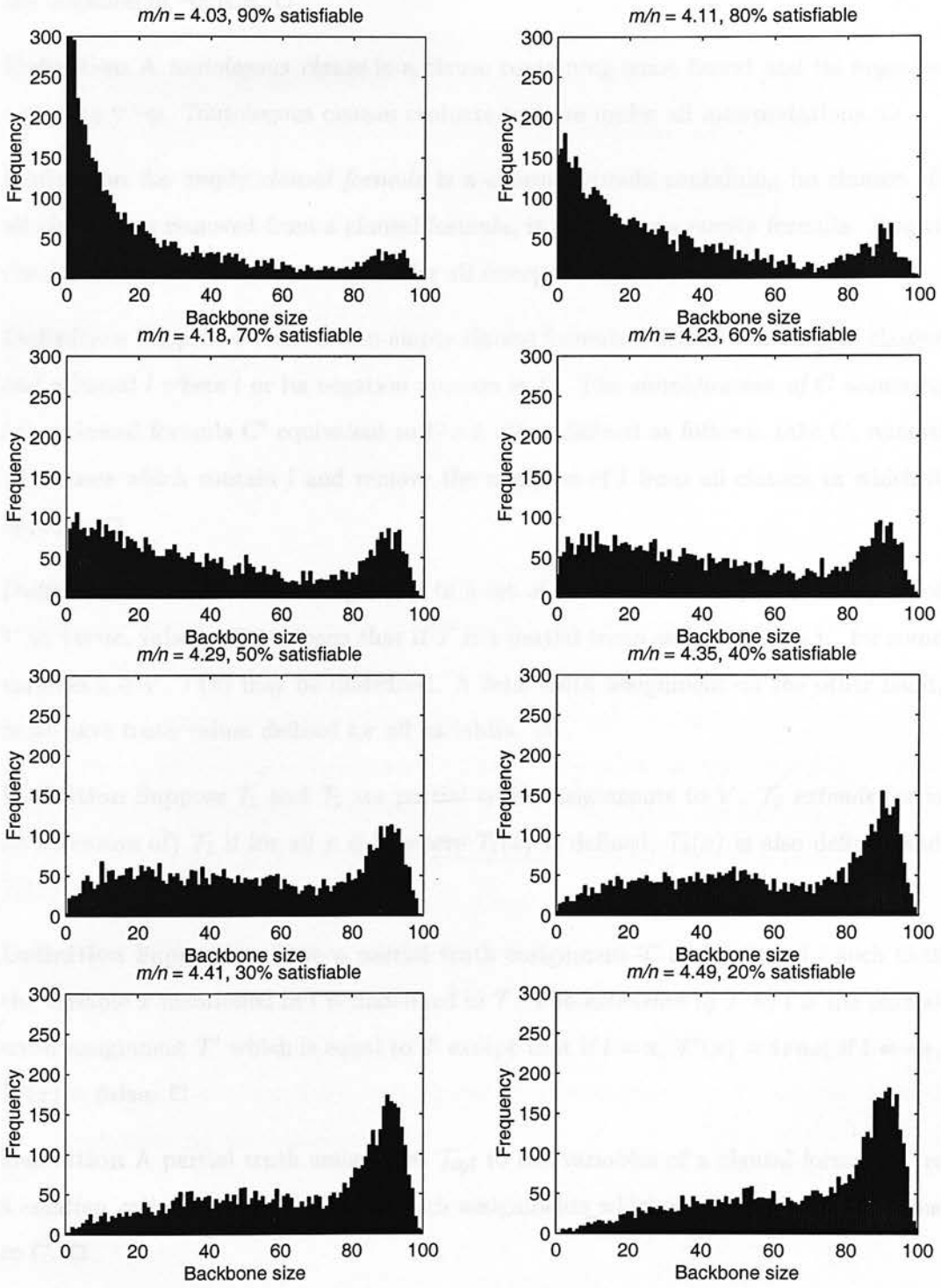

Figure 2.10: The evolution of the backbone size distribution as m/n is varied ($n = 100$).

Definition The *negation* of a literal is the opposite literal: the negation of x is $\neg x$ and the negation of $\neg x$ is x. □

Definition A *tautologous clause* is a clause containing some literal and its negation e.g. $x \vee y \vee \neg y$. Tautologous clauses evaluate to **true** under all interpretations. □

Definition An *empty clausal formula* is a clausal formula containing no clauses. If all clauses are removed from a clausal formula, it becomes an empty formula. Empty clausal formulas evaluate to **true** under all interpretations. □

Definition Suppose we have a non-empty clausal formula C free of tautologous clauses and a literal l where l or its negation appears in C. The *simplification of C assuming l* is a clausal formula C' equivalent to $C \wedge l$. C' is defined as follows: take C, remove all clauses which contain l and remove the negation of l from all clauses in which it appears. □

Definition A *partial truth assignment* to a set of variables V is a partial mapping of V to {**true**, **false**}. This means that if $\mathcal{T}$ is a partial truth assignment to V, for some variable $x \in V$, $\mathcal{T}(x)$ may be undefined. A *total* truth assignment on the other hand, must have truth values defined for all variables. □

Definition Suppose $\mathcal{T}_1$ and $\mathcal{T}_2$ are partial truth assignments to V. $\mathcal{T}_2$ *extends* (or is an *extension* of) $\mathcal{T}_1$ if for all $x \in V$ where $\mathcal{T}_1(x)$ is defined, $\mathcal{T}_2(x)$ is also defined and $\mathcal{T}_1(x) = \mathcal{T}_2(x)$. □

Definition Suppose we have a partial truth assignment $\mathcal{T}$ and a literal l such that the variable x mentioned in l is undefined in $\mathcal{T}$. The *extension of $\mathcal{T}$ by l* is the partial truth assignment $\mathcal{T}'$ which is equal to $\mathcal{T}$ except that if $l = x$, $\mathcal{T}'(x) =$ **true**; if $l = \neg x$, $\mathcal{T}'(x) =$ **false**. □

Definition A partial truth assignment $\mathcal{T}_{cyl}$ to the variables of a clausal formula C is a *solution cylinder* of C iff all total truth assignments which extend $\mathcal{T}_{cyl}$ are solutions to C. □

2.5.2 DPLL and its variants

We now review the DPLL procedure and some variants which can be used to calculate instance properties. We also note briefly the implementations of these which were used. All variants take a clausal formula as input and are recursive. In our presentation we will assume that the input formula contains no tautologous clauses. These can be removed as a preprocessing step and if so, do not re-occur in any clausal formula on which the procedure is called recursively. The procedures are also under-specified in the sense that the sub-procedure SELECT-SPLIT-LITERAL-*(C) can be instantiated in different ways: we discuss how this is done in our implementation towards the end of this section.

DPLL

The most basic procedure DPLL is shown in Figure 2.11. It solves the SAT decision problem by returning "satisfiable" or "unsatisfiable". If the formula is empty, it must be satisfiable; if it contains an empty clause, it cannot be. If the formula contains a unit clause, any solution to the formula must make the literal in that clause **true**. Hence we can assume that literal and call DPLL recursively on the simplified C. This is the *unit propagation* step. Similarly, if a literal l appears in the formula C, but its negation does not, then if there is any solution to C, there must be a solution which makes l **true**. Again in this case we can assume l is **true** and call DPLL recursively on the simplified C. This is the *pure deletion* step. Finally, if none of the above cases hold, we resort to searching. We select a literal l, assume it is **true** and attempt to show satisfiability under this assumption. If this fails, we attempt to show satisfiability under the opposite assumption. If both branches render unsatisfiability, the formula as a whole must be unsatisfiable.

DPLL-SOLUTION

This variant DPLL-SOLUTION returns a solution cylinder to the input formula if there is one and "unsatisfiable" otherwise. The solution cylinder can be made into a solution by assigning arbitrary truth values to its undefined variables. DPLL-SOLUTION

```
procedure DPLL(C)
    if C is empty
        return "satisfiable"
    end if
    if C contains an empty clause
        return "unsatisfiable"
    end if
    ;; Unit propagation
    if C contains a unit clause l
        return DPLL(the simplification of C assuming l)
    end if
    ;; Pure deletion
    if C mentions a literal l but not the negation of l
        return DPLL(the simplification of C assuming l)
    end if
    ;; Split
    l := SELECT-SPLIT-LITERAL-*(C)
    if DPLL(the simplification of C assuming l) returns "satisfiable"
        return "satisfiable"
    else
        return DPLL(the simplification of C assuming the negation of l)
    end if
```

Figure 2.11: The DPLL procedure for solving the SAT decision problem.

is shown in Figure 2.12. To allow a simpler presentation DPLL-SOLUTION has two arguments, the formula and a partial truth assignment. The set of defined values of this second argument represents the set of literals assumed to be true. At the top level DPLL-SOLUTION is called with the second argument set to the partial truth assignment which is undefined on all the variables of C. DPLL-SOLUTION is identical to DPLL except that each time a literal l is assumed to be true, the partial truth assignment is extended by l. If this sequence of assumptions leads to C being simplified to an empty formula, the partial assignment must be a solution cylinder.

```
procedure DPLL-SOLUTION(C, T)
    if C is empty
        return T
    end if
    if C contains an empty clause
        return "unsatisfiable"
    end if
    ;; Unit propagation
    if C contains a unit clause l
        T' := T extended by l
        return DPLL-SOLUTION(the simplification of C assuming l, T')
    end if
    ;; Pure deletion
    if C mentions a literal l but not the negation of l
        T' := T extended by l
        return DPLL-SOLUTION(the simplification of C assuming l, T')
    end if
    ;; Split
    l := SELECT-SPLIT-LITERAL-*(C)
    T' := T extended by l
    if DPLL-SOLUTION(the simplification of C assuming l, T') returns a
                    partial truth assignment T_sol
        return T_sol
    else
        T' := T extended by the negation of l
        return DPLL-SOLUTION(the simplification of C assuming the negation of l, T')
    end if
```

Figure 2.12: The DPLL-SOLUTION procedure for returning a solution to a SAT instance if one exists.

DPLL-TRAVERSE-ALL-SOLUTIONS

In some cases were are interested in properties of *all* solutions of a formula. DPLL-TRAVERSE-ALL-SOLUTIONS is a variant which "traverses" (i.e. visits) all solutions of the input formula C. That is, for every solution $\mathcal{T}_{sol}$ of C, DPLL-TRAVERSE-ALL-SOLUTIONS calls a sub-procedure TRAVERSE-*(C, $\mathcal{T}_{cyl}$) on some solution cylinder $\mathcal{T}_{cyl}$ which $\mathcal{T}_{sol}$ extends. How TRAVERSE-* is used (i.e. what property of all solutions we are interested in) may vary. When DPLL-TRAVERSE-ALL-SOLUTIONS traverses a solution cylinder it traverses all solutions which are extensions of that cylinder. DPLL-TRAVERSE-ALL-SOLUTIONS traverses each solution to C exactly once.

DPLL-TRAVERSE-ALL-SOLUTIONS is given in Figure 2.13. Again there are two arguments, the second initialised as in DPLL-SOLUTION. It is similar to DPLL-SOLUTION except for certain features. Firstly, no data is returned – the procedure is run simply for the side-effects of TRAVERSE-*, whatever these may be. Secondly, the pure deletion step is not used. Suppose a literal l is a literal which appears in C and that the negation of l does not. It does not follow that l must be **true** in all solution cylinders which extend the current partial assignment: there may be other solution cylinders where l is **false** i.e. where clauses containing l are made **true** by assigning other variables. If we use the pure deletion step such solution cylinders would not be traversed. Thirdly, in the split step we make a recursive call assuming l followed by a recursive call assuming the negation of l, again to ensure that all solution cylinders are traversed.

Implementation details

When DPLL and DPLL-SOLUTION were required, we interfaced our system with a highly optimised publicly available implementation NTAB (Crawford and Auton 1996). We also used the MODOC system (van Gelder and Okushi 1999), kindly made available by Allen van Gelder, for these purposes.

We implemented the DPLL-TRAVERSE-ALL-SOLUTIONS variant in C using the non-trie-based data structure suggested by Zhang & Stickel in Section 4 of their report (Zhang and Stickel 1994). In our implementation we used the Jeroslow-Wang heuristic for SELECT-SPLIT-LITERAL-* (Jeroslow and Wang 1990; Hooker and Vinay 1995). This

```
procedure DPLL-TRAVERSE-ALL-SOLUTIONS(C, T)
    if C is empty
        TRAVERSE-*(C, T)
        return
    end if
    if C contains an empty clause
        return
    end if
    ;; Unit propagation
    if C contains a unit clause l
        T' := T extended by l
        DPLL-TRAVERSE-ALL-SOLUTIONS(the simplification of C assuming l, T')
    end if
    ;; Split
    l := SELECT-SPLIT-LITERAL-*(C)
    T' := T extended by l
    DPLL-TRAVERSE-ALL-SOLUTIONS(the simplification of C assuming l, T')
    T' := T extended by the negation of l
    DPLL-TRAVERSE-ALL-SOLUTIONS(the simplification of C assuming the
                                    negation of l, T')
    return
```

Figure 2.13: The DPLL-TRAVERSE-ALL-SOLUTIONS procedure for traversing all solutions of a SAT instance.

selects the literal from all those appearing in the formula based on the number and lengths of the clauses in which it appears. Literals which appear in many short clauses are preferred the most.

2.6 Summary

We have reviewed the satisfiability (SAT) problem and the phenomenon of hard random SAT instances which is the object of study in this thesis. Next we detailed the local search procedures whose performance and behaviour on random SAT we intend to explain. We also surveyed some previous research which has contributed to the understanding of local search behaviour on random SAT. Finally, we reviewed the DPLL procedure which is used in the following chapters to calculate certain properties of random SAT instances which are related to the behaviour of local search.

Chapter 3

Local Search on RANDOM-3-SAT

3.1 Introduction

In this and the following chapter we are concerned with explaining the behaviour of local search on satisfiable instances in the threshold region of the RANDOM-3-SAT generation method. This chapter "poses the question" by elaborating the behaviour to be explained, while Chapter 4 attempts to answer it, by proposing and testing a causal hypothesis.

In our experiments we use just one local search algorithm: WSAT/SKC. One objection to studying a single algorithm from the local search class is that it may not be representative: results obtained for the algorithm may not generalise to other members of the class. While we accept this objection, there is evidence that under certain conditions, one local search algorithm is actually to a large extent representative of a wider class. For example Hoos (1998) found a very high correlation between the computational costs of solving random instances using pairs of different local search algorithms, including WSAT/SKC. That is, the computational cost of one random SAT instance for a particular local search algorithm is a good predictor of the computational cost for another algorithm. This also suggests that there is some algorithm-independent property of these instances which results in high cost for this class of algorithm.

Ultimately, our goal is to explain how instance properties affect search cost. As will be seen in this chapter, certain well-known properties can account for much of the cost behaviour. However, we will also find that some elements of the cost behaviour cannot

be explained in terms of these existing properties. Ideally we would like to extend our featural theory relating instance properties to cost so as to account for this unexplained behaviour.

To lay the groundwork for extending our theory, we study the *search behaviour* of WSAT/SKC i.e. the nature of the assignments visited during search. We will find that patterns in the search behaviour suggest that certain properties of the algorithm's search space are responsible for the cost phenomena. Since the search space is induced by the instance structure, in Chapter 4 we formulate a structural instance property which we think induces the search space properties responsible for the unexplained cost phenomena.

In this chapter there are two aims:

1. To obtain a finer description of the behaviour of WSAT/SKC cost in the threshold region, incorporating the roles of backbone size and the number of solutions.
2. To suggest features of the search space underlying the unexplained features of WSAT/SKC cost behaviour by studying the search behaviour of WSAT/SKC in the threshold region.

This chapter uses a number of computational and experimental methods with which the reader may not be familiar: these are explained in Section 3.2. Aim 1 is covered by Section 3.3 and Aim 2 in Section 3.4. Finally, Section 3.5 summarises the results of this chapter and Section 3.6 puts forward our speculative interpretation of these results which will motivate Chapter 4. Material from this chapter was used in an article in the *Journal of Artificial Intelligence Research* (Singer *et al.* 2000a).

3.2 Experimental methods

In this section we cover various experimental methods which are important to this and the following chapter. These are concerned with calculating various properties of clausal formulas.

3.2.1 Determining the number of solutions

If we allow for n variables, there are 2^n possible truth assignments. For some formula C it may be that all, some or none of these assignments are solutions (corresponding respectively to the validity, satisfiability and unsatisfiability of C). The number of assignments which are solutions is a property of SAT instances which has been quite well studied in relation to the threshold phenomenon and it is one of the properties of interest in this chapter.

We have implemented a variant of DPLL-TRAVERSE-ALL-SOLUTIONS which calculates the number of solutions to a clausal formula. Recall that TRAVERSE-* is called repeatedly with different solution cylinders such that each solution is an extension of exactly one of these cylinders. If a solution cylinder $\mathcal{T}_{cyl}$ is undefined on n_u of the variables of V, then 2^{n_u} solutions are extensions of $\mathcal{T}_{cyl}$. Our procedure simply sums the number of solutions which extend each cylinder: the total is the number of solutions of the formula C.

3.2.2 Determining backbone size

Recall from Chapter 2 that the backbone of C is the set of literals which are implicates of C. We determine backbone size by establishing which of all the possible literals are implicates. The procedure is based on three properties of implicate literals:

1. l is an implicate literal of C iff the simplification of C assuming the negation of l is unsatisfiable.
2. if l is **false** under some solution of C then l is not an implicate literal of C.
3. if l is an implicate literal of C then C is equivalent to the simplification of C assuming l.

These properties suggest the procedure DETERMINE-BACKBONE given in Figure 3.1. Note that under our definition of the backbone (although not that of Monasson *et al.*), for unsatisfiable instances, the backbone consists of all possible literals, since any literal is an implicate of a false formula.

In DETERMINE-BACKBONE the variable *Possible_backbone* represents the set of literals whose membership of the backbone has not been ruled out. Initially this contains all possible literals. The first step of the algorithm is to try to find a solution cylinder $\mathcal{T}_{cyl}$ for C. If this is successful, we can immediately rule out at least half the members of *Possible_backbone* by examining whether each of them is **false** under some extension of $\mathcal{T}_{cyl}$ (property 2 above). If C is unsatisfiable, we can return immediately the set of all literals.

The second part of the algorithm tests each member of *Possible_backbone* for membership of the backbone of C. In this part, C_{temp} is an instance equivalent to C but simplified by assuming known backbone literals. For each literal l in *Possible_backbone*, we generate an instance C_{test} by simplifying C_{temp} assuming the negation of l. We then attempt to find a solution cylinder for C_{test}. If this is successful, we know that l cannot be in the backbone (property 1 above). Also, this solution cylinder must be a solution cylinder for C and so as in the first part we can use it to eliminate further members of *Possible_backbone*. If C_{test} is unsatisfiable, we know that l is a backbone literal and so we can simplify C_{temp} assuming l (property 3 above). This simplification of C_{temp} can make future DPLL-SOLUTION calls on it much less costly.

After this computation, the remaining members of *Possible_backbone* must constitute the backbone of C.

As an extension to this technique, immediately after *Possible_backbone* is initialised, we can make one or more calls to a local search algorithm on C. These local search runs can potentially find several radically different solutions to C. The solutions can then be used to eliminate many members of *Possible_backbone* again based on property 2. By eliminating possible members of the backbone, we can reduce the subsequent number of calls to DPLL-SOLUTION. We found that this was particularly useful in the underconstrained region where small-backbone instances dominate.

3.2.3 "Controlling" backbone size

As discussed in Section 2.4.4, Parkes (1997) found that the backbone size of satisfiable threshold RANDOM-3-SAT instances is correlated with the local search cost. In this

```
procedure DETERMINE-BACKBONE(C)
    V := the set of variables of C
    Possible_backbone := ∅
    for each variable x ∈ V
        Possible_backbone := Possible_backbone ∪ {x, ¬x}
    end for
    if DPLL-SOLUTION(C) returns a solution cylinder T_cyl
        for each literal l ∈ Possible_backbone
            if there is some extension T_sol of T_cyl such that l is false under T_sol
                Remove l from Possible_backbone
            end if
        end for
    else
        return Possible_backbone
    end if
    C_temp := C
    for each literal l ∈ Possible_backbone
        C_test := the simplification of C_temp assuming the negation of l
        if DPLL-SOLUTION(C_test) returns a solution cylinder T_cyl
                Remove l from Possible_backbone
            for each literal l' ∈ Possible_backbone
                if there is some extension T_sol of T_cyl such that l' is false under T_sol
                    Remove l' from Possible_backbone
                end if
            end for
        else
            C_temp := the simplification of C_temp assuming l
        end if
    end for
    return Possible_backbone
```

Figure 3.1: The DETERMINE-BACKBONE procedure.

and the following chapter, our aim will often be to measure the correlation of some other instance property (say factor F) with the cost. However, F itself may be correlated with backbone size. Hence to estimate the variance in cost accounted for by the variance in F rather than backbone size we need to eliminate the variance due to backbone size. This is done simply by generating samples of instances such that all the instances in each sample have the same backbone size. By using multiple samples and a different backbone size in each sample, we can cover a range of backbone sizes. We then measure the correlation between F and the cost within each sample. In this way we can gauge the strength of relationship between local search cost and F conditional on a certain value of backbone size. Importantly the correlation will not be due to F's correlation with backbone size. We generate samples with a certain backbone size n_{back} by generating each instance from satisfiable threshold RANDOM-3-SAT, determining its backbone size and discarding it unless its backbone is of size n_{back}. We term this generation process "controlling" backbone size at n_{back}.

3.2.4 Determining Hamming distance to nearest solution

In this section we introduce the concept of Hamming distance to nearest solution which is used later in this chapter to analyse search behaviour. We also give practical details of how this can be measured.

Definition Suppose $\mathcal{T}_1$ and $\mathcal{T}_2$ are two total truth assignments. The *Hamming distance* between $\mathcal{T}_1$ and $\mathcal{T}_2$, denoted $hd(\mathcal{T}_1, \mathcal{T}_2)$, is the number of variables which $\mathcal{T}_1$ and $\mathcal{T}_2$ assign differently. □

Definition Let C be a SAT instance and let $\mathcal{T}$ be a total truth assignment to the variables of C. The *Hamming distance to the nearest solution* is a function of $\mathcal{T}$ and C denoted $hdns(\mathcal{T}, C)$. This is defined as the minimal $hd(\mathcal{T}, \mathcal{T}_{sol})$ such that $\mathcal{T}_{sol})$ is a solution to C. □

We implemented a variant of DPLL-TRAVERSE-ALL-SOLUTIONS which, given an assignment $\mathcal{T}$ and a satisfiable instance C will calculate $hdns(\mathcal{T}, C)$. Recall that TRAVERSE-* is called repeatedly with different solution cylinders such that each solution is an extension of exactly one of these cylinders. This variant maintains an upper bound on

$hdns(\mathcal{T}, C)$, denoted *Max_hdns*. Initially, *Max_hdns* is set to n. Each time TRAVERSE-* is called with a solution cylinder $\mathcal{T}_{cyl}$, we calculate the Hamming distance between $\mathcal{T}$ and the nearest solution which extends $\mathcal{T}_{cyl}$. This may provide a new upper bound on $hdns(\mathcal{T}, C)$, so *Max_hdns* is updated accordingly. After TRAVERSE-* is called for the last time, *Max_hdns* equals $hdns(\mathcal{T}, C)$.

3.2.5 Details of instances analysed

We now describe the principal instances used in this and the following chapter. We used satisfiable threshold RANDOM-3-SAT with set n to 100, which although it is quite a small instance size, is not trivial yet allows instance properties which are costly to calculate to be obtained for large numbers of instances.

We are primarily interested in the threshold region of m/n, where the cost peak occurs: the region near the point at which 50% of the instances are satisfiable. We looked at the region of m/n where the probability of satisfiability is between 90% and 20%. For $n = 100$ this is quite a narrow region of m/n: from 4.03 to 4.49. At each value of m/n we generated 1000 instances, controlling the backbone size at different values between $0.1n$ and $0.9n$. These instances have been archived in the SATLIB repository run by Holger Hoos and Thomas Stützle (`www.informatik.tu-darmstadt.de/AI/SATLIB`).

3.2.6 Measurement of cost for WSAT/SKC

Following Hoos (1998), we ran the algorithm without a restart mechanism. As discussed in Chapter 2, Section 2.4.2, for algorithms such as WSAT/SKC which have a sufficient alternative strategy for escaping local optima, random restarts do not significantly improve the overall cost.

In this and the following chapter it will be important to obtain an accurate estimate of WSAT/SKC cost for each instance, so we can correlate instance cost with other instance properties. Following Hoos (1998), we use the median run length (*mrl*) of 1000 WSAT/SKC runs on an instance C as our descriptive statistic representing WSAT/SKC's search cost on C. In a sense this represents a "typical" run of WSAT/SKC on C. Due to the positively skewed, exponential run length distribution

(Hoos and Stützle 1998), the mean is more sensitive to the long tail and overestimates the length of a typical run.

We set *noise* to 0.55, which Hoos (Hoos 1998) found to be approximately optimal on Random 3-SAT threshold instances with $n = 100$.

3.3 Cost behaviour in the threshold region

In this section we first demonstrate the cost peak for WSAT/SKC in the threshold region (Section 3.3.1). We confirm the existence of the cost peak using statistical methods in Section 3.3.2. Previously, the cost peak in these instances has been demonstrated for other local search algorithms such as GSAT (Clark *et al.* 1996). We then present a more detailed account of the phenomenon, based on backbone size and the number of solutions. Previous work studied has already studied backbone size in relation to cost (Parkes 1997) and the number of solutions in relation to cost (Clark *et al.* 1996; Hoos and Stützle 1998). In this section, by using our method of "controlling" backbone size, we obtain a detailed picture of the cost behaviour in relation to both backbone size *and* the number of solutions simultaneously. This allows us to identify some new results: features of cost behaviour which are not accounted for by these two instance properties. Section 3.3.3 explores the cost patterns revealed by controlling backbone size. Section 3.3.4 incorporates the number of solutions into the account.

3.3.1 The cost peak for WSAT/SKC

In Figure 3.2 we show the peak in WSAT/SKC cost which was mentioned (although not explicitly demonstrated) by Parkes (1997). At each level of m/n, we generated 5000 satisfiable instances, with m/n ranging from 4.03 to 4.7. We measured *mrl* for each of these. Each line in the plot gives a different point in the distribution of *mrl*, e.g. the 90th percentile is the *mrl* of the 500th most costly instance for WSAT/SKC.

The peak is slightly above the 50% point (4.29) for the median but appears to shift down for higher percentiles. A similar pattern was noticed by Hogg & Williams (Hogg and Williams 1994) in local search cost on random graph colouring instances.

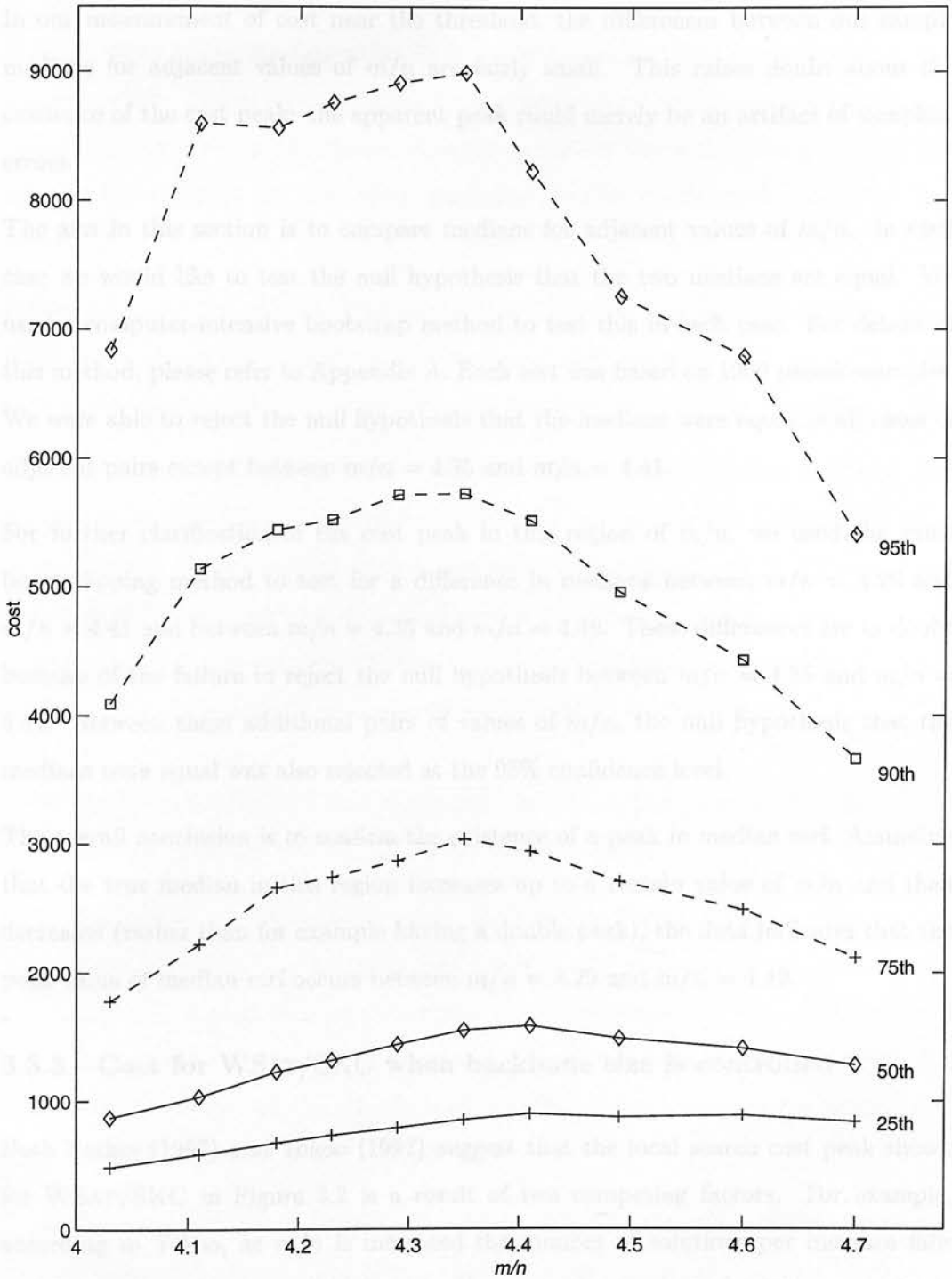

Figure 3.2: The cost peak for WSAT as m/n is varied.

3.3.2 Confirmation of the existence of the cost peak

In our measurement of cost near the threshold, the differences between our sample medians for adjacent values of m/n are fairly small. This raises doubt about the existence of the cost peak: the apparent peak could merely be an artifact of sampling errors.

The aim in this section is to compare medians for adjacent values of m/n. In each case we would like to test the null hypothesis that the two medians are equal. We used a computer-intensive bootstrap method to test this in each case. For details of this method, please refer to Appendix A. Each test was based on 1000 pseudo-samples. We were able to reject the null hypothesis that the medians were equal in all cases of adjacent pairs except between $m/n = 4.35$ and $m/n = 4.41$.

For further clarification of the cost peak in this region of m/n, we used the same bootstrapping method to test for a difference in medians between $m/n = 4.29$ and $m/n = 4.41$ and between $m/n = 4.35$ and $m/n = 4.49$. These differences are in doubt because of the failure to reject the null hypothesis between $m/n = 4.35$ and $m/n = 4.41$. Between these additional pairs of values of m/n, the null hypothesis that the medians were equal was also rejected at the 95% confidence level.

The overall conclusion is to confirm the existence of a peak in median *mrl*. Assuming that the true median in this region increases up to a certain value of m/n and then decreases (rather than for example having a double peak), the data indicates that the peak value of median *mrl* occurs between $m/n = 4.29$ and $m/n = 4.49$.

3.3.3 Cost for WSAT/SKC when backbone size is controlled

Both Parkes (1997) and Yokoo (1997) suggest that the local search cost peak shown for WSAT/SKC in Figure 3.2 is a result of two competing factors. For example, according to Yokoo, as m/n is increased the number of solutions per instance falls and this causes the onset of high cost. However, the number of solutions continues to fall in the overconstrained region but the cost decreases again. Yokoo concludes that there must therefore be a second, competing factor whose effect outweighs that of the number of solutions in the overconstrained region. The combined effect of the

two factors causes the peak in cost.

A pattern in WSAT/SKC cost on satisfiable threshold RANDOM-3-SAT identified by Parkes (1997) is our starting point as it is clear evidence for this competing factor interpretation. Parkes observed that for a given level of backbone size, the average cost for WSAT/SKC in the threshold region is highest below the 50% point and falls as m/n is increased. By controlling backbone size, we confirm this result and also extend it.

Figure 3.3 shows this pattern. Backbone size is controlled at $0.1n$, $0.5n$ and $0.9n$. Each point in the plot is the median *mrl* of a sample of 1000 instances and the length of the bars is the interquartile range (the difference between the 25th and 75th percentiles).

This experiment confirms the pattern found by Parkes. By controlling backbone size, we obtain large samples of instances of different backbone sizes at different values of m/n. There is a disadvantage to this approach: as mentioned, due to the scarcity of instances of certain backbone sizes at certain values of m/n, we had to limit n to 100 so as to produce large samples, whereas Parkes provided results for $n = 200$. However, the large sample size allows us to extend Parkes' result in several ways:

- The fall in median *mrl* is an approximately exponential decay for a range of m/n near the threshold and for a range of backbone sizes.
- The size of the interquartile range as shown in Figure 3.3 along with the log scale of the cost axis indicates that the distribution of per-instance cost is also positively skewed even once backbone size is controlled for. For example at the point where m/n is 4.11 and backbone size is $0.9n$ the difference between the 75th percentile and the median is about 4000 whereas between the median and the 25th percentile it is about half that.
- The spread of cost is large, particularly relative to the effect of the control parameter. We do not think that a significant portion of this variance in the cost among instances is due to sampling errors in our estimates of the cost for each instance, since each of these is based on a large sample of runs.

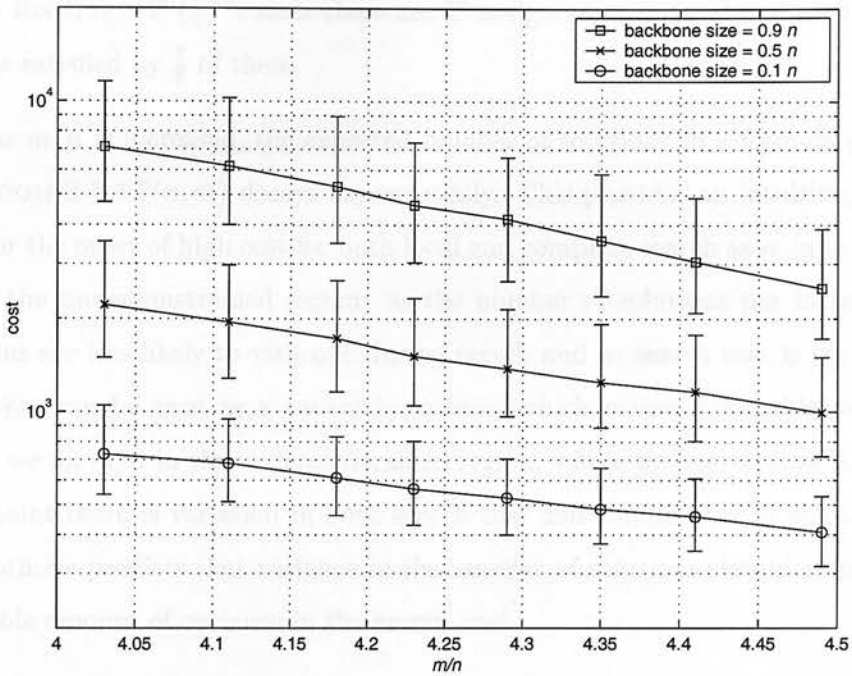

Figure 3.3: The effect of varying m/n on cost while backbone size is controlled. The length of the bars is the interquartile range (25th – 75th percentiles).

3.3.4 The effect of the number of solutions

In this section our aim is to integrate the number of solutions into our account of the cost peak. We first review previous work relating the number of solutions in RANDOM-3-SAT to local search cost. We then study the relationship between number of solutions and cost when backbone size is controlled. One key new result is that the number of solutions is tightly correlated with cost for small-backbone instances but less so for instances with larger backbones. Next we report on some evidence that for small-backbone instances, the number of solutions is *increasing* in the overconstrained region, which may (at least partially) account for the fall in cost in that region.

Previous work relating number of solutions to cost

In RANDOM-3-SAT (as in other simple stochastic generation methods of SAT and related problems) the expected number of solutions (averaged over both satisfiable and unsatisfiable instances) is straightforward to calculate. For RANDOM-3-SAT with

m and n fixed, it is $2^n(\frac{7}{8})^m$, since there are 2^n assignments in total and each of the m clauses is satisfied by $\frac{7}{8}$ of these.

Hence, as m/n is increased, the expected number of solutions to a formula generated by RANDOM-3-SAT(n, m) decays exponentially. This provides an intuitive, informal reason for the onset of high cost for both local and complete search as m/n is increased through the underconstrained region: as the number of solutions per instance falls, algorithms are less likely to visit one during search and so search cost is higher. This explanation can be seen as a causal hypothesis which makes a testable prediction. Suppose we fix m/n in the underconstrained region, where the search cost is rising. If at this point there is variation in both search cost and the number of solutions then the hypothesis predicts that variance in the number of solutions should account for a measurable amount of variance in the search cost.

Clark *et al.* (1996) showed that there is variance in the number of solutions in RANDOM-3-SAT and in the search cost (for the local search procedure GSAT) and that these two quantities are indeed correlated. In fact they found a fairly tight linear correlation between log of the number of solutions and log of the cost. Hence the prediction given the above hypothesis is supported by the experimental evidence.

Hoos (1998) also studied the relationship between these two quantities, using an improved methodology and a PAC variant algorithm GWSAT. An even tighter correlation was observed. One other observation by Hoos was that the correlation co-efficient between the number of solutions and the cost decreased as m/n was increased. That is, the number of solutions accounts for less of the variance in cost as we move into the overconstrained region.

Number of solutions and cost with backbone size controlled

We now study the relationship between the number of solutions and the WSAT/SKC cost with backbone size controlled at different values, hence integrating the study of the two instance properties. Figure 3.4 shows three log-log plots of the number of solutions against cost where m/n is 4.29 and backbone size is $0.1n$, $0.5n$ and $0.9n$. A linear least squares regression (*lsr*) fit is superimposed where appropriate. Table 3.1

m/n	Backbone size	Intercept of *lsr* fit (3 s. f.)	Gradient of *lsr* fit (3 s. f.)	r (3 s. f.)	Rank corr. (3 s. f.)
4.03	$0.1n$	3.90	−0.197	−0.781	-0.773
	$0.5n$	4.14	−0.212	−0.676	-0.670
	$0.9n$	4.21	−0.137	−0.131	-0.137
4.11	$0.1n$	3.87	−0.199	−0.770	-0.767
	$0.5n$	4.16	−0.230	−0.683	-0.686
	$0.9n$	4.14	−0.134	−0.128	-0.129
4.18	$0.1n$	3.79	−0.191	−0.766	-0.776
	$0.5n$	4.05	−0.218	−0.693	-0.697
	$0.9n$	4.02	−0.115	−0.116	-0.122
4.23	$0.1n$	3.78	−0.193	−0.783	-0.787
	$0.5n$	3.99	−0.214	−0.673	-0.687
	$0.9n$	3.99	−0.127	−0.132	-0.133
4.29	$0.1n$	3.73	−0.191	−0.779	-0.784
	$0.5n$	3.92	−0.208	−0.692	-0.694
	$0.9n$	3.78	−0.0610	−0.0612	-0.0534
4.35	$0.1n$	3.70	−0.190	−0.801	-0.799
	$0.5n$	3.89	−0.213	−0.687	-0.697
	$0.9n$	3.82	−0.102	−0.104	-0.0903
4.41	$0.1n$	3.61	−0.178	−0.778	-0.763
	$0.5n$	3.84	−0.209	−0.702	-0.709
	$0.9n$	3.78	−0.112	−0.118	-0.105
4.49	$0.1n$	3.55	−0.175	−0.797	-0.793
	$0.5n$	3.76	−0.204	−0.695	-0.699
	$0.9n$	3.62	−0.0842	−0.0992	-0.0783

Table 3.1: Data on log-log correlations between number of solutions and WSAT/SKC cost.

gives summary data on the log-log scatter plot for different backbone sizes through the transition : the gradient and intercept of *lsr* fits, the product-moment correlation r and the rank correlation.

The number of solutions is strongly and negatively correlated with the cost for smaller backbone sizes through the transition and the strength of the relationship is fairly constant as m/n is varied. We speculate that the strong relationship on these instances arises because finding an assignment which satisfies the backbone is straightforward and the main difficulty is encountering a solution once the backbone has been satisfied. The relative density of solutions in the region satisfying the backbone is then important.

For larger backbone sizes, the number of solutions is less relevant to the cost, indi-

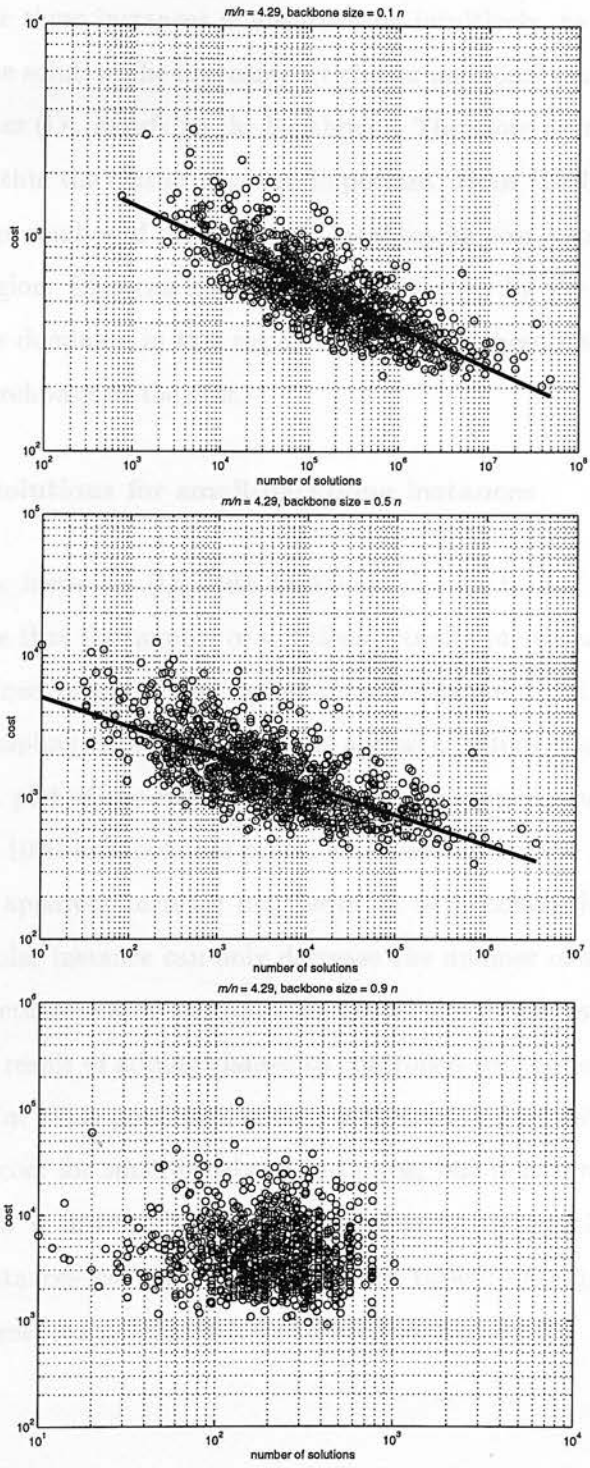

Figure 3.4: Scatter plot of the number of solutions against cost with backbone size controlled at $0.1n$ (top) $0.5n$ (middle) and $0.9n$ (bottom) ($m/n = 4.29$).

cated by the much lower r values. That the number of solutions and the cost are not strongly related for these instances is unsurprising intuitively, as the large backbone size implies that the solutions lie in a compact cluster and local search's main difficulty is finding this cluster (i.e. satisfying the backbone). Therefore we expect that the density of solutions within the cluster is not so important. Hoos (1998) observed that the correlation between number of solutions and local search cost becomes smaller in the overconstrained region. This can now be explained simply by the fact that the large-backbone instances dominate in this region and that for these instances, the number of solutions is less relevant to the cost.

The number of solutions for small-backbone instances

For small-backbone instances (i.e. with backbone size controlled at $0.1n$), we found some weak evidence that the number of solutions actually *increases* with m/n, at least in the overconstrained region. However we should also point out that this apparent effect could be a sampling error, given the large spread of values relative to the increase. Figure 3.5 shows a plot of the median number of solutions of small-backbone ($0.1n$) instances based on 1000 instances per point, with bars giving the interquartile range. At first sight this apparent increase may seem to be a contradiction, since adding clauses to a particular instance can only decrease the number of solutions. However, there is no contradiction because controlled-backbone size instances at one level of m/n are not simply the result of adding clauses to controlled-backbone size instances at a smaller level of m/n. This possible increase in the number of solutions may help to explain the fall in cost for small-backbone instances, but it is probably too weak an effect to account for it in full. No significant change in the number of solutions for larger-backbone instances was observed as m/n was varied: data for backbone sizes of $0.5n$ and $0.9n$ is presented in Figures 3.6 and 3.7 respectively.

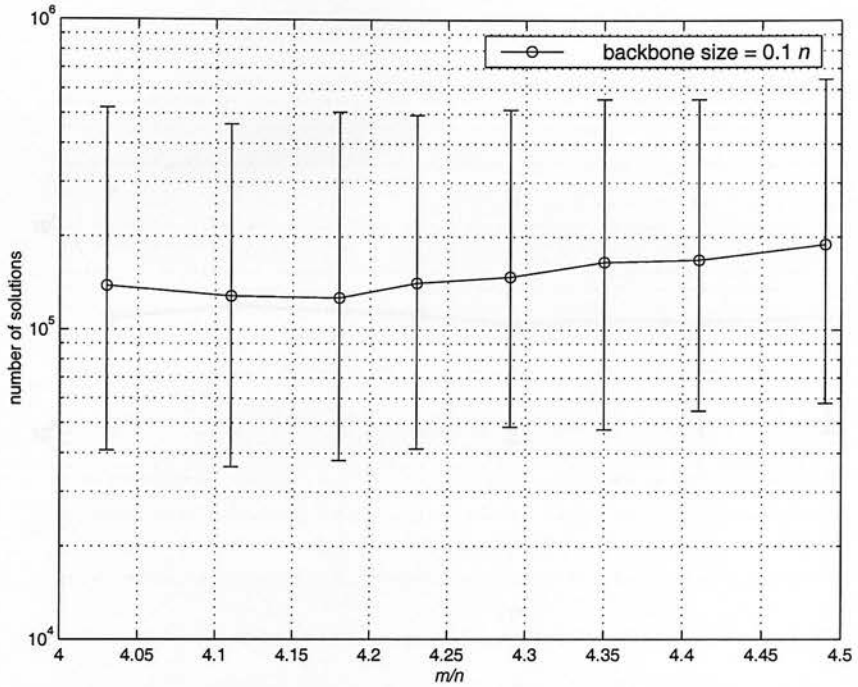

Figure 3.5: Number of solutions with m/n varied and backbone size controlled at $0.1n$. The length of the bars is the interquartile range (25th – 75th percentiles).

3.3.5 Summary

In this section we presented three sets of experimental results:

1. We demonstrated the peak in cost for WSAT/SKC near the satisfiability threshold.
2. We analysed cost behaviour when backbone size was controlled. This revealed two particularly interesting patterns:
   - An exponential decay in cost against m/n for a wide range of different backbone sizes.
   - A considerable amount of variation in cost for a given backbone size and value of m/n.
3. We analysed the relation between the number of solutions and the cost at different values of m/n when backbone size is controlled. This showed that the variance in cost for small-backbone instances could largely be accounted for by variance in

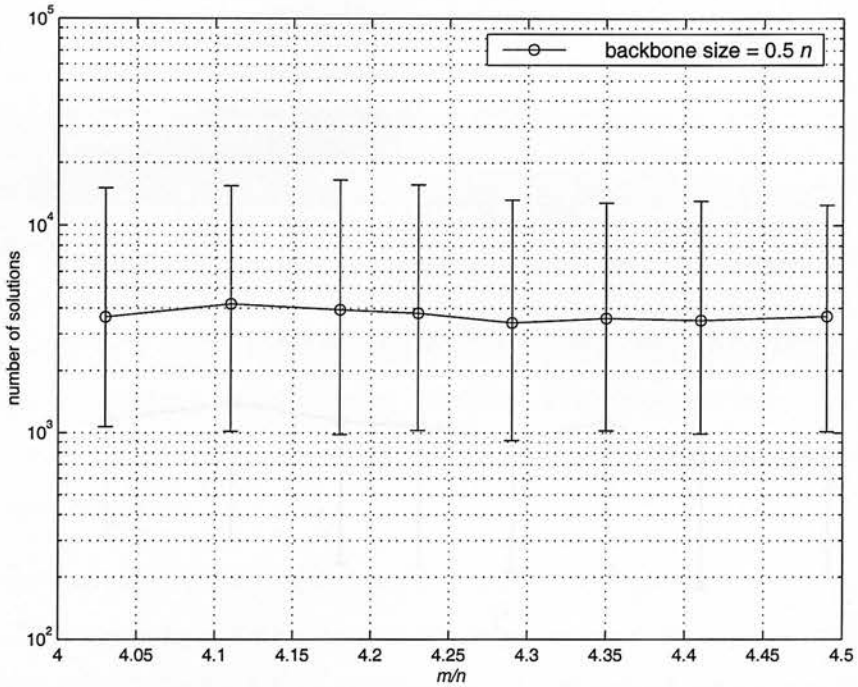

Figure 3.6: Number of solutions with m/n varied and backbone size controlled at $0.5n$. The length of the bars is the interquartile range (25th – 75th percentiles).

the number of solutions, whereas for large-backbone instances these two factors were largely unrelated.

3.4 Search behaviour in the threshold region

Our method of controlling backbone size along with studying the number of solutions has identified two elements of cost behaviour which lack a full explanation.

One aspect of the cost peak which remains to be explained is the decay in cost for instances when backbone size is controlled, which was observed in Section 3.3. Particularly interesting is the decay in cost for large-backbone instances. Since these dominate in the overconstrained region, understanding this decay will allow us to understand the overall decay in local search cost in the overconstrained region. Another aspect of cost behaviour which is currently unexplained is the large amount of variation in cost when m/n is fixed and backbone size is controlled at a large size e.g. $0.9n$. As we saw in Section 3.3.4 this variation is not accounted for by differences in the number

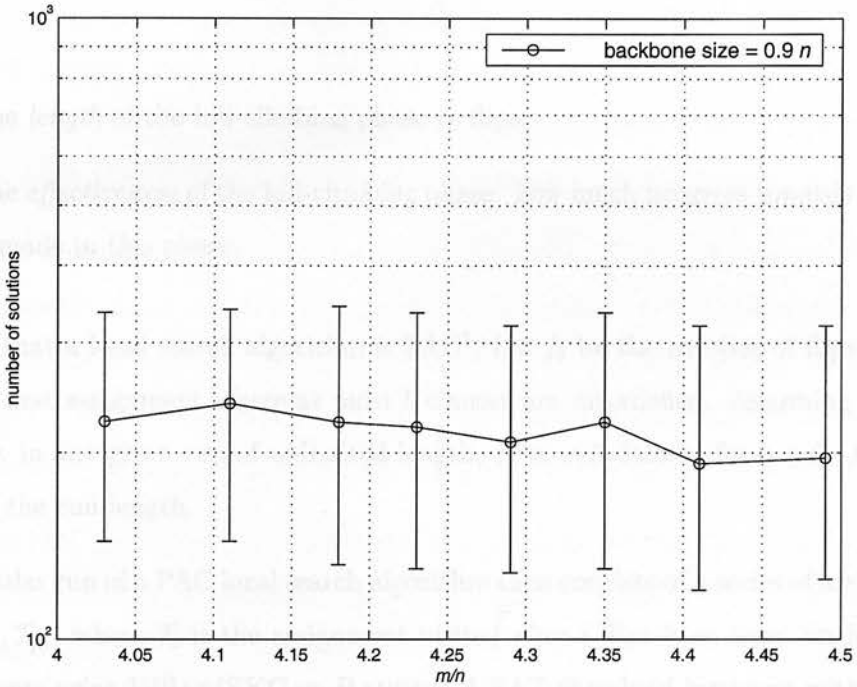

Figure 3.7: Number of solutions with m/n varied and backbone size controlled at $0.9n$. The length of the bars is the interquartile range (25th – 75th percentiles).

of solutions.

In order to identify the mechanisms underlying these two aspects of cost behaviour, we made a detailed study of WSAT/SKC's *search behaviour* which we report in this section. The search behaviour comprises the nature of the assignments visited during search.

Section 3.4.1 discusses the aspects of search behaviour which we will study. In Section 3.4.2 we give experimental results detailing how instance properties (m/n, backbone size) relate to the search behaviour and in 3.4.3 how the search behaviour relates to the cost.

3.4.1 Measurements of search behaviour: The length and effectiveness of the hill-climbing phase

Our study of the search behaviour concentrates on the point during a run when the behaviour changes from the hill-climbing phase to the plateau-like phase. In particular we are interested in two aspects of search behaviour which are introduced in this

section.

- The *length* of the hill-climbing phase in flips.
- The *effectiveness* of the hill-climbing phase: how much progress towards solutions is made in this phase.

Assume that a local search algorithm is PAC[1]. Let f_b be the number of flips taken to find the first assignment where at most b clauses are unsatisfied. Assuming the PAC property, in any given run of unlimited length, f_b is well-defined for $b \geq 0$. f_0 is then equal to the run length.

A particular run of a PAC local search algorithm then consists of a series of assignments $\mathcal{T}_0, \mathcal{T}_1, ..., \mathcal{T}_{f_0}$, where $\mathcal{T}_i$ is the assignment visited after i flips have been made. Initial experiments using WSAT/SKC on RANDOM-3-SAT threshold instances with $n = 100$ revealed that an assignment satisfying all but a few clauses is quickly found and that during the remainder of the search, few clauses (1 - 10) are unsatisfied. As discussed in Chapter 2, Section 2.4.2 there is first a hill-climbing phase, where the number of unsatisfied clauses drops rapidly to a very low level, followed by a long plateau-like phase in which the number of unsatisfied clauses is low but constantly changing. In our experiments we used f_5 as an arbitrary indicator of the length of the hill-climbing phase. Unlike for GSAT, for WSAT/SKC there is no well-defined end point for the hill-climbing phase, since short bursts of hill-climbing continue to occur for the rest of the search. We think that using f_b as the indicator with any value of b between 1 and 10 would give similar results.

We examined two aspects of the search behaviour. The first is simply f_5, the length of the hill-climbing phase. The second aspect is based on the Hamming distance between the current assignment and the nearest solution (i.e. $hdns(\mathcal{T},C)$ defined in Section 3.2.4). In particular, we were interested in the value of this quantity at the end of the hill-climbing phase i.e. $hdns(\mathcal{T}_{f_5}, C)$. This represents the ineffectiveness of the hill-climbing phase in Hamming-distance terms: how little progress was made towards a solution by hill-climbing.

[1] Recall from Section 2.3.8 that although it is unknown analytically whether WSAT/SKC is PAC, experiments suggest this may be so.

In the following two sections, data is based on RANDOM-3-SAT instances with $n = 100$ and backbone size controlled at various values between $0.1n$ and $0.9n$. As explained in Section 3.4.1 $hdns(\mathcal{T}_{f_5}, C)$ is the Hamming distance between the first assignment at which no more than 5 clauses are unsatisfied and the nearest solution.

We looked at summary statistics for f_5 and $hdns(\mathcal{T}_{f_5}, C)$ over a number of runs on each instance so as to ascertain the general type of search behaviour on the instance. For each instance we made 1000 runs of WSAT/SKC. The hill-climbing length (*hcl*) is defined as the median value for f_5 over the 1000 runs. The hill-climbing ineffectiveness (*hci*) is defined as the mean value for $hdns(\mathcal{T}_{f_5}, C)$ based on 1000 runs of WSAT/SKC.

We studied the typical search behaviour of instances by plotting the median of *hcl* and *hci* in each collection of instances. Hence, in Figure 3.8 each point is the median *hcl* of 1000 instances. In Figure 3.9 each point is the median *hci* of 1000 instances.

3.4.2 The relation between instance properties and search behaviour

This section relates the instance properties m/n and backbone size to the two search behaviour metrics *hcl* and *hci*.

Figure 3.8 shows the effect of varying m/n on *hcl* when the backbone size is controlled. The values for *hcl* (based on f_5) are low compared to those for the cost (f_0) in Figure 3.3. Also although the cost to find a solution varies considerably from one backbone size to another, a quasi-solution ($\mathcal{T}_{f_5}$) is found almost as quickly on large-backbone as on small-backbone instances. However, there are some notable aspects of the relationships between backbone size and m/n and *hcl*. As might be expected, on the larger backbone instances, for which overall cost is generally higher, WSAT/SKC takes slightly longer to find a quasi-solution. The relationship between m/n and *hcl* is unexpected. If backbone size is controlled at $0.5n$ or more, as m/n is increased WSAT/SKC takes slightly *longer* to find a quasi-solution, although simultaneously cost is decreasing as we have seen in Figure 3.3.

Figure 3.9 shows the relationship between varying m/n and *hci* when the backbone size is controlled. In this plot, the bars give the interquartile range. The spread of values for *hci* at each point is also small relative to the effect of varying m/n. Again the

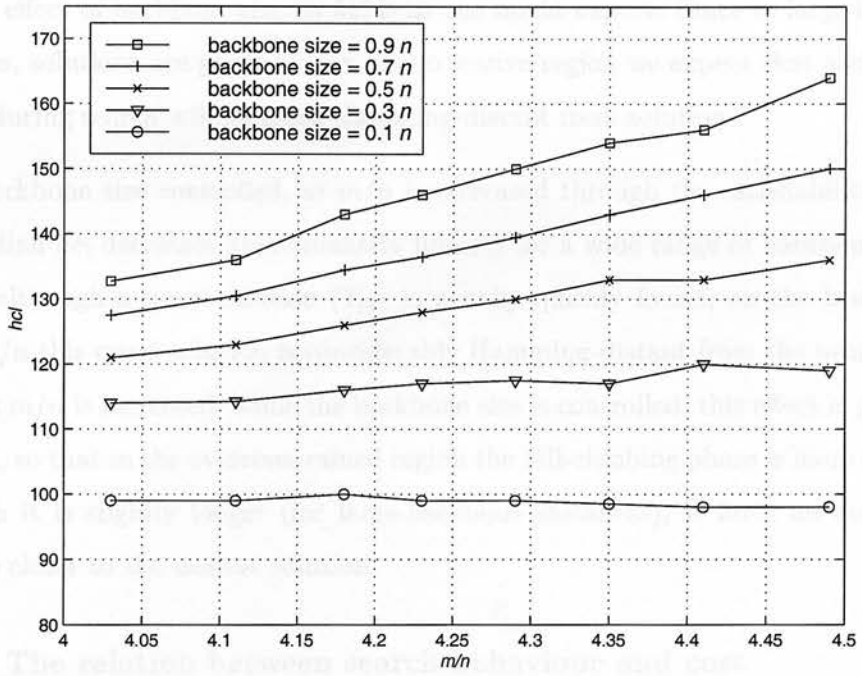

Figure 3.8: The relationship between m/n and hcl while backbone size is controlled.

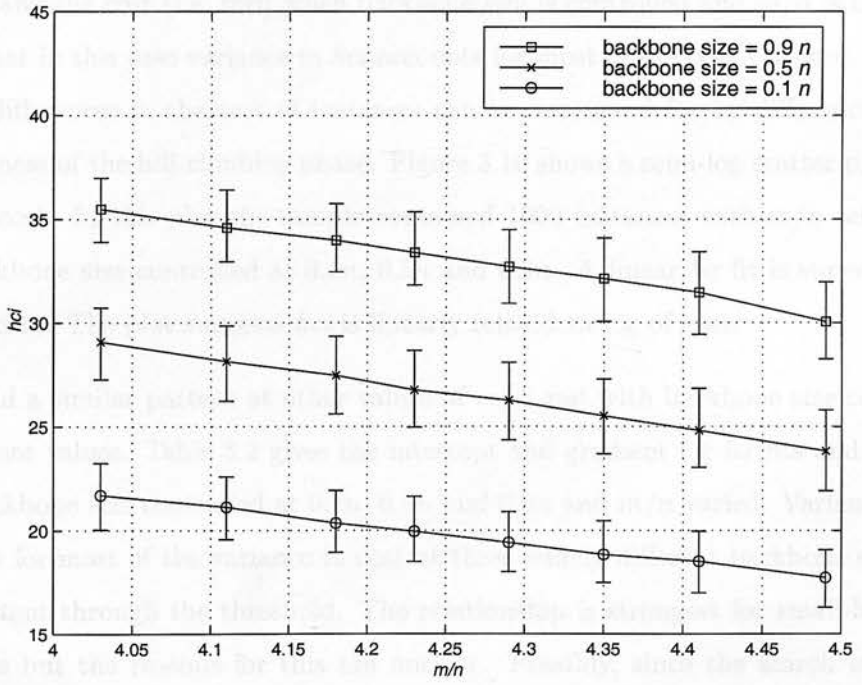

Figure 3.9: The relationship between m/n and hci when backbone size is controlled. The length of the bars is the interquartile range (25th – 75th percentiles).

positive effect of backbone size on *hci* is as one might expect. Since in large-backbone instances, solutions are grouped in a less extensive region we expect that assignments visited during search will be more Hamming-distant from solutions.

With backbone size controlled, as m/n is increased through the satisfiability threshold, median *hci* decreases approximately linearly for a wide range of backbone values. Hence, although a quasi-solution ($\mathcal{T}_{f_5}$) is usually quickly found, on the instances of lower m/n this quasi-solution is considerably Hamming-distant from the nearest solution. As m/n is increased, while the backbone size is controlled, this effect is gradually lessened, so that in the overconstrained region the hill-climbing phase is more effective: although it is slightly longer (for large-backbone instances), it finds an assignment which is closer to the nearest solution.

3.4.3 The relation between search behaviour and cost

In this section, we analyse the relationship between the search behaviour metrics *hcl* and *hci* and the cost (i.e. *mrl*) when backbone size is controlled and m/n is fixed. We found that in this case variance in *hci* accounts for most of the cost variance. In other words, differences in the cost of instances can be accounted for by differences in the effectiveness of the hill-climbing phase. Figure 3.10 shows a semi-log scatter plot of *hci* against cost. In this plot the sample contained 1000 instances with m/n set to 4.29 and backbone size controlled at $0.1n$, $0.5n$ and $0.9n$. A linear *lsr* fit is superimposed in each case. The plot suggests *hci* is linearly related to log of cost.

We found a similar pattern at other values of m/n and with backbone size controlled at different values. Table 3.2 gives the intercept and gradient for *lsr* fits and r values with backbone size controlled at $0.1n$, $0.5n$ and $0.9n$ and m/n varied. Variance in *hci* accounts for most of the variance in cost at three widely different backbone sizes and is consistent through the threshold. The relationship is strongest for small-backbone instances but the reasons for this are unclear. Possibly, since the search is shorter on the small-backbone instances, success follows more quickly after f_5 and so *hci* is a better indicator of the likelihood of finding a solution. In large-backbone instances, which have a much longer plateau-like phase, more can happen between the end of the hill-climbing phase and the end of the search. This is a possible reason for the weaker

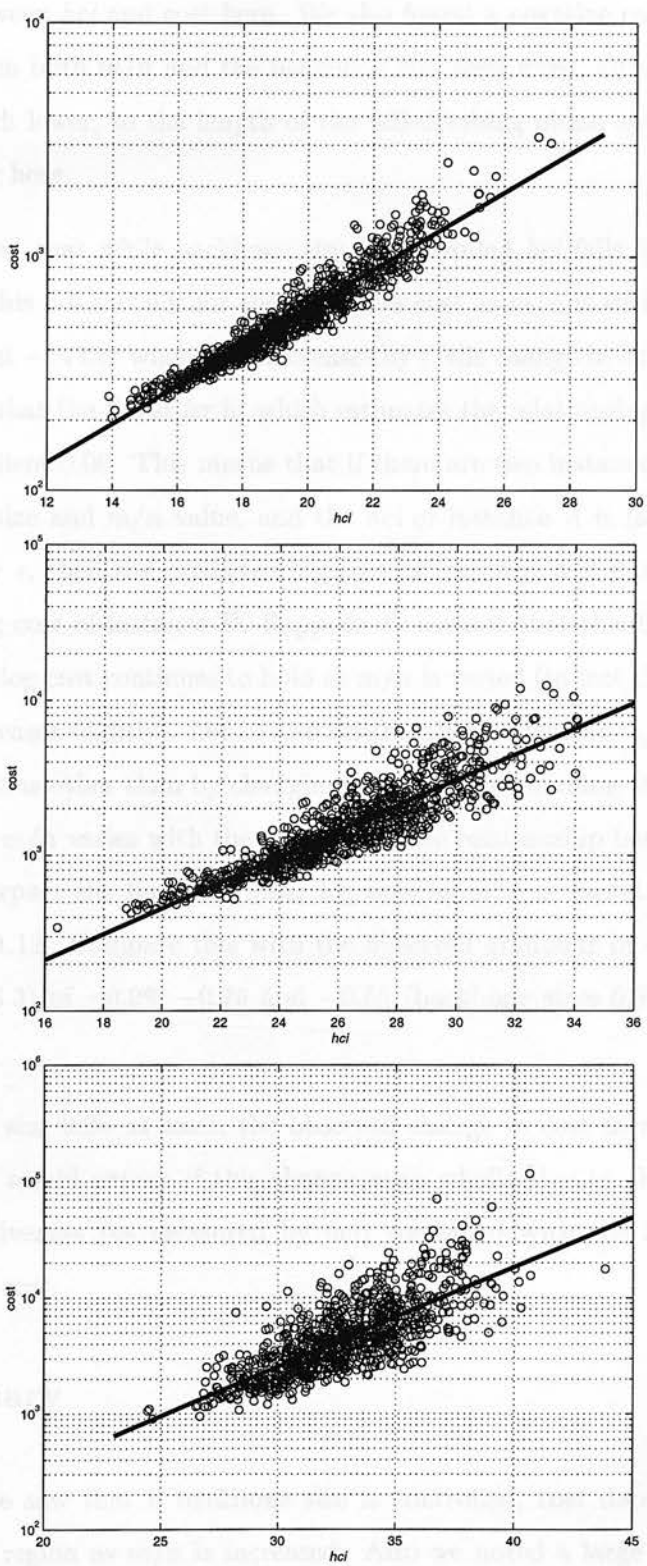

Figure 3.10: The correlation between *hci* and cost when backbone size is controlled at $0.1n$ (top), $0.5n$ (middle) and $0.9n$ (bottom) ($m/n = 4.29$).

relationship between *hci* and cost here. We also found a negative correlation between *hcl* and cost when both m/n and the backbone size were fixed, although the r values of this were much lower, so the length of the hill-climbing phase appears to be a less important factor here.

Figure 3.9 showed that while backbone size is controlled *hci* falls linearly as m/n is increased. Can this fall account for the change in cost as m/n is varied? The gradient of the fall is about -14 i.e. when m/n increases by ϵ, the change in *hci* is about $-14 \times \epsilon$. Table 3.2 shows that the linear *lsr* fit which estimates the relationship between *hci* and log cost has gradient 0.08. This means that if there are two instances A and B of the same backbone size and m/n value, and the *hci* of instance A is larger than the *hci* of instance B by ϵ, then the predicted log cost of instance A is $0.08 \times \epsilon$ larger than the predicted log cost of instance B. Suppose we assume that this linear relationship between *hci* and log cost continues to hold as m/n is varied (in fact the gradient of the relationship decreases slightly). Let us also assume that increasing m/n is not affecting the cost by a means other than by changing *hci*. Then, combining the gradient of the change in *hci* as m/n varies with the gradient of the relationship between *hci* and log cost we would expect the line describing log cost as m/n is varied to have gradient $-14 \times 0.08 = -1.12$. Compare this with the observed gradients of the change in log cost (in Figure 3.3) of -0.99, -0.76 and -0.55 (backbone sizes $0.9n$, $0.5n$ and $0.1n$ respectively).

So for backbone size $0.9n$ at least, the observed change in cost as m/n varies is not unlike what one would expect if this change were wholly due to the change in hill-climbing ineffectiveness (as measured by *hci*) combined with the linear correlation between *hci* and cost.

3.5 Summary

In Section 3.3 we saw that if backbone size is controlled, cost decays exponentially in the threshold region as m/n is increased. Also we noted a large variation in cost among instances with the same backbone size and m/n setting. These are the two cost phenomena of interest. We observed that differences in the number of solutions

m/n	Backbone size	Intercept of *lsr* fit (3 s. f.)	Gradient of *lsr* fit (3 s. f.)	r (3 s. f.)
4.03	$0.1n$	1.05	0.0844	0.945
	$0.5n$	0.693	0.0925	0.877
	$0.9n$	0.707	0.0895	0.730
4.11	$0.1n$	1.02	0.0868	0.951
	$0.5n$	0.632	0.0955	0.885
	$0.9n$	0.816	0.0867	0.720
4.18	$0.1n$	1.09	0.0839	0.956
	$0.5n$	0.809	0.0895	0.880
	$0.9n$	0.811	0.0864	0.720
4.23	$0.1n$	1.13	0.0821	0.958
	$0.5n$	0.834	0.0887	0.897
	$0.9n$	0.748	0.0878	0.769
4.29	$0.1n$	1.13	0.0826	0.955
	$0.5n$	1.00	0.0828	0.894
	$0.9n$	0.838	0.0856	0.758
4.35	$0.1n$	1.17	0.0811	0.963
	$0.5n$	0.984	0.0842	0.900
	$0.9n$	0.984	0.0808	0.773
4.41	$0.1n$	1.20	0.0795	0.957
	$0.5n$	1.03	0.0830	0.914
	$0.9n$	1.11	0.0768	0.782
4.49	$0.1n$	1.25	0.0777	0.967
	$0.5n$	1.15	0.0787	0.920
	$0.9n$	1.19	0.0742	0.809

Table 3.2: Data on correlations between *hci* and $\log_{10}$ *mrl* with backbone size controlled.

account for most of the variation when the backbone size is small, but little if any of the variation for large-backbone instances.

In Section 3.4 we looked at WSAT/SKC's search behaviour so as to suggest a more immediate cause of the two phenomena of cost behaviour. We found that when backbone size is controlled, for more underconstrained instances, the hill-climbing phase is short yet ineffective, in that it results in an assignment which is Hamming-distant from the nearest solution. As m/n is increased, the hill-climbing phase becomes more effective and (for large-backbone instances) longer. We also found that with backbone size controlled and m/n fixed, the Hamming distance to the nearest solution at the end of the hill-climbing phase is strongly and linearly correlated with log of cost, the correlation being particularly tight for small-backbone instances. The arithmetic of the relationships suggests that this aspect of search behaviour (the ineffectiveness of the hill-climbing phase) could account for both phenomena of cost behaviour.

3.6 Discussion

We interpret these search behaviour patterns by putting forward some speculative hypotheses about the *search space structure*: the way solutions and quasi-solutions are laid out in the space of assignments.

Our interpretation of the search behaviour pattern is illustrated by a conceptual diagram given in Figure 3.11. Recall that *quasi-solutions* are those total truth assignments at which a small number of clauses (in this case, between 1 and about 10) are unsatisfied. We propose that differences in search cost for WSAT/SKC between instances of the same large backbone size is dependent on the positions occupied by the quasi-solutions in the search space.

The boxes A and B are the sets of assignments of two instances. Within each box, Euclidean distance represents Hamming distance. This kind of diagram is unrealistic in many ways and is primarily useful as an illustration of intuitions. The darkly shaded regions are the solutions. Note that A and B have the same set of solutions and therefore the same backbone size. A and B are large-backbone instances, indicated by the fact that the solutions are concentrated in one small region (in reality this region

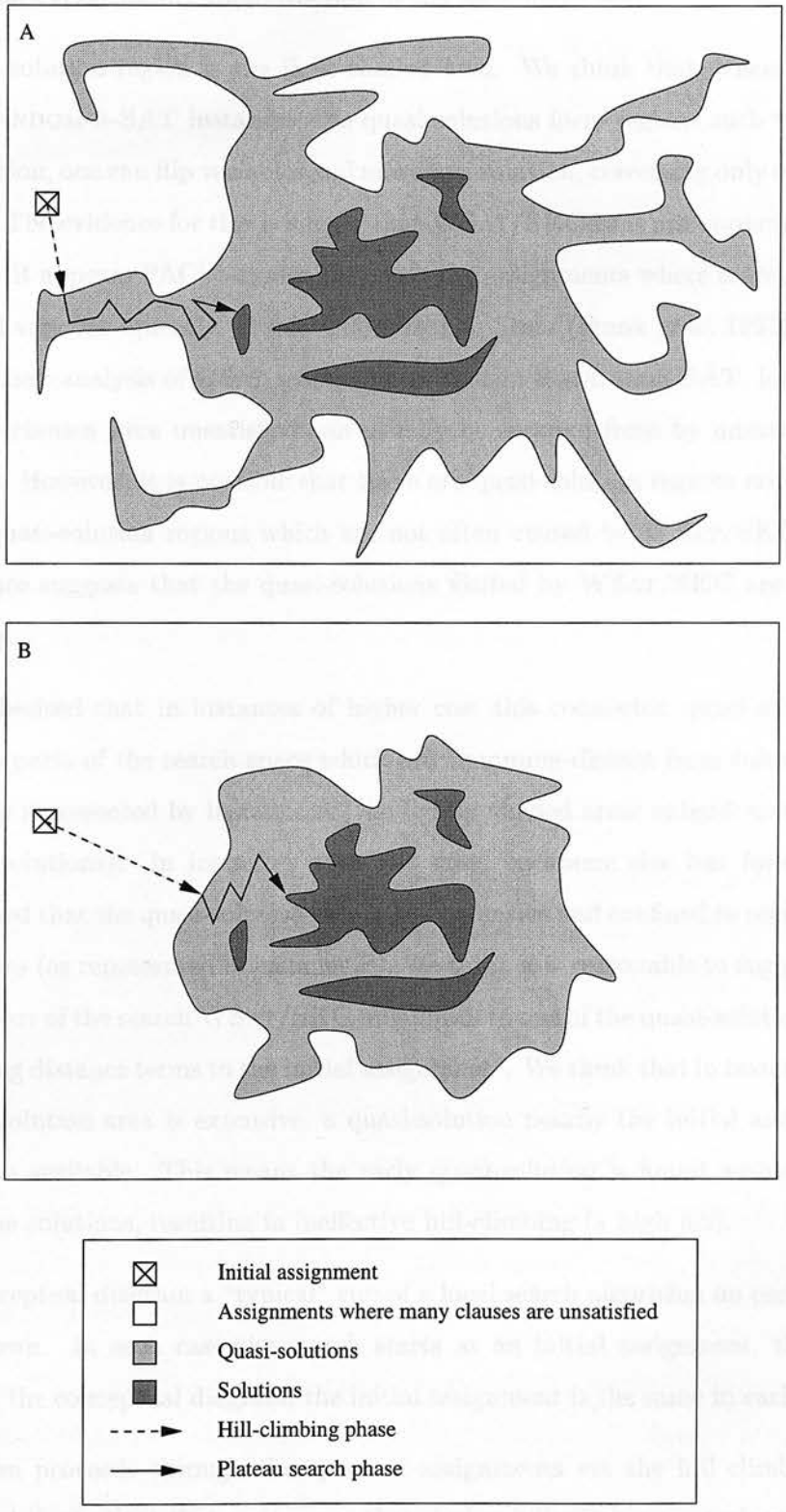

Figure 3.11: Conceptual interpretation of results from Chapter 3.

would be an exponentially small fraction of the assignment set).

The quasi-solution region is the light shaded area. We think that generally, in satisfiable RANDOM-3-SAT instances, the quasi-solutions form regions such that from a quasi-solution, one can flip variables and move to a solution, traversing only other quasi-solutions. The evidence for this is simply that WSAT/SKC runs are apparently always successful (it appears PAC) but also they visit the assignments where more clauses are unsatisfied very infrequently. Frank, Cheeseman & Stutz (Frank *et al.* 1997) also mentioned in their analysis of GSAT search spaces that in RANDOM-3-SAT, local minima where few clauses were unsatisfied can usually be escaped from by unsatisfying just one clause. However, it is possible that there are quasi-solution regions not connected to other quasi-solution regions which are not often visited by WSAT/SKC. At least the evidence suggests that the quasi-solutions visited by WSAT/SKC are connected in this way.

We hypothesised that in instances of higher cost this connected quasi-solution area extends to parts of the search space which are Hamming-distant from solutions. This situation is represented by instance A (the lightly shaded areas extend some distance from the solutions). In instances with the same backbone size but lower cost we hypothesised that the quasi-solution area is less extensive and confined to regions nearer the solutions (as represented in instance B). We think it is reasonable to suggest that in the early part of the search WSAT/SKC hill-climbs to one of the quasi-solutions nearest in Hamming distance terms to the initial assignment[2]. We think that in instances where the quasi-solution area is extensive, a quasi-solution nearby the initial assignment is likely to be available. This means the early quasi-solution is found without moving towards the solutions, resulting in ineffective hill-climbing (a high *hci*).

In the conceptual diagram a "typical" run of a local search algorithm on each instance is also shown. In each case the search starts at an initial assignment, the crossed square. In the conceptual diagram, the initial assignment is the same in each instance. Search then proceeds through the space of assignments via the hill climbing phase (the dashed line). Note that we assume that during hill-climbing, search moves fairly

[2] This assumption could be tested experimentally but this was not done as part of the thesis work.

directly towards a nearby quasi-solution area. Hence the location of the current assignment at the end of this phase depends on the location of the nearest quasi-solution. Once the quasi-solution region is entered the run is in the plateau phase (the solid line). Note that search is then confined to the quasi-solution area and is less directed because of the plateau-like topology. The run ends when a solution is found. Again the realism of these diagrams is limited: for example the length of the hill-climbing phase is usually a fraction of that of the plateau-like phase.

According to this interpretation, if the quasi-solution area is extensive as in instance A, a hill-climbing phase from a random point to a quasi-solution is short, but will be less effective in moving towards solutions, as is shown by the "typical" run on instance A. The subsequent search of the quasi-solution area will then be long possibly because of this initial large distance from the solutions or the extensiveness of the quasi-solution area in which the solutions lie, or a combination of these factors.

If the quasi-solution area is less extensive as in instance B, the hill-climbing phase will be longer, but the resulting quasi-solution will be closer to a solution and so the subsequent plateau search will be less costly.

As mentioned, we think that the extensiveness of the quasi-solution area affects the effectiveness of the hill-climbing phase. If m/n is varied and backbone size is controlled at a large value, hill-climbing becomes more effective. We think this is because the quasi-solution area becomes less extensive.

Why do we think that the quasi-solution area becomes less extensive? Consider a RANDOM-3-SAT instance with a large backbone. Suppose we add clauses to the instance at random, but only add each clause if it does not affect the backbone and does not render the instance unsatisfiable. This is approximately the situation when we increase m/n whilst controlling the backbone size.

Each of the 2^n assignments has a (possibly empty) set of unsatisfied clauses. The new clause is violated by 2^{n-3} assignments – all those which set its three literals false.

The new clause is more likely to contain backbone literals than non-backbone variables or negations of backbone literals. It cannot contain three negations of backbone literals as this would cause unsatisfiability. Neither can it contain two negations of backbone

literals and a non-backbone variable as this would increase the size of the backbone. On the other hand, if a new clause contains one or more backbone literals, it cannot possibly bring unsatisfiability or an increase in backbone size.

Given that the clause is likely to contain backbone literals, it is more likely to be violated by assignments which violate backbone literals than assignments which do not. Assignments which are Hamming-distant from the nearest solutions violate many backbone literals. These assigments are therefore more likely than assignments which are Hamming-near to the nearest solution to violate the new clause.

The more violated clauses a quasi-solution accumulates as clauses are added, the less it can be considered a quasi-solution. Hence those quasi-solutions which are Hamming-distant from the nearest solution are more likely to lose their quasi-solution status as clauses are added than those which are Hamming-near the nearest solution. This amounts to a decrease in the extensiveness of the quasi-solution area.

The question which remains is that of what factor determines the extensiveness of the quasi-solution area and hence the effectiveness of hill-climbing and in turn the overall cost for WSAT/SKC. In Chapter 4 we investigate this unknown factor.

Chapter 4

The Backbone Fragility Hypothesis

4.1 Introduction

In this chapter we propose that there is a structural property of instances which induces a search space structure which in turn causes the search behaviour and thus the unexplained cost pattern which was observed in Chapter 3. *Backbone fragility* is the structural property. We propose the following hypothesis, which is the core idea of the thesis:

> *Differences in cost amongst certain instances can be accounted for by differences in backbone fragility.*

In Section 4.2 we discuss the notion of backbone fragility and relate it to our interpretation of the search behaviour results from Section 3.6. We also discuss how backbone fragility may be measured and present some backbone fragility data on the instances used in Chapter 3.

A hypothesis is only of scientific merit if it makes correct predictions. Our hypothesis makes three correct predictions for which we provide experimental evidence. In Section 4.3 we show that the degree to which an instance is backbone-fragile accounts for some of the variance in cost when m/n is fixed and the backbone size is controlled. Following on from this, in Section 4.4 we use multiple regression to determine how much cost variance can be accounted for using combinations of instance properties. In Section

4.5 we generate instances which are altered so as to be more backbone-fragile. Our hypothesis correctly predicts that when RANDOM-3-SAT instances are altered in this way, the cost becomes considerably higher. In Section 4.6 we show that the hypothesis makes a correct prediction relating to the search behaviour. In Section 4.7 we relate this study to previous research. Finally, Section 4.8 summarises this chapter. Material from this chapter was used in an article in the *Journal of Artificial Intelligence Research* (Singer *et al.* 2000a).

4.2 What is backbone fragility?

In this section we define backbone fragility and motivate the definition. We also introduce a metric for calculating backbone fragility and apply this metric to the instances studied in Chapter 3.

Although backbone fragility can be measured in different ways, the following qualitative definition should give the general idea of what backbone fragility is.

Definition A SAT instance C with backbone size $bsize(C)$ is *backbone-fragile* iff the removal of a small proportion of the clauses of C at random on average results in an instance C' such that $bsize(C')$ is much smaller than $bsize(C)$. □

Conversely, a SAT instance is termed *backbone-robust* iff the removal of a small number clauses at random on average has little effect on the backbone size.

4.2.1 Motivation

Following on from the speculative discussion in Section 3.6 we now motivate our choice to study backbone fragility as a factor in the etiology of WSAT/SKC search cost. Recall from Chapter 3 the evidence that search cost was largely dependent on the effectiveness of the hill-climbing phase: how close it moved the current assignment towards solutions. The hill-climbing phase is ineffective when early on, the search visits quasi-solutions which are Hamming distant from solutions. In Section 3.6 we speculated that the attraction of these misleading quasi-solutions may be dependent on the extensiveness of the quasi-solution area. In this section we motivate backbone

fragility by explaining how it is related to the likelihood of the hill-climbing phase being ineffective and how it is related to the extensiveness of the quasi-solution area.

Suppose B is a small sub-bag of the clauses of a satisfiable SAT instance C. Let Q_B be the set of all assignments $\mathcal{T}_B$ such that UNSATISFIED-CLAUSES($\mathcal{T}_B$, C) is a non-empty subset of B. So Q_B is the set of all the non-solution quasi-solutions where at most the clauses B are unsatisfied. What structural property of C would cause the quasi-solutions Q_B to be attractive to WSAT/SKC? We already know that if the backbone of a RANDOM-3-SAT instance is small, its solutions are found with little search[1]. The solutions to $C - B$ ($C - B$ denotes C with one copy of each member of B removed) are either solutions to C or members of Q_B and not both. In the early part of the search at least, it is reasonable to assume that the assignments which are attractive to WSAT/SKC in the search space induced by the instance C are approximately the same assignments which are attractive in the search space induced by $C - B$, given that B is small. In this case, the members of Q_B (which are solutions of $C - B$ but not of C) will be attractive in the search space of C when the backbone of $C - B$ is small, particularly if C's backbone is large, because in this case the finding of a solution to $C - B$ will happen much earlier in a search on the search space of C than the finding of a solution to C.

Furthermore for any assignment $\mathcal{T}_B \in Q_B$, the number of variables which do not appear in the backbone of $C - B$ is an upper bound on $hdns(\mathcal{T}_B, C)$. This is because $\mathcal{T}_B$ is a solution to $C - B$ and therefore satisfies all the literals in the backbone of $C - B$. Since these literals are also in the backbone of C, $hdns(\mathcal{T}_B, C)$ can only be as large as the number of variables not in the backbone of $C - B$.

Therefore a large difference in the backbone size after removing the clauses B allows $hdns(\mathcal{T}_B, C)$ to be high. To summarise, if the removal of a certain small sub-bag of clauses causes the backbone size to be greatly reduced, we can expect that quasi-solutions where only these clauses are unsatisfied will be attractive to WSAT/SKC and possibly Hamming-distant from the nearest solution. If the removal of the clauses does not affect the backbone size much, the associated quasi-solutions must be near solutions (at least in large-backbone instances).

[1] A result due to Parkes (1997) – see Section 2.4.4.

We are interested in quasi-solutions in general rather than those in any particular Q_B. Suppose a large-backbone instance is backbone-fragile. In other words, removing a small set of clauses at random on average reduces the backbone considerably. By extension of the above argument we expect that in general quasi-solutions will be attractive and that they may extend to regions Hamming-distant from the nearest solution. If a large-backbone instance is backbone-robust on the other hand, quasi-solutions must be more concentrated in regions which are less Hamming-distant from solutions.

Recall from Section 3.5 that differences in the layout of the quasi-solution area may underlie the differences in cost between large-backbone instances of the same backbone size. We proposed that if the quasi-solution area extended to regions which were Hamming-distant from the solutions, the cost would be higher than if the quasi-solution area was restricted to regions near the solutions. Backbone fragility is related to this idea, since it (partly) corresponds to how far the quasi-solution area extends to regions which are Hamming-distant from solutions.

However the correspondence between backbone fragility and the extensiveness of the quasi-solution area is not exact because smallness of backbone does not exactly correspond to how widely-distributed (in terms of Hamming distance) the solutions are. An appropriate measure of how widely-distributed solutions are is the average Hamming distance between solutions (*ahdbs*). $ahdbs(C)$ for an instance C is defined as the average of $hd(\mathcal{T}_{sol1}, \mathcal{T}_{sol2})$ over all pairs $(\mathcal{T}_{sol1}, \mathcal{T}_{sol2})$ of distinct solutions to C. This would be one direct measure of how widely-distributed solutions are. However, no research to date has measured it, presumably because of the prohibitive cost. The question is then how backbone size is related to this factor.

On the one hand, the number of variables not appearing in the backbone of C is an upper bound on $ahdbs(C)$. On the other hand, in extreme cases the backbone size is not necessarily correlated with $ahdbs(C)$. For example, suppose in some SAT instance the solutions are precisely those assignments where at most one variable is set true[2]. Such an instance would have no backbone but the Hamming distance between any

[2] For example, a SAT instance with n variables $x_1, \ldots, x_n$ with a clause $\neg x_i \vee \neg x_j$ for every $i \neq j$, $i, j \in \{1, \ldots, n\}$.

pair of distinct solutions would be no more than 2. Hence, solutions to this instance would be narrowly-distributed in terms of Hamming distance. Similarly, backbone fragility does not correspond exactly to how widely-distributed the quasi-solutions are, compared to the solutions. Despite this lack of a guarantee of correspondence, since the counterexample is highly structured we conjecture that on RANDOM-3-SAT instances, backbone size does indeed approximately represent how widely-distributed solutions are. This conjecture could at some point be tested but this would again run into problems of feasibility due to the difficulty of measuring *ahdbs*.

A factor which corresponds more closely to how widely-distributed the quasi-solutions are compared to the solutions may be superior to backbone fragility in explaining differences in cost. The correct predictions shown in this chapter would then be a result of backbone fragility's correlation with this superior factor. However, we think that even if this proves to be the case the relative simplicity of backbone fragility and the relative tractability of measuring it will make it useful in classifying different kinds of SAT instance.

The idea that backbone fragility is an important factor in explaining the search behaviour pattern is appealing for other reasons. For each implicate literal[3] l of C, there must be a sub-bag of clauses in C whose conjunction implies l. For any given backbone size, as clauses are added, for any given implicate literal l we expect that the extra clauses allow alternative combinations of clauses which imply l. Hence after increasing m/n whilst controlling the backbone size, the random removal of clauses will have less effect on the backbone since the fact that a literal is implicate depends less on the presence of particular sub-bags. As clauses are added, we expect that instances will become less backbone-fragile i.e. more backbone-robust. Given the hypothetical relationship between backbone fragility and the search behaviour, this would then explain qualitatively why the search behaviour changes as it does when m/n is varied and backbone size is controlled.

[3] i.e. member of the backbone – see Section 2.4.4.

```
procedure ROBUSTNESS-TRIAL(C)
    Current_instance := C
    Clauses_removed := 0
    Backbone_size := backbone size of C
    Current_backbone_size := Backbone_size
    while Current_backbone_size > Backbone_size/2
        Current_instance := Current_instance with one clause removed at random
        Clauses_removed := Clauses_removed + 1
        Current_backbone_size := backbone size of Current_instance
    end while
    return Clauses_removed
```

Figure 4.1: The ROBUSTNESS-TRIAL procedure.

4.2.2 Measuring backbone robustness using robustness trials

We now define a measure of the backbone robustness of an instance (the opposite to backbone fragility) which will allow us to test predictions of the hypothesis. This measure of backbone robustness, which will be used throughout the rest of this chapter, is based on *robustness trials.* These are runs of a stochastic procedure shown in Figure 4.1 which takes a SAT instance and returns an integer.

One robustness trial proceeds as follows. We take the instance C and delete clauses at random, halting the process when the backbone size is reduced by at least half. At this point we return as the result the number of deleted clauses.

Definition The *trial-based backbone robustness* of an instance C (denoted *tbbr*) is the mean result of all possible runs of ROBUSTNESS-TRIAL(C), i.e. the average number of random deletions of clauses which must be made so as to reduce the backbone size by at least half. □

It is infeasible to compute the results of all possible robustness trials since there are so many different sequences of clauses which can be removed ($m!$). Therefore, when measuring *tbbr* we estimate it by computing the average of a random sample of robustness trials on C. We used at least 100 robustness trials on each instance and in order to ensure a reasonably accurate estimate, we continued to sample more robustness trials until the standard error was less than 0.05 $\times$ the sample mean. In this case the con-

fidence interval for our estimate of the population mean from the sample mean equals the sample mean +/− about 10% of the sample mean, under the assumption that the sample mean is normally distributed. These conditions were used to estimate *tbbr* in all experiments reported in this chapter.

With $n = 100$, using satisfiable instances from near the satisfiability threshold whose backbone size was controlled at 50, usually less than 250 robustness trials were required for the estimate to converge in this way. Even then an estimate of *tbbr* was costly to compute.

Observe that although *tbbr* is an appropriate measure of backbone robustness, there are different possible metrics for backbone fragility/robustness. We found that *tbbr* gave the clearest results for this chapter's purposes without an unnecessarily complicated definition. Other metrics, such as the reduction in backbone size when a random fixed fraction of clauses is removed, may be more suitable in other contexts.

Since *tbbr* is defined in terms of the size of the backbone, it is most useful when comparing instances of equal backbone size and may be less useful when comparing instances of different backbone sizes.

4.2.3 The change in backbone robustness as m/n is varied

As discussed in Section 4.2.1 we expect that if backbone size is controlled, backbone robustness increases as m/n is increased.

We found that increasing m/n while controlling backbone size increased the median estimated *tbbr* of instances, as expected. Figure 4.2 shows the effect on the median estimated *tbbr* of increasing m/n through the tisfiability threshold while $n = 100$ and backbone size is controlled. Each point is the median estimated *tbbr* of 1000 instances.

We note that median estimated *tbbr* is higher for instances with larger backbones. We think that this is because on the large-backbone instances, the backbone must be reduced by a larger number of literals in each robustness trial and that this requires more clauses to be removed. Another effect is that median estimated *tbbr* increases with m/n at a higher rate for large-backbone instances than for small backbone instances. The reasons for this are unclear.

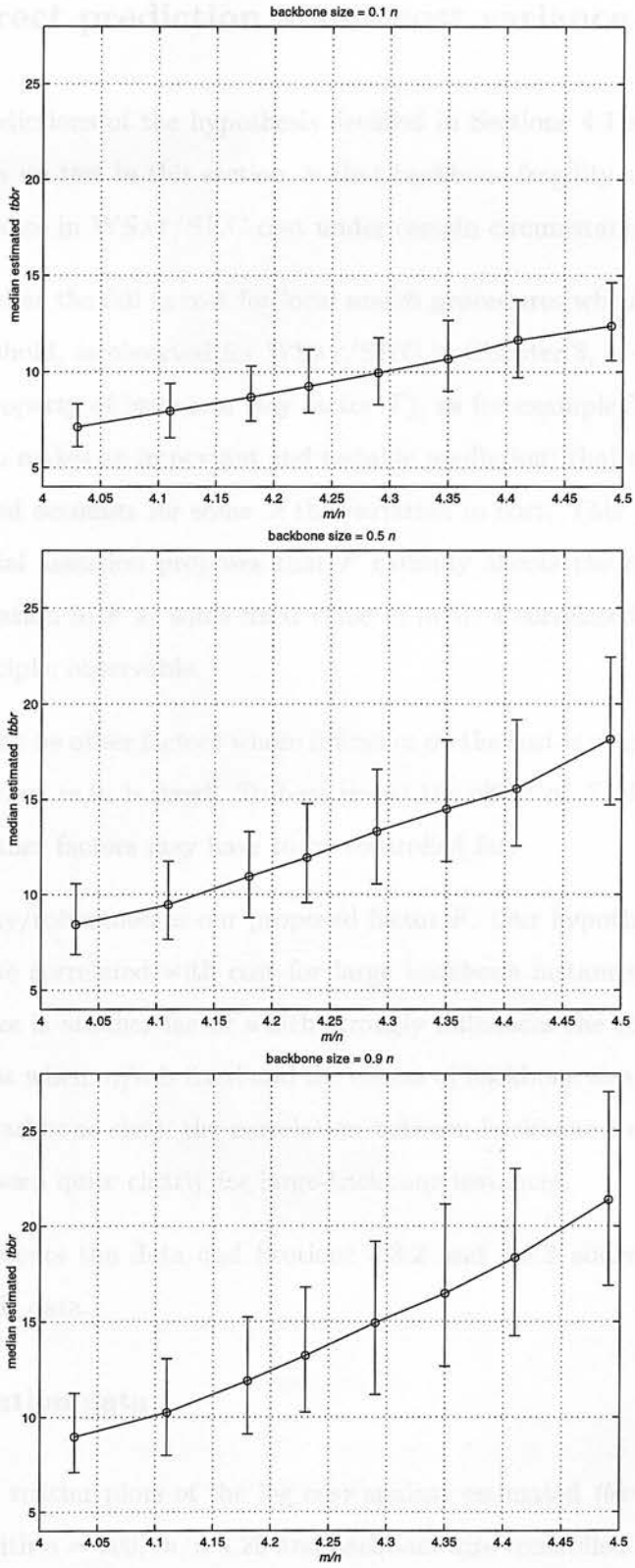

Figure 4.2: Median estimated *tbbr* through the satisfiability transition, with backbone size controlled at $0.1n$ (top) $0.5n$ (middle) and $0.9n$ (bottom). The bars show the interquartile range (25th – 75th percentiles) to give an indication of the spread of values.

4.3 A correct prediction about cost variance

We now test predictions of the hypothesis detailed in Sections 4.1 and 4.2. The first prediction, which we test in this section, is that backbone fragility should account for some of the variance in WSAT/SKC cost under certain circumstances.

One may assert that the fall in cost for local search procedures when m/n is increased beyond the threshold, as observed for WSAT/SKC in Chapter 3, is due to the change in some other property of instances (say factor F), as for example Yokoo (1997) has. Such an assertion makes an important and testable prediction: that any variation in F when m/n is fixed accounts for some of the variation in cost. This prediction follows because the initial assertion proposes that F causally affects the cost. Therefore if there is any variation in F at some fixed value of m/n, a correlated variation in cost must be, in principle, observable.

However there may be other factors whose influence on the cost is so great as to obscure the effect of F when m/n is fixed. To best reveal the effect of F, if there is any, the effects of some other factors may have to be controlled for.

Backbone fragility/robustness is our proposed factor F. Our hypothesis predicts that this factor will be correlated with cost for large backbone instances if m/n is fixed. The backbone size is another factor which strongly influences the cost. Our result in this section is that when m/n is fixed and the effects of backbone size are controlled for (by controlling backbone size), the correlation between backbone fragility/robustness and cost can be seen quite clearly for large-backbone instances.

Section 4.3.1 presents the data and Sections 4.3.2 and 4.3.3 address the statistical significance of this data.

4.3.1 Correlation data

Figure 4.3 shows scatter plots of the log cost against estimated *tbbr* for RANDOM-3-SAT instances with $n = 100$, m/n 4.29 and backbone size controlled at $0.1n$, $0.5n$ and $0.9n$. A linear *lsr* fit is superimposed where appropriate. Table 4.1 gives the intercept, gradient and r values for *lsr* fits to corresponding correlation data with backbone size

controlled at the same three values and with m/n varied through the threshold.

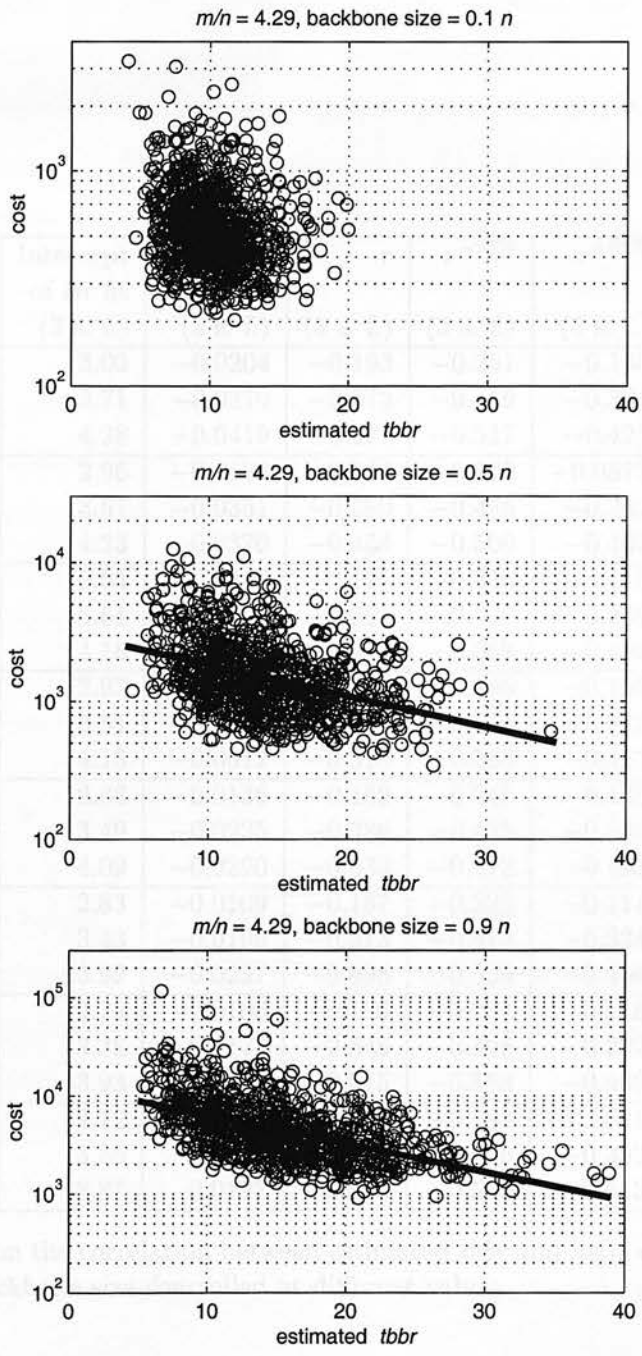

Figure 4.3: Scatter plot of estimated *tbbr* versus cost with $n = 100$, $m/n = 4.29$ and backbone size controlled at $0.1n$, $0.5n$ and $0.9n$.

The r values suggest an effect of *tbbr* on cost for large backbone instances. For smaller backbone sizes, we speculate that finding the backbone is less of an issue for local search

m/n	*bsize*	Intercept of *lsr* fit (3 s. f.)	Gradient of *lsr* fit (3 s. f.)	r (3 s. f.)	$r^{-95\%}$ (3 s. f.)	$r^{+95\%}$ (3 s. f.)	Rank corr. coefficient (3 s. f.)
4.03	$0.1n$	3.03	−0.0204	−0.193	−0.251	−0.140	−0.193
	$0.5n$	3.71	−0.0370	−0.373	−0.419	−0.324	−0.371
	$0.9n$	4.28	−0.0419	−0.471	−0.517	−0.425	−0.470
4.11	$0.1n$	2.96	−0.0134	−0.149	−0.209	−0.0873	−0.140
	$0.5n$	3.67	−0.0351	−0.390	−0.436	−0.342	−0.377
	$0.9n$	4.23	−0.0370	−0.454	−0.500	−0.407	−0.466
4.18	$0.1n$	2.94	−0.0146	−0.175	−0.236	−0.115	−0.166
	$0.5n$	3.61	−0.0302	−0.384	−0.427	−0.339	−0.374
	$0.9n$	4.18	−0.0338	−0.531	−0.569	−0.492	−0.547
4.23	$0.1n$	2.93	−0.0155	−0.211	−0.266	−0.155	−0.212
	$0.5n$	3.51	−0.0239	−0.364	−0.411	−0.315	−0.344
	$0.9n$	4.13	−0.0312	−0.525	−0.565	−0.482	−0.546
4.29	$0.1n$	2.88	−0.0136	−0.189	−0.248	−0.130	−0.206
	$0.5n$	3.49	−0.0225	−0.386	−0.435	−0.340	−0.400
	$0.9n$	4.09	−0.0290	−0.533	−0.572	−0.493	−0.547
4.35	$0.1n$	2.83	−0.0109	−0.167	−0.225	−0.111	−0.172
	$0.5n$	3.43	−0.0199	−0.373	−0.422	−0.324	−0.378
	$0.9n$	3.99	−0.0237	−0.498	−0.539	−0.456	−0.524
4.41	$0.1n$	2.79	−0.0100	−0.176	−0.232	−0.118	−0.168
	$0.5n$	3.38	−0.0172	−0.345	−0.395	−0.292	−0.358
	$0.9n$	3.93	−0.0211	−0.515	−0.558	−0.469	−0.527
4.49	$0.1n$	2.72	−0.0073	−0.139	−0.193	−0.0841	−0.136
	$0.5n$	3.35	−0.0170	−0.403	−0.459	−0.352	−0.403
	$0.9n$	3.87	−0.0198	−0.555	−0.595	−0.513	−0.560

Table 4.1: Data on the correlation between estimated *tbbr* and $\log_{10}$ cost with $n = 100$ and m/n and backbone size controlled at different values.

and so backbone fragility, which we think hinders the identification of the backbone, has less of an effect. For the larger backbone sizes, we think the main difficulty for WSAT/SKC is satisfying the backbone; so backbone fragility is then important. However, given the somewhat unclear shape of the scatter plots, there are several concerns about the significance of the observed correlation coefficients, which we now address using some simple statistical methods.

4.3.2 Rank correlation coefficient

The r coefficient, given above, can be greatly affected by outliers. Therefore the rank correlation coefficient, which is less affected, was also calculated. The rank correlation is also given in Table 4.1. We found that in each case the rank correlation coefficient is not considerably different from the r coefficient. This demonstrates that the observed r was not greatly affected by outliers.

4.3.3 Confidence intervals for the correlation

Given that there is a relationship between the two variables which is not merely an artifact of outliers, how accurate is our measurement of r? A bootstrap method can be used to obtain bounds on a confidence interval for this statistic. Again, the reader is referred to Appendix A for details of this method. Using this method with 1000 pseudo-samples we obtained lower and upper bounds on the 95% confidence interval for r, which are also given in Table 4.1 as $r^{-95\%}$ and $r^{+95\%}$ respectively. The data implies that with 95% confidence, each of our estimates of r is within about 0.05 of the true correlation coefficient.

4.4 Assessment of the predictability of WSAT/SKC cost

How much variance in cost can we now account for using algorithm-independent instance properties as predictors? When can knowledge of the instance properties make a significant difference to the predictability of cost? In this section we assess in more detail how predictable WSAT/SKC cost is based on the three instance properties: m/n, number of solutions and backbone robustness. First, in Section 4.4.1 we discuss

the methods which we used and other conventions we adopted. Then in Section 4.4.2 we present data for two scenarios: one where both backbone size and m/n are fixed at different combinations of values and another where backbone size is fixed, but m/n is treated as a predictor variable. In each case we comment on interesting aspects of the statistics.

4.4.1 Methods

In this section we use *multiple regression* to assess the predictability of one variable from several other variables. Multiple regression is a generalisation of the linear regression method used elsewhere in the thesis. Rather than simply finding a 2-dimensional linear prediction function which minimises the sum of squared residuals, multple regression finds a linear prediction function of higher dimension which minimises the sum of squared residuals. It can be used to make quantitative predictions of a variable taking several variables into account. Linear regression for a single predictor variable is then a special case. We follow the notational conventions and other methods of Cohen (1995) (Section 8.6 and appendices 8A.5 and 8A.6). This provides a means of assessing the performance and analysing the relative contribution of different predictors within the resulting prediction function.

Here we explain the instances used, how the dependent and predictor variable measurements were taken and the meaning of the statistics which are presented.

Instances

The instances used in this section are satisfiable, backbone-controlled RANDOM-3-SAT instances with $n = 100$ and backbone sizes $0.1n$, $0.5n$ and $0.9n$. The value of m/n which was used varies from 4.03 to 4.49. There were 1000 instances for each combination of backbone size and m/n.

Dependent variable

In this section the dependent variable (the one which is being predicted) is the log of *mrl* of WSAT/SKC with *noise* = 0.55. Recall that *mrl* is the median length of 1000

runs of the algorithm.

Predictor variables

The predictor variables are:

- The log of the number of solutions (labeled in tables as *lgsolns*). We use the log here because this transformation was found to give a good prediction in Section 3.3.4.
- The estimated *tbbr* (labeled in tables as *etbbr*). Estimated *tbbr* represents backbone robustness, and was calculated for this section using the method given in Section 4.2.2.
- In the last multiple regression scanario, m/n is also a predictor variable.

Statistics

For each predictor function which we derived, three kinds of statistic are recorded in the summary tables:

- R^2 is the fraction of variance in the dependent variable which is explained by the predictor function. For the single predictor variable case, R^2 equals r^2, the square of the correlation coefficient.
- For each predictor present in the predictor function, the β (or standardised) coefficient of that predictor is given. Suppose x is a predictor variable and y is the dependent variable. The β_x coefficient for a predictor x measures the change in the prediction of y when x is varied. If x is varied by s_x (one standard deviation of x) then the predicted y varies by $\beta_x \times s_y$ ($s_y =$ one standard deviation of y). The β coefficients of two predictors within a predictor function can be compared to assess the relative contribution of the predictors in accounting for y's variance. For the single predictor variable case, β equals the correlation coefficent r. See Cohen (1995) appendix 8A.2 for details of the definition of β-coefficients.

- For each predictor present in the predictor function, the F-statistic is calculated to assess the whether the additional variance in the dependent variable which is apparently accounted for by the predictor variable is actually significant. This is as opposed to the null hypothesis which states that the predictor variable accounts for no additional variance in the dependent variable. To test the null hypothesis, the obtained F-statistic must be compared with the appropriate F distribution. For details if how the F-statistic is calculated and used, see Cohen (1995). Although the F-test assumes a that the underlying population distributions are normal, this is less of a serious concern here as sample sizes are large; at least 1000 (see Cohen (1995) p. 206). In each case the number of denominator degrees of freedom which was used to calculate the F-statistic for each predictor was the sample size minus the number of predictors (see Cohen (1995) p. 328 and Section 8A.6).

4.4.2 Summary statistics for multiple regression prediction functions

In the first scenario we fix both backbone size and m/n at different combinations of values and derive a prediction function for log of *mrl* based on estimated *ttbr* and log of number of solutions. In the second scenario we fix backbone size and then the predictors are m/n, estimated *ttbr* and log of number of solutions.

Summary statistics for m/n and backbone size fixed

Tables 4.2, 4.3 and 4.4 give all summary statistics on the derived predictor functions for backbone size $= 0.1n$, $0.5n$ and $0.9n$ respectively. For each backbone size, each value of m/n is treated separately. For each value of backbone size and m/n, a prediction function is derived using each possible combination of predictor variables.

In all cases except one, the F-test for each variable rejected at the 99% confidence level the null hypothesis that the variable accounted for no additional variance. The one exception was where backbone size $= 0.9n$, $m/n = 4.29$ and the only predictor was the number of solutions. In this case, the null hypothesis could not even be rejected at the 95% confidence level. In all other cases, even when the fraction of variance was small, the F-tests judge the variance accounted for to be significant (in the sense that

it is unlikely to be a sampling error). This is because of the very large sample size.

In the case where backbone size is $0.1n$, we can see that the number of solutions is the main factor in the *mrl*, with variance in the log of the number of solutions accounting for between 58% and 64% of the variance in log of *mrl*. The backbone robustness is a minor factor, only accounting for between 2% and 4% of the variance. With both variables as predictors, we can account for between 62% and 66% of the variance.

In the case where backbone size is $0.5n$, we can see that the number of solutions is the main factor in the cost, with variance in the log of the number of solutions accounting for between 45% and 49% of the variance in log of *mrl*. The backbone robustness is also a major factor though, accounting for between 12% and 16% of the variance. With both variables as predictors, we can account for between 57% and 62% of the variance.

In the case where backbone size is $0.9n$, we can see that the backbone robustness is a much greater factor in the cost than the number of solutions. Variance in the backbone robustness accounts for between 20% and 30% of the variance in log of *mrl*. The number of solutions is a very minor factor, only accounting for between 0.4% and 1.7% of the variance. As mentioned, in one case this could just be a result of sampling. With both variables as predictors, we can account for between 22% and 32% of the variance.

In all cases of backbone size and m/n, if we compare the β coefficients of the predictors when both are used to the β coefficents when only one is used, we find that the former is not much smaller than the latter. This indicates that the variance accounted for by each predictor is largely not a result of a correlation between the two predictors themselves.

Summary statistics for backbone size fixed, with m/n as a predictor variable

In this scenario, for each value of backbone size, we aggregated the 8 collections of 1000 instances at different values of m/n into one collection of 8000 instances. We then treat m/n as a predictor variable. Table 4.5 gives all summary statistics on the derived predictor functions for different backbone sizes. For each value of backbone size a prediction function is derived using each possible combination of predictor variables.

m/n	Predictor variables		R^2 (3 s.f.)	Standardised coefficients (3 s.f)		F-statistic (3 s.f.)	
	etbbr	*lgsolns*		β_{etbbr}	$\beta_{lgsolns}$	*etbbr*	*lgsolns*
4.03	×		0.0372	−0.193	n/a	38.6	n/a
4.03		×	0.607	n/a	−0.781	n/a	1560
4.03	×	×	0.637	−0.164	−0.775	74.0	1650
4.11	×		0.0222	−0.149	n/a	22.7	n/a
4.11		×	0.592	n/a	−0.770	n/a	1450
4.11	×	×	0.620	−0.166	−0.773	72.7	1570
4.18	×		0.0305	−0.175	n/a	31.4	n/a
4.18		×	0.587	n/a	−0.766	n/a	1420
4.18	×	×	0.625	−0.195	−0.772	101	1590
4.23	×		0.0444	−0.211	n/a	46.4	n/a
4.23		×	0.613	n/a	−0.783	n/a	1580
4.23	×	×	0.651	−0.194	−0.779	108	1730
4.29	×		0.0359	−0.189	n/a	37.2	n/a
4.29		×	0.606	n/a	−0.779	n/a	1540
4.29	×	×	0.631	−0.156	−0.772	65.6	1610
4.35	×		0.0279	−0.167	n/a	28.7	n/a
4.35		×	0.641	n/a	−0.801	n/a	1790
4.35	×	×	0.662	−0.144	−0.797	61.6	1870
4.41	×		0.0311	−0.176	n/a	32.1	n/a
4.41		×	0.606	n/a	−0.778	n/a	1540
4.41	×	×	0.635	−0.171	−0.777	80.2	1650
4.49	×		0.0194	−0.139	n/a	19.8	n/a
4.49		×	0.636	n/a	−0.797	n/a	1740
4.49	×	×	0.657	−0.149	−0.798	61.0	1850

Table 4.2: Statistics for various predictor functions for log *mrl*, backbone size = $0.1n$, $n = 100$.

m/n	Predictor variables		R^2 (3 s.f.)	Standardised coefficients (3 s.f)		F-statistic (3 s.f.)	
	etbbr	*lgsolns*		β_{etbbr}	$\beta_{lgsolns}$	*etbbr*	*lgsolns*
4.03	×		0.139	−0.373	n/a	162	n/a
4.03		×	0.457	n/a	−0.676	n/a	842
4.03	×	×	0.590	−0.365	−0.672	325	1100
4.11	×		0.151	−0.390	n/a	178	n/a
4.11		×	0.467	n/a	−0.683	n/a	876
4.11	×	×	0.601	−0.366	−0.671	335	1120
4.18	×		0.147	−0.384	n/a	173	n/a
4.18		×	0.481	n/a	−0.693	n/a	924
4.18	×	×	0.613	−0.364	−0.682	341	1200
4.23	×		0.133	−0.364	n/a	153	n/a
4.23		×	0.453	n/a	−0.673	n/a	827
4.23	×	×	0.579	−0.356	−0.668	300	1060
4.29	×		0.149	−0.386	n/a	175	n/a
4.29		×	0.479	n/a	−0.692	n/a	919
4.29	×	×	0.617	−0.372	−0.684	361	1220
4.35	×		0.139	−0.373	n/a	162	n/a
4.35		×	0.472	n/a	−0.687	n/a	895
4.35	×	×	0.608	−0.368	−0.684	346	1190
4.41	×		0.119	−0.345	n/a	135	n/a
4.41		×	0.493	n/a	−0.702	n/a	974
4.41	×	×	0.614	−0.348	−0.704	314	1280
4.49	×		0.163	−0.403	n/a	194	n/a
4.49		×	0.484	n/a	−0.695	n/a	936
4.49	×	×	0.623	−0.373	−0.679	367	1220

Table 4.3: Statistics for various predictor functions for log *mrl*, backbone size = $0.5n$, $n = 100$.

m/n	Predictor variables		R^2 (3 s.f.)	Standardised coefficients (3 s.f)		F-statistic (3 s.f.)	
	etbbr	*lgsolns*		β_{etbbr}	$\beta_{lgsolns}$	*etbbr*	*lgsolns*
4.03	×		0.222	−0.471	n/a	285	n/a
4.03		×	0.0171	n/a	−0.131	n/a	17.4
4.03	×	×	0.233	−0.465	−0.104	281	13.9
4.11	×		0.206	−0.454	n/a	259	n/a
4.11		×	0.0162	n/a	−0.128	n/a	16.5
4.11	×	×	0.220	−0.451	−0.118	260	17.8
4.18	×		0.282	−0.531	n/a	392	n/a
4.18		×	0.0134	n/a	−0.116	n/a	13.6
4.18	×	×	0.291	−0.527	−0.0987	392	13.7
4.23	×		0.276	−0.525	n/a	381	n/a
4.23		×	0.0173	n/a	−0.132	n/a	17.6
4.23	×	×	0.287	−0.520	−0.105	377	15.3
4.29	×		0.284	−0.533	n/a	396	n/a
4.29		×	0.00374	n/a	−0.0612	n/a	3.76
4.29	×	×	0.291	−0.536	−0.0836	404	9.83
4.35	×		0.248	−0.498	n/a	330	n/a
4.35		×	0.0109	n/a	−0.104	n/a	11.0
4.35	×	×	0.262	−0.502	−0.118	340	18.7
4.41	×		0.265	−0.515	n/a	361	n/a
4.41		×	0.0139	n/a	−0.118	n/a	14.1
4.41	×	×	0.284	−0.520	−0.137	377	26.2
4.49	×		0.308	−0.555	n/a	445	n/a
4.49		×	0.00984	n/a	−0.0992	n/a	9.94
4.49	×	×	0.320	−0.557	−0.108	455	17.2

Table 4.4: Statistics for various predictor functions for log *mrl*, backbone size = $0.9n$, $n = 100$.

In all cases the F-test for each variable rejected at the 99% confidence level the null hypothesis that the variable accounted for no additional variance. Again the very large sample size means that the null hypothesis will be rejected even for small values of R^2. Using all three predictors allows us to account for 69% of the variance with backbone size $0.1n$, 65% of the variance with backbone size $0.5n$ and 40% of the variance with backbone size $0.9n$. This is somewhat more than when m/n is fixed.

One interesting point is made by comparing the $\beta_{m/n}$ coefficient where m/n is the only predictor with $\beta_{m/n}$ when both other predictors are used. In each case the latter is much smaller, especially for medium and large backbone instances. This indicates that in these cases the main way in which m/n affects the dependent variable is via one of the other predictors.

Back-bone size	Predictor variables			R^2 (3 s.f.)	Standardised coefficients (3 s.f)			F-statistic (3 s.f.)		
	m/n	*etbbr*	*lgsolns*		$\beta_{m/n}$	β_{etbbr}	$\beta_{lgsolns}$	m/n	*etbbr*	*lgsolns*
$0.1n$	×			0.163	−0.404	n/a	n/a	1560	n/a	n/a
$0.1n$		×		0.123	n/a	−0.351	n/a	n/a	1120	n/a
$0.1n$			×	0.544	n/a	n/a	−0.738	n/a	n/a	9560
$0.1n$	×	×		0.187	−0.302	−0.186	n/a	630	239	n/a
$0.1n$	×		×	0.673	−0.359	n/a	−0.715	3140	n/a	12500
$0.1n$		×	×	0.647	n/a	−0.320	−0.724	n/a	2320	11900
$0.1n$	×	×	×	0.695	−0.262	−0.177	−0.714	1260	577	13300
$0.5n$	×			0.146	−0.383	n/a	n/a	1370	n/a	n/a
$0.5n$		×		0.246	n/a	−0.496	n/a	n/a	2610	n/a
$0.5n$			×	0.391	n/a	n/a	−0.626	n/a	n/a	5140
$0.5n$	×	×		0.259	−0.139	−0.414	n/a	136	1210	n/a
$0.5n$	×		×	0.549	−0.397	n/a	−0.635	2800	n/a	7140
$0.5n$		×	×	0.637	n/a	−0.495	−0.625	n/a	5390	8590
$0.5n$	×	×	×	0.654	−0.162	−0.400	−0.629	394	2410	9120
$0.9n$	×			0.194	−0.441	n/a	n/a	1930	n/a	n/a
$0.9n$		×		0.382	n/a	−0.618	n/a	n/a	4930	n/a
$0.9n$			×	0.00445	n/a	n/a	−0.0667	n/a	n/a	35.7
$0.9n$	×	×		0.391	−0.121	−0.547	n/a	127	2590	n/a
$0.9n$	×		×	0.204	−0.448	n/a	−0.0993	2000	n/a	98.5
$0.9n$		×	×	0.390	n/a	−0.622	−0.0943	n/a	5060	117
$0.9n$	×	×	×	0.401	−0.128	−0.547	−0.100	145	2640	134

Table 4.5: Statistics for various predictor functions for log *mrl* for various backbone sizes, using instances aggregated from accross the satisfiability transition, $n = 100$.

4.5 A correct prediction about very backbone-fragile instances

Our hypothesis proposes that high backbone fragility of instances causes high WSAT/SKC cost for those instances. However, it is plausible that the high backbone fragility is a by-product of some unmeasured latent factor and that it is not causally related to the cost. In other words it may be the case that backbone fragility and cost are correlated, but that differences in cost are not actually due to differences in backbone fragility, but due to differences in another factor which is correlated with backbone fragility.

To help establish the causal link between backbone fragility and cost, we therefore created sets of random SAT instances which had higher backbone fragility than usual RANDOM-3-SAT instances. This is to some degree following the methodological precedent of Bayardo and Schrag (1996), who created random instances which contained small unsatisfiable sub-instances but which had few constraints overall. These were often found to be exceptionally hard for the complete procedure TABLEAU. Their experiments thereby helped establish that this feature of instance structure was the cause of exceptionally high cost for complete procedures.

We cannot easily set backbone fragility directly, since it is not a parameter of the generation method. One manipulation experiment which is possible is the use of an instance generation procedure which results in instances with a higher backbone fragility. Our hypothesis predicts that instances generated using such a procedure will be harder than RANDOM-3-SAT instances due to higher backbone fragility. In this section we define such a procedure and test the prediction. It may be that our procedure is also manipulating the latent factor, so this experiment does not have the potential to prove the causal link. However, if the instances with increased backbone fragility are not harder, then the hypothesis would be falsified. So, since the procedure is specifically designed to increase backbone fragility, a correct prediction here still lends some credibility to our hypothesis.

Section 4.5.1 introduces a new concept, that of the backbone-minimal sub-instance which is important for this experiment. Section 4.5.2 uses this to define our procedure which increases backbone fragility and presents results.

4.5.1 Backbone-minimal sub-instances

Suppose we have a SAT instance C and we remove a clause such that the backbone is not affected by the removal of the clause. If such clauses are repeatedly removed, eventually the instance will be such that no clause can be removed without disturbing the backbone. In this case we have a *backbone-minimal sub-instance* (BMS) of C. More formally, we have the following definition:

Definition A SAT instance C' is a BMS of C iff

- C' is a sub-instance of C (i.e. an instance consisting of a sub-bag of the clauses of C) such that C' has the same backbone as C.
- for each clause c of C' there exists a literal l such that:
  1. $C' \rightarrow l$
  2. $(C' - \{c\}) \wedge \neg l$ is satisfiable

  i.e. every strict sub-instance of C' has a strictly smaller backbone than the backbone of C' □

BMSs can be seen as satisfiable analogues of the *minimal unsatisfiable sub-instances* (MUSs) of unsatisfiable instances studied by amongst others Culberson and Gent (1999b) in the context of graph colouring and Gent and Walsh (1996) and Bayardo and Schrag (1996) in satisfiability. An MUS of an instance C is a sub-instance which is unsatisfiable, but such that the removal of any one clause from the sub-instance renders it satisfiable. Just as all unsatisfiable instances must have an MUS, all satisfiable SAT instances must have a BMS. Having a BMS does not depend on having a non-empty backbone – if the backbone of the instance is empty, its BMS is the empty sub-instance. An instance can have more than one BMS. Different BMSs of an instance may share clauses. One BMS of an instance cannot be a strict sub-instance of another.

Suppose the backbone of a satisfiable instance C is the set of literals $\{l_1, l_2, \ldots, l_k\}$. Let d be the clause $\neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_k$. Then we have the following useful fact:

Theorem C' is a BMS of C iff $C' \wedge d$ is an MUS of $C \wedge d$ □

```
procedure FIND-RANDOM-MUS(C)
    C' := C
    ⟨c_1, ..., c_m⟩ := a random permutation of the m clauses of C'
    for i = 1 to m
        C'' := C' − c_i
        if C'' is unsatisfiable
            C' := C''
        end if
    end for
    return C'
```

Figure 4.4: The FIND-RANDOM-MUS procedure, from Gent and Walsh (1996).

A simple proof of the above is given in Appendix B. Due to this fact, methods for studying MUSs can be applied to the study of BMSs. We can study the BMSs of a satisfiable instance C by finding the backbone of C and then studying the MUSs of $C \wedge d$: each of these corresponds to a BMS of C since d must be present in every MUS of $C \wedge d$.

To find a BMS of C we determine the backbone, then find a random MUS of $C \wedge d$ using the same MUS-finding method as Gent and Walsh (1996) and remove d from the result. The method is given in Figure 4.4.

4.5.2 Data on very backbone-fragile instances

Once a BMS C' has been established, we can also study the effects of interpolation between C and C' by removing at random from C some of the clauses which do not appear in C'. This is equivalent to removing clauses at random such that the backbone is preserved. PRESERVE-BACKBONE(C, m_r, C') will denote C with m_r clauses which do not appear in the BMS C', removed at random. The resulting instance will have the same backbone as C.

Just as increasing m/n while controlling the backbone size causes *tbbr* to increase, deleting clauses such that the backbone is unaffected causes *tbbr* to decrease, as one might expect. This is because by removing clauses, we are reducing for each implicate literal the number of sub-formulas in the instance which imply that literal.

We used 500 RANDOM-3-SAT instances with $n = 100$ and $m/n = 4.29$. The in-

Instances	Estimated *tbbr* (3 s. f.)		
	10th percentile	Median	90th percentile
PRESERVE-BACKBONE(C, 0, C')	8.58	13.0	20.6
PRESERVE-BACKBONE(C, 5, C')	7.74	12.0	19.2
PRESERVE-BACKBONE(C, 10, C')	7.20	11.1	17.4
PRESERVE-BACKBONE(C, 20, C')	6.07	9.39	14.5
PRESERVE-BACKBONE(C, 40, C')	4.26	6.49	9.99
PRESERVE-BACKBONE(C, 80, C')	2.07	2.87	3.99
BMS	1.02	1.06	1.16

Table 4.6: The effect of PRESERVE-BACKBONE on the backbone robustness of satisfiable RANDOM-3-SAT instances at $m/n = 4.29$.

stances did not have backbone size controlled. They were simply a set of satisfiable instances at the RANDOM-3-SAT threshold. For each instance we found one BMS using FIND-RANDOM-MUS. The BMSs and the instances from which they were produced have been archived in the SATLIB repository run by Holger Hoos and Thomas Stützle (`www.informatik.tu-darmstadt.de/AI/SATLIB`). We then used PRESERVE-BACKBONE to interpolate with m_r set at various values. Table 4.6 shows the effect of increasing m_r on backbone robustness. The BMSs of the threshold instances are so backbone-fragile that the removal of just one clause from them is likely to reduce the backbone by a half or more.

Our hypothesis predicts that as this interpolation from C to C' proceeds, the cost for local search increases because the backbone robustness decreases. It is conceivable, although it would be very surprising, that removing *any* clauses from random instances near the threshold generally makes their cost for local search increase. If this were the case, any increase in cost during interpolation towards a BMS could merely be due to the removal of clauses *per se* rather than the removal of clauses whilst preserving the backbone. To control for this possibility we also removed clauses according to two other procedures. The procedure RANDOM(C, m_r) removes m_r clauses from C at random. The procedure REDUCE-BACKBONE(C, m_r) removes m_r clauses such that each time a clause is removed, the size of the backbone is reduced. The clause to be removed is chosen randomly from all such clauses. This procedure therefore uses the opposite removal criterion to PRESERVE-BACKBONE. If the backbone becomes empty, no further clauses are removed.

Instances	*mrl* (3 s. f.)		
	10th percentile	Median	90th percentile
PRESERVE-BACKBONE(C, 0, C')	517	1,450	5,180
PRESERVE-BACKBONE(C, 5, C')	537	1,520	5,660
PRESERVE-BACKBONE(C, 10, C')	557	1,610	6,010
PRESERVE-BACKBONE(C, 20, C')	570	1,800	7,040
PRESERVE-BACKBONE(C, 40, C')	643	2,300	10,700
PRESERVE-BACKBONE(C, 80, C')	816	4,150	24,300
BMS	1,560	16,900	136,000

Table 4.7: The effect of PRESERVE-BACKBONE on cost and cost statistics for BMSs.

Figure 4.5 shows the effect on *mrl* of applying the three clause removal procedures to the same set of 500 RANDOM-3-SAT threshold instances. Each line is the median *mrl*, plotted on a log scale.

We observe that removing clauses randomly or such that the backbone is strictly reduced, causes cost to be reduced, so the removal of clauses does not in itself cause higher cost. The REDUCE-BACKBONE procedure causes a greater initial fall in cost, as the backbone size is reduced more quickly than with RANDOM. However, the cost then stabilises for REDUCE-BACKBONE because the backbone becomes empty and thereafter no more clauses are removed.

Removing clauses according to PRESERVE-BACKBONE causes the local search cost to increase by an amount approximately exponential in the number of clauses removed. Table 4.7 gives more data on this effect and also cost data for BMSs. The 10th and 90th percentiles suggest that the interpolation shifts the whole distribution up, not just the median. The median cost of the BMSs, which are the most backbone-fragile of all the instances, is more than three times that of the 90th cost percentile of RANDOM-3-SAT instances.

The BMSs of these instances had between 254 and 318 clauses. The above results therefore demonstrate the existence of instances in the underconstrained region which are much harder than the typical instances from near the satisfiability threshold. However since these were not obtained by sampling from RANDOM-3-SAT directly, we do not know how often they occur. As far as we know, they are vanishingly rare and therefore, in contrast to exceptionally hard instances for complete algorithms, it seems

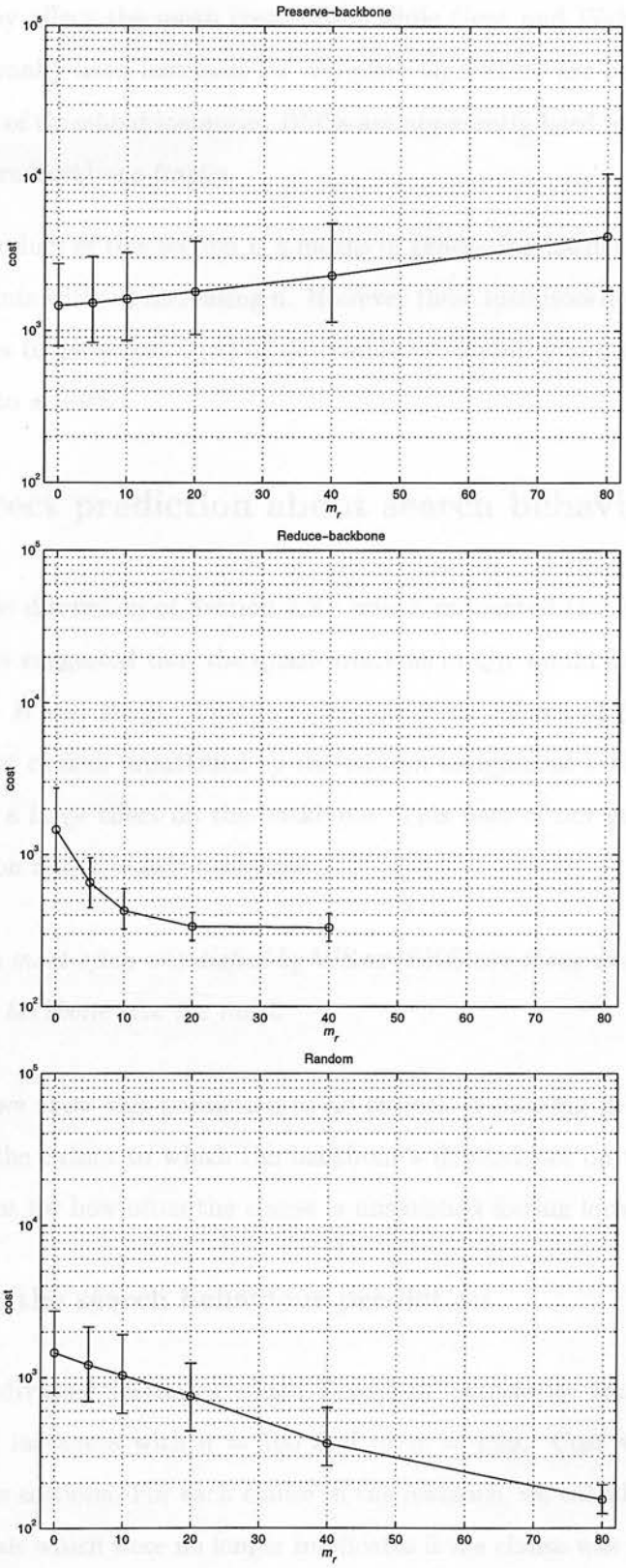

Figure 4.5: The effect of the three clause removal procedures on the median *mrl* of satisfiable RANDOM-3-SAT instances at $m/n = 4.29$. Note that the REDUCE-BACKBONE procedure data stops at $m_r = 40$, at which point all instances had empty backbones. The length of the bars is the interquartile range (25th – 75th percentiles).

unlikely that they affect the mean cost. Also, while Gent and Walsh (1996) showed that the exceptionally hard instances for complete algorithms are hard for a different reason from that of threshold instances, BMSs are apparently hard for the same reason – because they are backbone-fragile.

One useful by-product of this section is a means of generating harder test instances for local search variants without increasing n. However these instances do require $O(m+n)$ complete searches to generate: $O(n)$ to determine satisfiability and the backbone and $O(m)$ to reduce to a BMS.

4.6 A correct prediction about search behaviour

Recall that in the discussion of Section 4.2.1 which motivated the backbone fragility hypothesis, it was suggested that the quasi-solutions in Q_B would be attractive if the backbone of $C - B$ was small. That is to say that the clauses of B are more likely to equal the set of clauses unsatisfied by the current assignment if the removal of the clauses of B has a large effect on the backbone. This part of our general model also makes a prediction about search behaviour:

> *The clauses most often unsatisfied by* WSAT/SKC *are those whose removal reduces the backbone size the most.*

In Section 4.6.1 we show this prediction to be correct. Following this, we investigate in Section 4.6.2 the extent to which the backbone's dependence on the presence of a clause can account for how often the clause is unsatisfied during local search.

4.6.1 Testing the search behaviour prediction

We looked at individual instances which were cost percentiles from a set of 5000 RANDOM-3-SAT instances with $n = 100$ and $m/n = 4.29$. Cost was measured by *mrl* as in previous sections. For each clause in the instance, we calculated the number of backbone literals which were no longer implicates if the clause was removed. This is a simple measure of the *backbone contribution* (*bc*) of the clause – how much the backbone size depends on the presence of the clause. If a clause's backbone contribution

is high, it is termed a *backbone-critical* clause. We made 1000 runs of WSAT/SKC on each instance under the same conditions as in previous sections. During search, each time the current assignment changed we recorded whether each clause was unsatisfied. The result of averaging the number of times the clause was unsatisfied over all assignment-changes in all runs gives the *unsatisfaction frequency* (uf) of that clause. Although uf is measured over the whole run, one could argue that whether or not a clause is unsatisfied during the hill-climbing phase is unimportant. However, including the hill-climbing phase makes little difference to the results because:

1. The hill-climbing phase is so short compared to the rest of the run.
2. At least at the beginning of the hill-climbing phase all clauses have an equal chance of being unsatisfied, so overall the unsatisfaction frequency at this stage will be equal.

Figure 4.6 shows a scatter plot of these two quantities for the clauses of the instance whose cost was the median of 5000 threshold instances. We note from this figure that the clauses whose presence contributes the most to the backbone are more often unsatisfied than average during WSAT/SKC search.

Table 4.8 confirms this pattern. Each row of the table gives data for one instance. We selected cost percentiles; individual instances of varying degrees of difficulty. For example the row labelled '30th' corresponds to the instance whose cost is the 1500th in rank from the easiest to the most difficult of the 5000 instances, while the 50th percentile instance is the one used to produce Figure 4.6. The third and fourth columns give the mean and standard deviation of the unsatisfaction frequency over all clauses in the instance and the last two columns give the same statistics for the sub-bag of the clauses which were most backbone-critical (their backbone contribution was in the top 10%).

Table 4.9 shows that the converse effect is also present: the clauses which are most often unsatisfied (their unsatisfaction frequency is in the top 10%) are more backbone-critical than average.

Cost Percentile	Backbone size	All clauses		Most backbone-critical clauses	
		uf mean (3 s. f.)	*uf* std. dev. (3 s. f.)	*uf* mean (3 s. f.)	*uf* std. dev. (3 s. f.)
10th	16	11.3	8.07	20.9	8.87
20th	11	13.1	11.0	30.2	16.9
30th	13	21.0	16.4	41.5	21.1
40th	36	23.0	21.3	56.7	27.5
50th	48	29.6	25.9	72.1	38.8
60th	25	36.3	35.6	96.2	54.3
70th	63	52.4	48.1	120	66.8
80th	70	92.3	87.8	167	150
90th	93	108	127	307	199

Table 4.8: Unsatisfaction frequencies of clauses in different cost percentile instances.

Cost Percentile	Backbone size	All clauses		Most often unsatisfied clauses	
		bc mean (3 s. f.)	*bc* std. dev. (3 s. f.)	*bc* mean (3 s. f.)	*bc* std. dev. (3 s. f.)
10th	16	0.592	1.24	2.09	1.85
20th	11	0.485	1.04	1.77	1.96
30th	13	0.396	1.24	1.84	2.45
40th	36	1.81	4.24	8.32	6.46
50th	48	1.06	3.28	6.32	6.60
60th	25	1.38	3.49	7.75	5.78
70th	63	3.39	8.86	14.9	15.8
80th	70	0.695	3.46	2.60	7.86
90th	93	3.07	10.0	17.0	20.6

Table 4.9: Backbone contributions of clauses in different cost percentile instances.

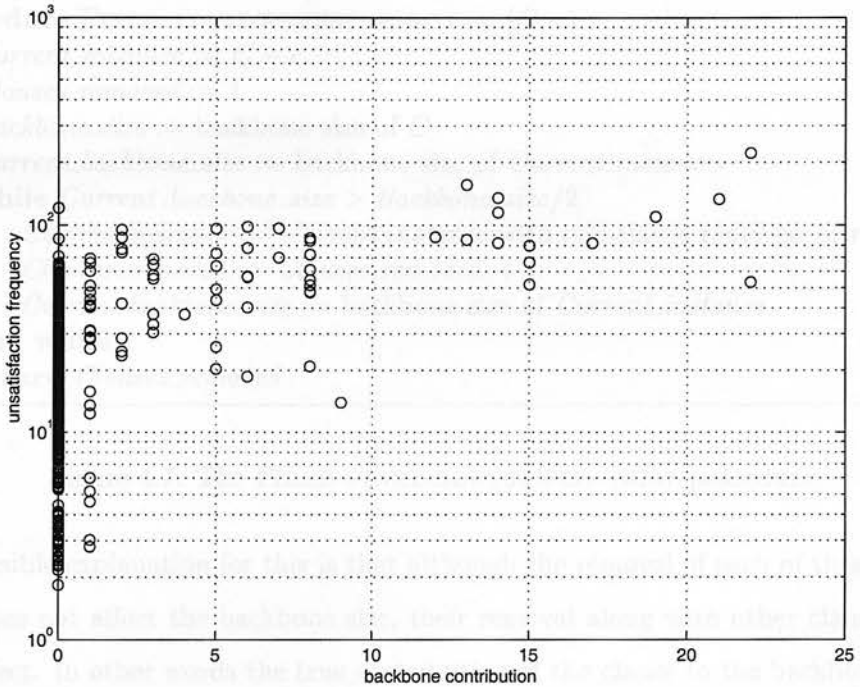

Figure 4.6: Scatter plot of unsatisfaction frequency against backbone contribution for the clauses of the cost median of 5000 instances, $m/n = 4.29$, $n = 100$.

For instances of different costs at the satisfiability threshold, the clauses which are most likely to be unsatisfied during search have a higher backbone contribution than average. Conversely, the clauses which have the largest backbone contribution are more likely to be unsatisfied during search. As well as satisfying the prediction, this experiment also demonstrates that as well as explaining differences in cost between instances, the backbone fragility hypothesis can also explain differences in the difficulty of satisfying particular clauses during search.

4.6.2 Can search behaviour be explained further using backbone robustness?

Although a difference in the sample means is apparent in Table 4.9, there are sometimes particularly large standard deviations in *bc* values for the most frequently unsatisfied clauses. This is because, as can be seen from Figure 4.6, some clauses are very often unsatisfied even though removing them on their own does not affect the backbone size at all.

```
procedure FIXED-START-ROBUSTNESS-TRIAL(C, c)
    Current_instance := C − c
    Clauses_removed := 1
    Backbone_size := backbone size of C
    Current_backbone_size := backbone size of Current_instance
    while Current_backbone_size > Backbone_size/2
        Current_instance := Current_instance with one clause removed at random
        Clauses_removed := Clauses_removed + 1
        Current_backbone_size := backbone size of Current_instance
    end while
    return Clauses_removed
```

Figure 4.7: The FIXED-START-ROBUSTNESS-TRIAL procedure.

One possible explanation for this is that although the removal of each of these clauses alone does not affect the backbone size, their removal along with other clauses has a large effect. In other words the true contribution of the clause to the backbone size is underestimated by *bc*.

In this section we investigate whether a more refined measure of backbone contribution might provide a more accurate estimate and might therefore be correlated more tightly with the unsatisfaction frequency.

Conditional *tbbr*: a more accurate measure of backbone contribution

We wanted to measure more accurately how much each clause contributed to the overall size of the backbone. If the backbone size is dependent on the presence of a clause, this could mean either that the removal of the clause on its own will reduce the backbone or that the removal of the clause will render the instance more backbone-fragile than it was, or some combination of these. To capture this, we measure the backbone contribution of a clause by *conditional tbbr*[4]. To define this, we use *fixed-start* robustness trials as defined in Figure 4.7. These are similar to regular robustness trials, but the user specifies a clause which is always the first clause removed.

Definition We define the *tbbr* of an instance C *conditional on the clause* c, where c

[4] In fact, conditional *tbbr* will be high when the backbone contribution of a clause is low and low when the backbone contribution is high, so in moving from *bc* to conditional *tbbr* the metric has "changed direction".

is a clause of C. This is the mean result of all possible runs of FIXED-START-ROBUSTNESS-TRIAL(C, c) □

Observe that the conditional *tbbr* is therefore the *tbbr* given that every robustness trial starts with the removal of c. We estimated conditional *tbbr* by random sampling. For each clause c on each instance C we conducted at least 100 robustness trials (each beginning with c) and continued to sample until the standard error was less than 0.05 $\times$ the sample mean[5]. Computing conditional *tbbr* for every clause of an instance was computationally costly and was only done for those instances studied in Section 4.6.1.

Results using conditional *tbbr*

Figures 4.8, 4.9 and 4.10 show semi-log scatter plots showing the relationship between unsatisfaction frequency and conditional *tbbr* in each of the nine instances used in Section 4.6 (each scatter plot represents one instance and each point represents one clause).

There is a clear relationship between our chosen measure of a clause's contribution to the backbone and the clause's unsatisfaction frequency. The shape of the scatter plot appears to be very consistent among instances of widely different costs at the satisfiability threshold. The common feature is the absence of clauses with both a low unsatisfaction frequency and a low conditional *tbbr*. We can therefore safely conclude that on these instances, any clause whose presence contributes significantly to the backbone will often be unsatisfied relative to the other clauses.

However, there are also many infrequently-satisfied clauses which contribute little to the backbone. This pattern is essentially similar to our first measure of backbone contribution. Therefore, in conclusion, refining the measure of backbone contribution does not help us explain the variation in satisfaction frequency. These clauses may be infrequently-satisfied for some other reason not related to the backbone and so demonstrate the limitations of backbone-based explanations of this aspect of search behaviour. It is likely that our assumption about which assignments are most of-

[5] Therefore the conditions used to measure conditional *tbbr* were similar here to those used for our estimates of non-conditional *tbbr* – see Section 4.2.2.

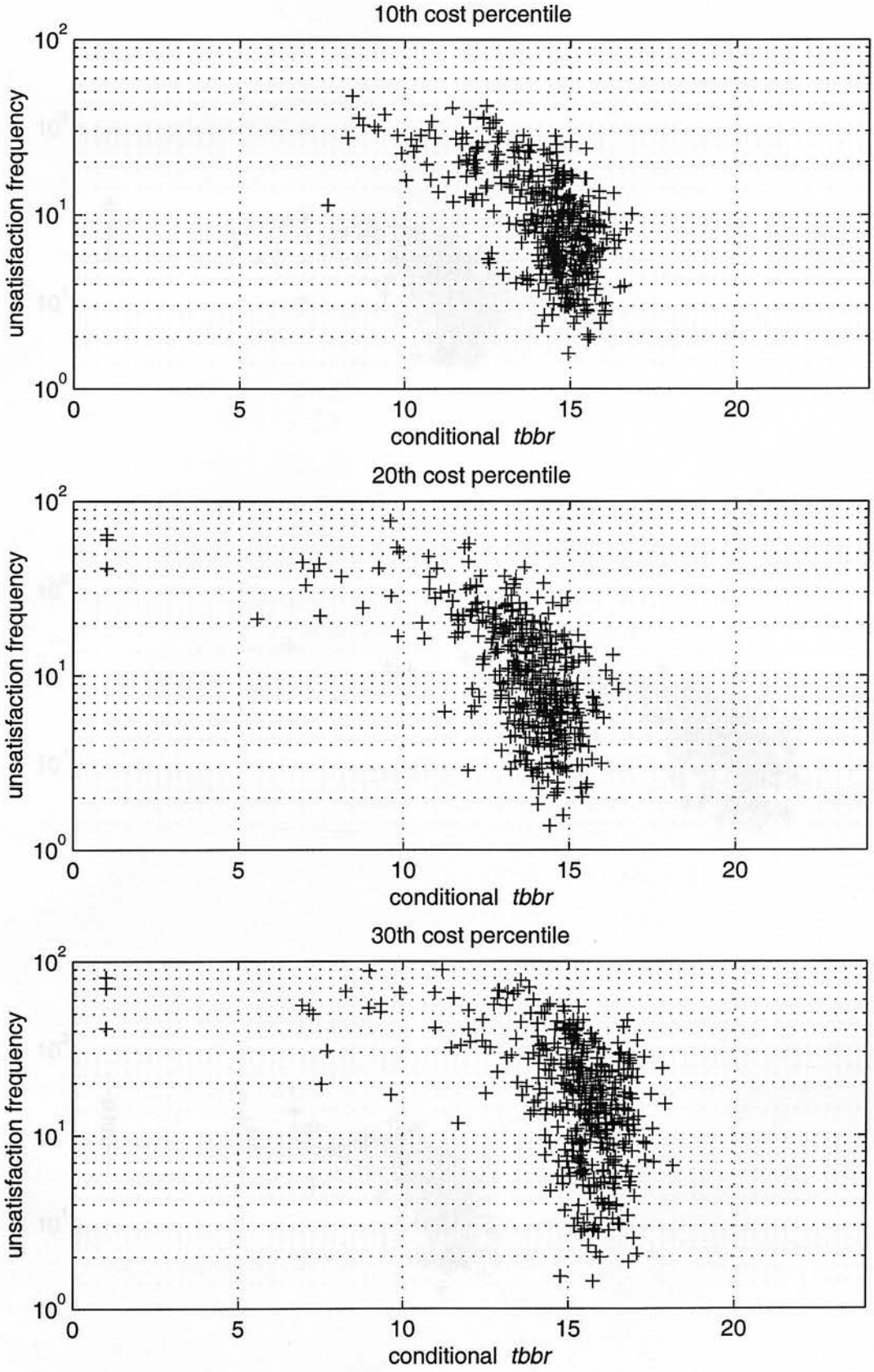

Figure 4.8: The relationship between unsatisfaction frequency and conditional *tbbr* for the 10th, 20th and 30th cost percentiles.

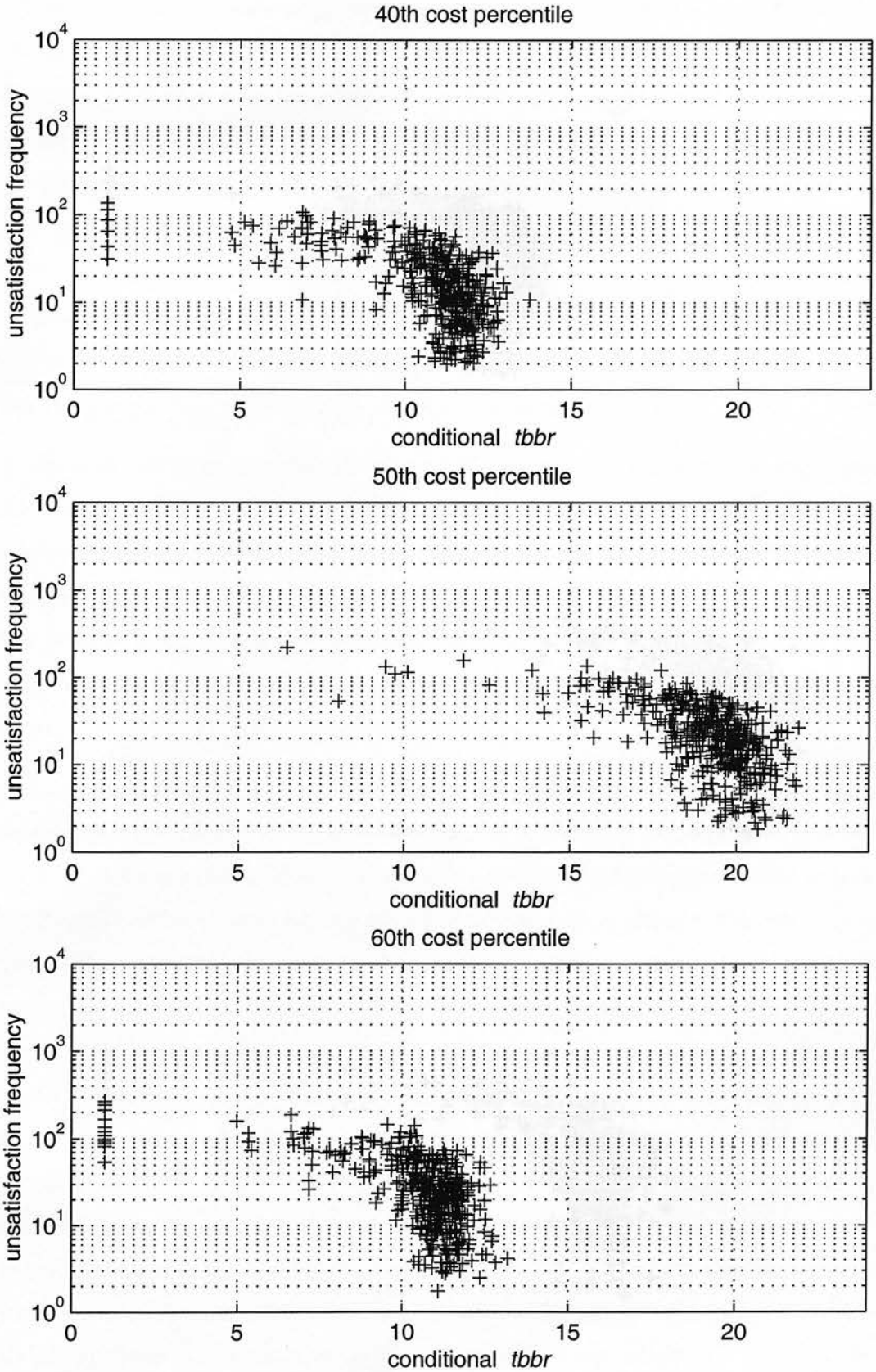

Figure 4.9: The relationship between unsatisfaction frequency and conditional *tbbr* for the 40th, 50th and 60th cost percentiles.

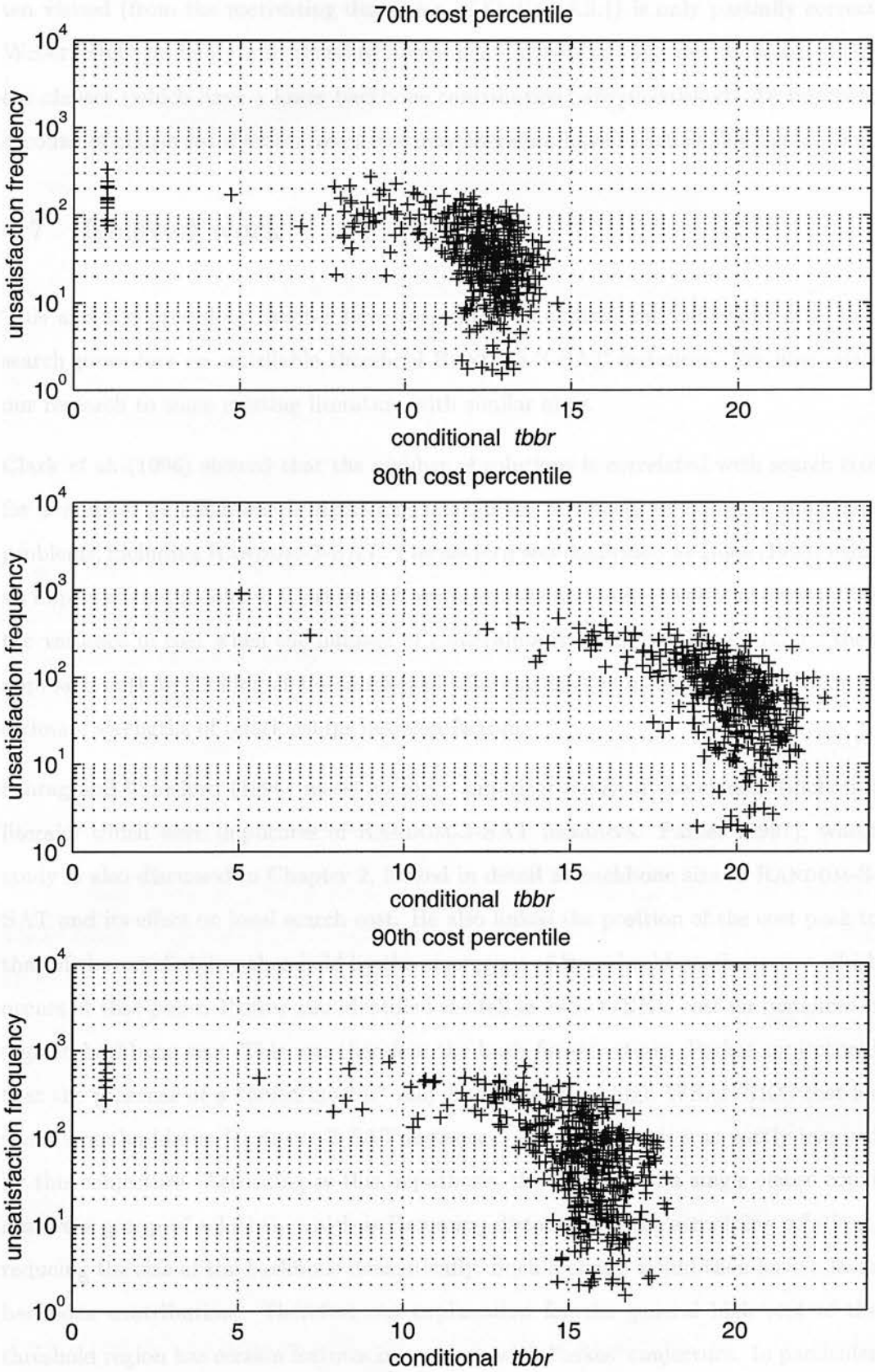

Figure 4.10: The relationship between unsatisfaction frequency and conditional *tbbr* for the 70th, 80th and 90th cost percentiles.

ten visited (from the motivating discussion of Section 4.2.1) is only partially correct. WSAT/SKC probably also searches assignments closer to solutions at which different clauses (which have a lower backbone contribution) are unsatisfied. An improved account of clause satisfaction frequency may therefore have to refine this assumption.

4.7 Related work

This and the preceding chapter have attempted to explain the behaviour of a local search procedure on satisfiable threshold RANDOM-3-SAT instances. We now relate our research to some existing literature with similar aims.

Clark *et al.* (1996) showed that the number of solutions is correlated with search cost for a number of local search algorithms on random instances of different constraint problems, including RANDOM-3-SAT. The pattern was confirmed by Hoos (1998) using an improved methodology. Clark *et al.*'s work was the first step towards understanding the variance in cost when the number of constraints is fixed. We have followed their approach both by looking at the number of solutions and by using linear regression to estimate strengths of relationships between factors.

Schrag and Crawford (1996) made an early empirical study of the clauses (including literals) which were implicates of RANDOM-3-SAT instances. Parkes (1997), whose study is also discussed in Chapter 2, looked in detail at backbone size in RANDOM-3-SAT and its effect on local search cost. He also linked the position of the cost peak to that of the satisfiability threshold by the emergence of large-backbone instances which occurs at that point. Parkes also identified the fall in WSAT/SKC cost for instances of a given backbone size. This was therefore the basis for our study. Parkes conjectured that the presence of a "failed cluster" may be the cause of high WSAT/SKC cost for some large-backbone RANDOM-3-SAT instances. Our hypothesis was partly inspired by this conjecture. According to this hypothesis, the addition of a single clause could remove a group of solutions which is Hamming distant from the remaining solutions, reducing the size of the backbone dramatically. Such a clause would then have a large backbone contribution. Therefore our explanation for the general high cost of the threshold region has certain features in common with Parkes' conjecture. In particular

we agree that it is the presence of clauses with a large backbone contribution which causes high cost. This is especially demonstrated by our results from Section 4.6. However, our explanation differs from Parkes' conjecture in that we propose that in hard instances the quasi-solutions which cause difficulties for WSAT/SKC are not grouped in a single cluster but are widely-distributed in different areas of the search space.

Frank *et al.* (1997) studied in detail the topology of the GSAT search space induced by different classes of random SAT instances. Their study discussed the implications of search space structure for future algorithms, as well as the effects of these structures on algorithms such as GSAT. They also noted that some local search algorithms such as WSAT/SKC may be blind to the structures they studied because they search in different ways to GSAT.

Yokoo (1997) also addressed the question of why there is a cost peak for local search as m/n is increased. The approach was to analyse the whole search space of small satisfiable random instances. While in this study, we have only examined SAT, Yokoo also showed his results generalised to the colourability problem. Yokoo used a deterministic hill-climbing algorithm. He studied the number of assignments from which a solution is reachable (solution-reachable assignments) via the algorithm's deterministic moves, which largely determines the cost for the algorithm.

We followed Yokoo in looking for a factor competing with the number of solutions whose effect on cost changes as m/n is increased. The factor which Yokoo proposed as the cause of the overall fall in cost was the decrease in the number of local minima – assignments from which no local move decreased the number of unsatisfied clauses. The decrease in this number was demonstrated as m/n is increased. The decrease was attributed to the decreasing size of "basins" (interconnected regions of local minima with the same number of unsatisfied clauses). Yokoo claimed (p. 363) that:

> "adding constraints [...] makes the [instance] easier by decreasing the number of local minima".

However, we do not think it is clear *a priori* what the relationship between the number of local minima and the cost is in a given instance and Yokoo did not study it sufficiently

for his explanation to be convincing. In contrast with Yokoo, we have studied in detail the relationship between the backbone fragility of instances and WSAT/SKC's cost on these instances and confirmed it by testing predictions of our hypothesis. Also, we studied instance properties that related to the logical structure of the clauses rather than the search space topology which was induced as we think this has more potential to generalise across algorithms and even to address complexity issues, as we explain towards the end of this section.

Hoos (1998) also analysed the search spaces of SAT instances in relation to local search cost by looking at two new measures of the induced objective function which he defined, including one based on local minima. Although via these measures, Hoos was not able to account for the RANDOM-3-SAT cost peak, he found that the features were correlated with cost for some SAT encodings of other problems and has also shown (Hoos 1999b) that his measures can help distinguish between alternative encodings of the same search task.

How does the pattern we have uncovered fit in to other work on what makes instances require a high cost to solve? Gent and Walsh (1996), using random SAT instances[6] looked at the probability that an unsatisfiable instance becomes satisfiable if a fixed number of clauses are removed at random. They found that at the SAT threshold, the unsatisfiable instances which had the highest computational cost for a DPLL variant were those which were *unsatisfiability-fragile* – their unsatisfiability was sensitive to the random removal of clauses. It may therefore be that the fragility of an instance's unsatisfiability or backbone size is the cause of high computational cost both in the context of complete procedures and incomplete local search, which would be an interesting link between the two algorithm classes. However it should always be noted when comparing the two classes of algorithm that research on local search has concentrated on only satisfiable instances, while complete algorithms are run on both satisfiable and unsatisfiable instances. The theory may also represent a link between the reasons for hardness in satisfiable and unsatisfiable instances. This link may form the basis of a possible explanation of the reasons why threshold RANDOM-3-SAT instances may be *universally* hard in the average case, as opposed to merely costly for some class of

[6] Although Gent and Walsh's instances were not generated by the RANDOM-3-SAT method, they were generated by a simple stochastic method which exhibits a satisfiability threshold.

algorithms.

4.8 Summary

We now summarise Chapter 4. We hypothesised that differences in backbone fragility were the cause of two phenomena of cost behaviour observed in Chapter 3. Firstly, the decay in cost for large backbone instances as m/n is increased. Secondly the variation in cost for large backbone instances when m/n is fixed. We motivated this hypothesis and provided a practical method for measuring backbone robustness. We used this to measure the backbone robustness of the threshold instances studied in Chapter 3.

We were then able to test predictions of the hypothesis that the fall in backbone fragility is the cause of the overall decay in cost as m/n is increased. We found that the hypothesis made three correct predictions. Firstly that the degree to which an instance is backbone-robust is correlated with the cost when the effects of other factors are controlled for. If we combine backbone fragility with other factors, we can account for more variance in cost than before. Secondly, that when RANDOM-3-SAT instances are altered so as to be more backbone-fragile (by removing clauses without disrupting the backbone) their cost increases. Thirdly, that the clauses most often unsatisfied during search are those whose deletion has most effect on the backbone.

We now summarise the overall picture which can be determined from this chapter if one accepts the speculative account from Chapter 3. In the underconstrained region, instances with small backbones are predominant. In this region, the rapid hill-climbing phase typically results in an assignment which is close to the nearest solution (and probably satisfies the backbone). Since finding the small backbone is largely accomplished by hill-climbing, typical cost for WSAT/SKC is low in this region and variance in cost is due to variance in the density of solutions in the region of the search space where the backbone is satisfied.

In the threshold region, large-backbone instances quickly appear in large quantities. For large-backbone instances, the main difficulty for local search is to identify the backbone rather than to find a solution once the backbone has been identified. The identification of a large backbone may be accomplished by the rapid hill-climbing

phase to a greater or lesser extent. We think that the effectiveness of the hill-climbing phase is determined by the backbone fragility of the instance. If a large-backbone instance is backbone-fragile the hill-climbing phase is ineffective and results in an assignment which is Hamming-distant from the nearest solution (probably implying that much of the backbone has not been identified). Then a costly plateau search is required to find a solution. Hence when the rare large-backbone instances do occur in the underconstrained region, they are extremely costly to solve because of their high backbone fragility.

If a large-backbone instance is more backbone robust, the rapid hill-climbing phase is more effective in determining the backbone and the plateau phase is shorter. So overall the instance is less costly for WSAT/SKC to solve. Hence for large-backbone instances, since backbone robustness increases as m/n is increased, cost decreases. In the overconstrained region, large backbone instances are dominant and so backbone fragility becomes the main factor determining cost. Hence cost decreases in this region. Our hypothesis proposes the following explanation for the cost peak: *Typical cost peaks in the threshold region because of the appearance as m/n increases of many large-backbone instances which are still moderately backbone-fragile, followed by the increasing backbone robustness of these instances.*

Chapter 5

Local Search on RANDOM-2+p-SAT

5.1 Introduction

In k-SAT, the value of k affects the worst-case complexity. For $k = 2$ there is a worst-case linear time algorithm (Aspvall *et al.* 1979) and so 2-SAT and sub-problems such as RANDOM-2-SAT are not considered hard problems in the typical case. 3-SAT on the other hand is NP-complete and, as was discussed in Chapter 2, RANDOM-3-SAT instances near the threshold are apparently hard in the typical as well as the worst case for a range of substantially different algorithms, leading to the conjecture that threshold RANDOM-3-SAT instances are universally hard in the typical case.

In this chapter we use the RANDOM-2+p-SAT generation method due to Monasson *et al.* (1999a). This is related to the RANDOM-k-SAT problem generator introduced in Chapter 2. RANDOM-2+p-SAT instances consist of a mixture of 2-clauses and 3-clauses. The parameter p governs the ratio of 3- to 2-clauses in the instance. By varying p, we may interpolate between RANDOM-2-SAT ($p = 0$) and RANDOM-3-SAT ($p = 1$).

The underlying notion which motivates the use of the RANDOM-2+p-SAT generation method is this. Given that RANDOM-2-SAT is not universally hard in the typical case and that RANDOM-3-SAT is, then typical case hardness must emerge at some point *between* $k = 2$ and $k = 3$. By studying the properties of RANDOM-2+p-SAT as p is varied, both via analysis and experiment, we can gain an understanding of why

typical-case hardness emerges. Studies using RANDOM-2+p-SAT therefore attack the same question of explaining problem hardness as those which study the RANDOM-3-SAT threshold: whereas RANDOM-3-SAT studies ask why hardness emerges as m/n is varied, studies of RANDOM-2+p-SAT ask why hardness emerges as p is varied.

It is conjectured that some structural properties emerge when p equals a particular value p_0 (discussed in detail in Section 5.2). Monasson *et al.* also studied the effect of varying p on typical (i.e. median) search cost for TABLEAU, a highly optimised variant of DPLL. They observed a transition from linear typical case scaling to simple exponential typical case scaling. Their experimental results were consistent with the change in cost scaling coinciding with p_0.

One aim of this chapter is to study the typical cost scaling of local search on RANDOM-2+p-SAT. This allows a cross-algorithm comparison with TABLEAU, to see whether these cost scaling patterns may be algorithm-independent in the RANDOM-2+p-SAT model. In this chapter we use the WSAT/NOVELTY$^+$ local search algorithm (Hoos 1999a; McAllester *et al.* 1997). This is a state-of-the-art variant of WSAT which outperforms WSAT/SKC on random instances. Our experimental data lends some credibility to the hypothesis that the typical case complexity of RANDOM-2+p-SAT changes near p_0.

We also investigate which emergent structural properties might cause the change in typical cost for local search as p is varied. The approach we take is to look for structural properties which distinguish typical instances where $p < p_0$ from those where $p > p_0$ and which could conceivably affect the algorithm's operation.

Section 5.2 gives some further background on RANDOM-2+p-SAT and details of the test instances which were used. In Section 5.3 we discuss the experimental conditions under which WSAT/NOVELTY$^+$ was run. Section 5.4 presents experimental scaling data for the search cost of WSAT/NOVELTY$^+$ on RANDOM-2+p-SAT. In Section 5.5 we relate the structural properties of RANDOM-2+p-SAT to the behaviour of local search cost. Finally, Section 5.6 summarises this chapter. Material from this chapter was published in a paper presented at *ECAI-2000* (Singer *et al.* 2000b).

5.2 RANDOM-2+p-SAT instances

This section gives some further background on RANDOM-2+p-SAT and details of the instance collections which were used in the experimental work of this chapter.

Definition A RANDOM-2+p-SAT instance consists of $(1-p)m$ RANDOM-2-SAT 2-clauses and pm RANDOM-3-SAT 3-clauses. All clauses are generated using the same set of n variables.□

It is possible to mathematically analyse RANDOM-2-SAT in some detail. For example the location of the threshold can be predicted. Monasson *et al.* suggested that the range over which p varies in RANDOM-2+p-SAT may be divided into two qualitatively different regions delineated by a value p_0. The value p_0 is defined such that with $p < p_0$, RANDOM-2+p-SAT behaves like RANDOM-2-SAT in the sense that in the limit as $n \to \infty$ the 3-clauses almost surely do not affect the satisfiability of the formula. This implies that only the 2-clauses are relevant and that the analysis possible on RANDOM-2-SAT, such as predicting the location of the threshold, can also be extended to RANDOM-2+p-SAT. With $p > p_0$, the 3-clauses become relevant and so the behaviour of RANDOM-2+p-SAT is then thought to be more similar to that of RANDOM-3-SAT.

Monasson *et al.* (1999a) show that for $p < p_0$ the RANDOM-2+p-SAT satisfiability probability changes continuously in the limit. Bounds of 2/5 (lower) and 0.695 (upper) have been established for p_0 (Achlioptas *et al.* 1997). Monasson *et al.* estimate p_0 numerically (under certain statistical mechanics assumptions) at about 0.41. For $p > p_0$ the change in the satisfiability probability is thought to be discontinuous[1].

For TABLEAU, the observed typical cost scaled linearly for $p \leq 0.4$ but, as in RANDOM-3-SAT, it scaled as a simple exponential for $p = 0.6$. This is consistent with a change in typical case complexity due to the change in structure at p_0. If this change in search cost scaling applies to substantially different algorithms (such as local search) and is not peculiar to TABLEAU or its algorithm class, this would suggest more strongly that the structural properties which emerge at p_0 cause the onset of typical-case intractability.

[1] See e.g. Achlioptas *et al.* (1997) for a fuller explanation of of the meaning of (dis)continuous in this context.

p	(n, m) points at approx. 50 % satisfiability
0.0	(25, 41) (50, 74) (100, 136) (250, 313) (500, 598) (1000, 1147) (1500, 1690) (2000, 2228)* (3000, 3288)†
0.3	(25, 52) (50, 95) (100, 177) (250, 414) (500, 799) (1000, 1555)* (1500, 2299)† (2000, 3052)†
0.4	(25, 58) (50, 105) (100, 196) (250, 463) (500, 897) (750, 1328)* (1000, 1758)* (1500, 2610)† (2000, 3456)†
0.5	(25, 63) (50, 115) (100, 217) (250, 520) (500, 1015)* (750, 1511)* (1000, 2005)† (1500, 2991)†
1.0	(25, 112) (50, 218) (100, 428) (150, 640) (200, 853)* (250, 1065)†

Table 5.1: Details of RANDOM-2+p-SAT test instances. Each instance collection contained 5,000 instances, except those marked $*$ which contained 2,500 and those marked †, which contained 1,500.

In this study we used instances generated from the RANDOM-2+p-SAT generation method using various values of p and n. For each combination of p and n we determined the largest value of m for which at least 50 % of the instances were satisfiable. We generated a large set of satisfiable instances at this point, using both TABLEAU (Crawford and Auton 1996) and MODOC (van Gelder and Okushi 1999) as SAT testers. Table 5.1 gives details of these test instances. Collection sizes were reduced for larger n so that the computation time required was manageable.

5.3 Experimental conditions

The WSAT/NOVELTY$^+$ algorithm is discussed in detail in Sections 2.3.7 and 2.3.8. In this section we discuss the experimental conditions which were used to evaluate the performance of WSAT/NOVELTY$^+$ on the RANDOM-2+p-SAT instances.

Following Hoos (1998), we test the algorithm without a restart mechanism and use the median run length (*mrl*) as our measure of per-instance search cost: this represents a "typical" run of WSAT/NOVELTY$^+$ on the instance. We took the *mrl* of 100 runs of the algorithm per instance except on collections marked $*$ where 50 runs per instance were made and collections marked † where 30 runs were made. Following Monasson *et al.* (1999a) we then look at the typical per-instance cost (i.e. median *mrl*) over the collection as n is increased. The *wp* parameter was set to 0.01.

Parameter optimisation for WSAT/NOVELTY$^+$ on RANDOM-2+p-SAT

The remainder of this section deals with the optimisation of one of the algorithm's parameters. The algorithm has a *noise* parameter ($0.0 \leq noise \leq 1.0$). The role of *noise* is explained in detail in Chapter 2. To test the algorithm's performance scaling, this parameter should be optimised if possible. To complicate matters, the optimal *noise* setting at 50 % satisfiability depends on p, n and the measure being optimised. For those instance collections where it was possible, we determined the *noise* setting which was optimal to within 0.05, minimising the mean total number of flips to perform all runs on each instance to completion[2]. We term this optimisation criterion "Mean total flips". On three collections ($p = 0.3$, $n = 25, 50, 100$) 100 runs per instance were not enough to distinguish a clear *noise* optimum. This is because of the rare occurrence within these collections of runs which are occasionally many times longer than typical runs on typical instances. On these collections we found that if 2,000 runs per instance were performed a clear *noise* optimum could be identified.

On certain collections ($p = 0.3$, $n \geq 1000$, $p = 0.4$, $n \geq 500$, $p = 0.5$, $n \geq 500$) it was not feasible to allow all runs to complete and so instead we imposed a cut-off run length large enough so as not to affect our measurement of the median *mrl*. On these collections, since the cut-off prevented the measurement of Mean total flips, we used the *noise* setting which minimised the median *mrl* (optimal to within 0.05) – the optimisation criterion is "Median *mrl*". Data on optimal noise settings is given in Table 5.2.

One possibility was to use the median *mrl* as the optimisation criterion for all settings of p and n, particularly as we will use this measure to study cost scaling. However this approach also has difficulties: for certain values of p and n the optimal *noise* value using this criterion is very low. For example, Figure 5.1 shows the dependence of both Median *mrl* and Mean total flips on *noise* for $p = 0$, $n = 50$. Note that the values of *noise* which optimise median *mrl* and mean total flips are very different (0.05 and 0.65 respectively). This is because for these collections, a low-noise (i.e. greedy) search solves the typical instances slightly more quickly, leading to a lower median. However,

[2] So, for example, this is a different optimality criterion from that of McAllester *et al.* (McAllester *et al.* 1997).

p	n	Runs per instance	Optimisation criterion	Optimal *noise* to within 0.05
0	25	100	Mean total flips	0.6
	50	100	Mean total flips	0.65
	100	100	Mean total flips	0.7
	250	100	Mean total flips	0.75
	500	100	Mean total flips	0.8
	1000	100	Mean total flips	0.8
	1500	100	Mean total flips	0.85
	2000	50	Mean total flips	0.85
	3000	30	Mean total flips	0.85
0.3	25	2000	Mean total flips	0.75
	50	2000	Mean total flips	0.7
	100	2000	Mean total flips	0.85
	250	100	Mean total flips	0.75
	500	100	Mean total flips	0.6
	1000	50	Median *mrl*	0.8
	1500	30	Median *mrl*	0.8
	2000	30	Median *mrl*	0.8
0.4	25	100	Mean total flips	0.85
	50	100	Mean total flips	0.9
	100	100	Mean total flips	0.65
	250	100	Median *mrl*	0.75
	500	100	Median *mrl*	0.75
	750	50	Median *mrl*	0.8
	1000	50	Median *mrl*	0.75
	1500	30	Median *mrl*	0.8
	2000	30	Median *mrl*	0.8
0.5	25	100	Mean total flips	0.8
	50	100	Mean total flips	0.85
	100	100	Mean total flips	0.95
	250	100	Mean total flips	0.75
	500	50	Median *mrl*	0.75
	750	50	Median *mrl*	0.75
	1000	30	Median *mrl*	0.75
	1500	30	Median *mrl*	0.7
1.0	25	100	Mean total flips	0.95
	50	100	Mean total flips	0.8
	100	100	Mean total flips	0.65
	150	100	Mean total flips	0.6
	200	50	Mean total flips	0.6
	250	30	Mean total flips	0.55

Table 5.2: Optimal levels of *noise* for WSAT/NOVELTY$^+$ on RANDOM-2+p-SAT for $p = 0, 0.3, 0.4, 0.5$ and 1.0

this low noise level is very detrimental to the harder instances (presumably because search becomes trapped due to the greediness) and results in very poor performance as measured by Mean total flips. Therefore we chose the noise level optimising Mean total flips where possible. Either way, there was some evidence that the two optimal noise levels for Median *mrl* and Mean total flips converge as n is increased and that the optimum in each case is rather broad, so that small changes do not affect the gross properties of the overall cost scaling.

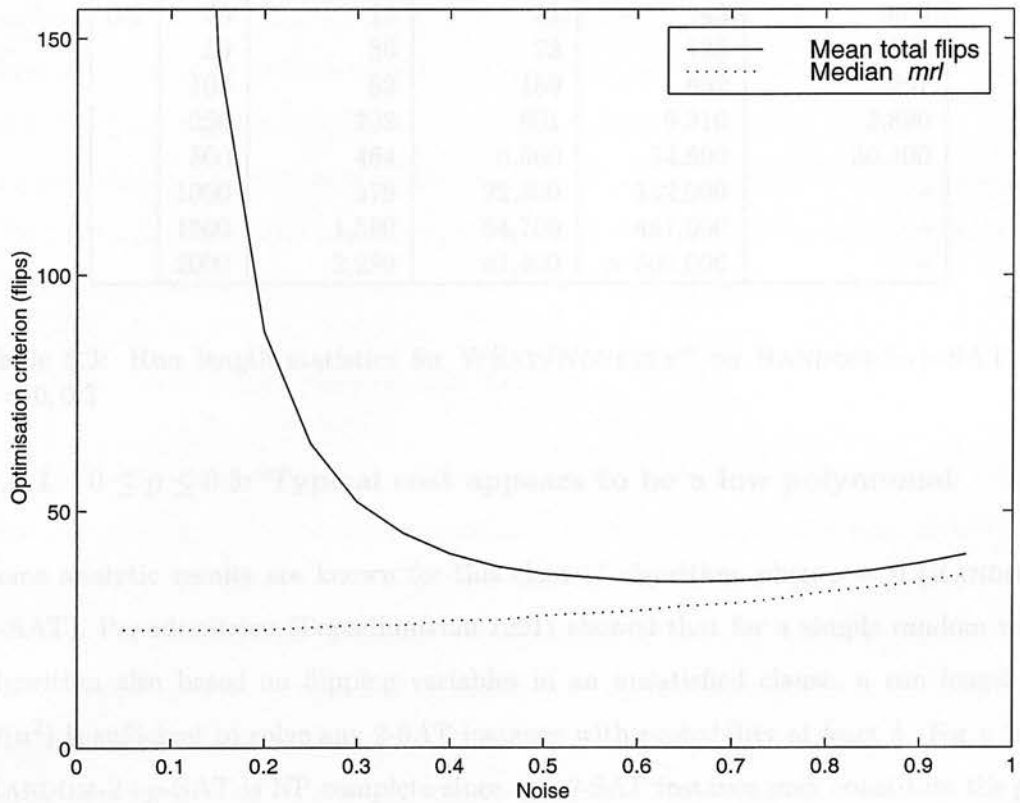

Figure 5.1: Dependence of median *mrl* and mean total flips on *noise* for $p = 0$, $n = 50$.

5.4 WSAT/NOVELTY$^+$ on RANDOM-2+p-SAT

In this section we present results of experimental tests of WSAT/NOVELTY+ on the instance collections described in Section 5.2.

p	n	Median *mrl* (3 s.f.)	95th perc. *mrl* (3 s.f.)	99th perc. *mrl* (3 s.f.)	Mean run length (3 s.f.)
0	25	15	24.5	30	17.8
	50	30	49.5	61.5	35.1
	100	60	95	115	68.3
	250	147	219	255	161
	500	294	401	463	311
	1000	562	746	862	591
	1500	832	1,060	1190	863
	2000	1,090	1,350	1,490	1,120
	3000	1,590	1,950	2,130	1,630
0.3	25	18	31	43	30.6
	50	36	73	138	101
	100	82	189	682	297
	250	208	801	9,310	3,890
	500	464	6,560	54,600	30,400
	1000	979	22,300	142,000	–
	1500	1,580	54,700	481,000	–
	2000	2,290	81,400	$> 500{,}000$	–

Table 5.3: Run length statistics for WSAT/NOVELTY$^+$ on RANDOM-2+p-SAT for $p = 0, 0.3$

5.4.1 $0 \leq p \leq 0.3$: Typical cost appears to be a low polynomial

Some analytic results are known for this class of algorithm when $p = 0$ (RANDOM-2-SAT). Papadimitriou (Papadimitriou 1991) showed that for a simple random walk algorithm also based on flipping variables in an unsatisfied clause, a run length of $O(n^2)$ is sufficient to solve any 2-SAT instance with probability at least $\frac{1}{2}$. For $p > 0$, RANDOM-2+p-SAT is NP-complete since any 3-SAT instance may constitute the pm 3-SAT clauses.

Table 5.3 gives experimental data for $p = 0$ and $p = 0.3$ on RANDOM-2+p-SAT. As well as Median *mrl*, we also give 95th and 99th percentiles of *mrl* and the mean run length (for those collections where there was no cut-off). The data for the typical cost for $p = 0, 0.3$ is consistent with a dependence on n which is $O(n^b)$ with b slightly less than 1 for $p = 0$ and slightly more than 1 for $p = 0.3$.

For $p = 0$ a slightly sublinear growth is also observed for the 95th and 99th percentiles and for the mean. This suggests that algorithms of this class may perform well on

2-SAT, or that instances requiring superlinear cost of the algorithms occur very rarely in this generation method. The sublinear scaling may be an artifact of some aspect of the methodology, for example the decreasing collection size.

For $p = 0.3$ the higher parts of the cost distribution apparently grow more quickly, although these high percentiles were not stable enough to infer a growth function. The more difficult instances appear to become so costly that they affect the growth of the mean run length. The presence of these instances caused large variances in mean run length which meant that more runs per instance were required for the smaller n. Possible ramifications of the superlinear mean are that either the generation method for $0 < p \leq 0.3$ requires superlinear cost in the average case or that WSAT/NOVELTY$^+$ is a poor choice of algorithm at this level of p if all satisfiable instances are to be solved. These hard but infrequent instances perhaps represent the hard "worst case" instances predicted by the NP-completeness of RANDOM-2+p-SAT when $p > 0$.

5.4.2 $p = 0.4$: Inconclusive results

The case when $p = 0.4$ is interesting as it is the current best lower bound for p_0. Also, the scaling of the complete algorithm TABLEAU has been observed to be linear at this level of p. So, if the change in complexity is bounded below by p_0 we might expect good (polynomial) performance from WSAT/NOVELTY$^+$ at this point. On the other hand even if the change in complexity is at p_0, local search might not be a good choice of algorithm for RANDOM-2+p-SAT, and might yield poor (superpolynomial) performance irrespective of the complexity of the problem. Figure 5.2 plots the typical (median) *mrl* for the different values of p as n is increased. Certainly the median *mrl* scales superlinearly at $p = 0.4$. Unfortunately, our results were inconclusive as to whether the median cost scales superpolynomially. On a log-log plot of the same data (Figure 5.3) there is a slight upward curve (suggesting superpolynomial scaling) between $n = 250$ and $n = 1500$, but this does not continue to $n = 2000$. Only with larger collection sizes, more runs per instance and a larger range of n will we be able to observe whether or not the scaling here is superpolynomial.

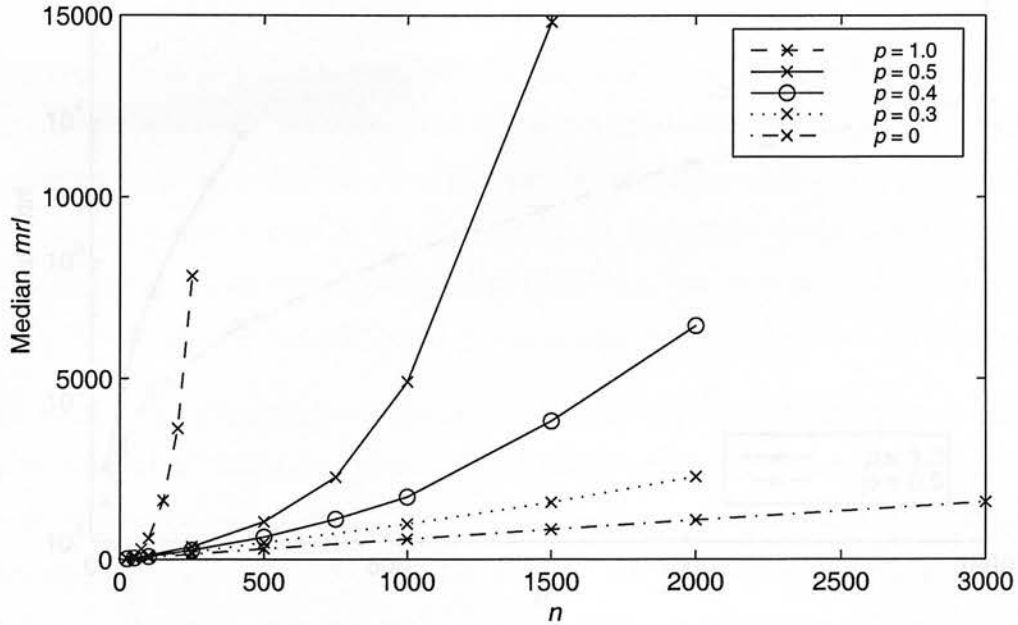

Figure 5.2: Typical cost for WSAT/NOVELTY$^+$ on RANDOM-2+p-SAT.

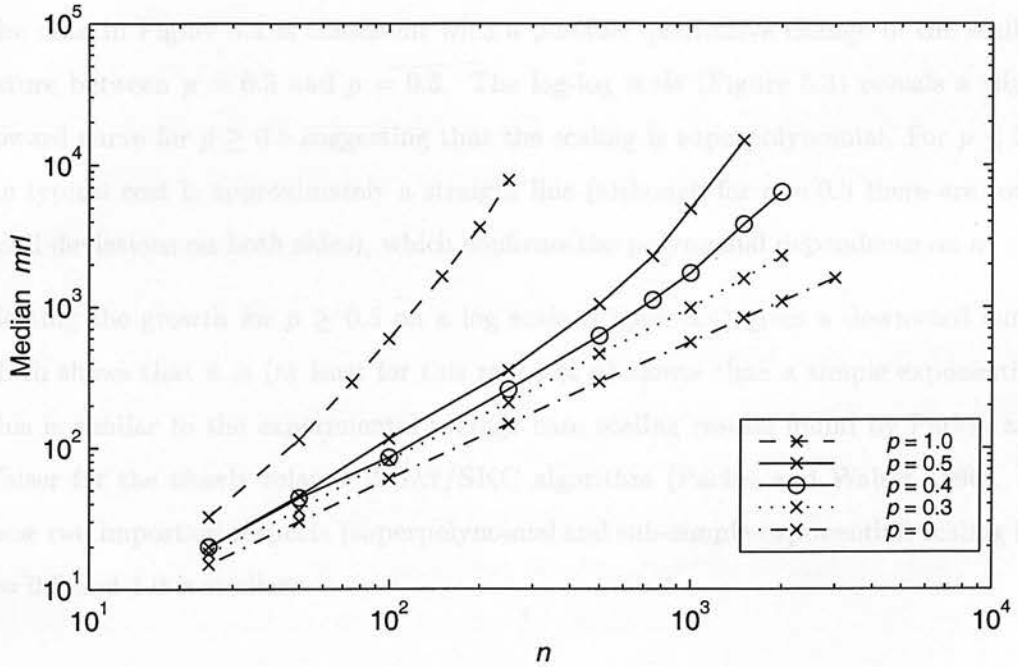

Figure 5.3: Typical cost for WSAT/NOVELTY$^+$ on RANDOM-2+p-SAT, log-log scale,

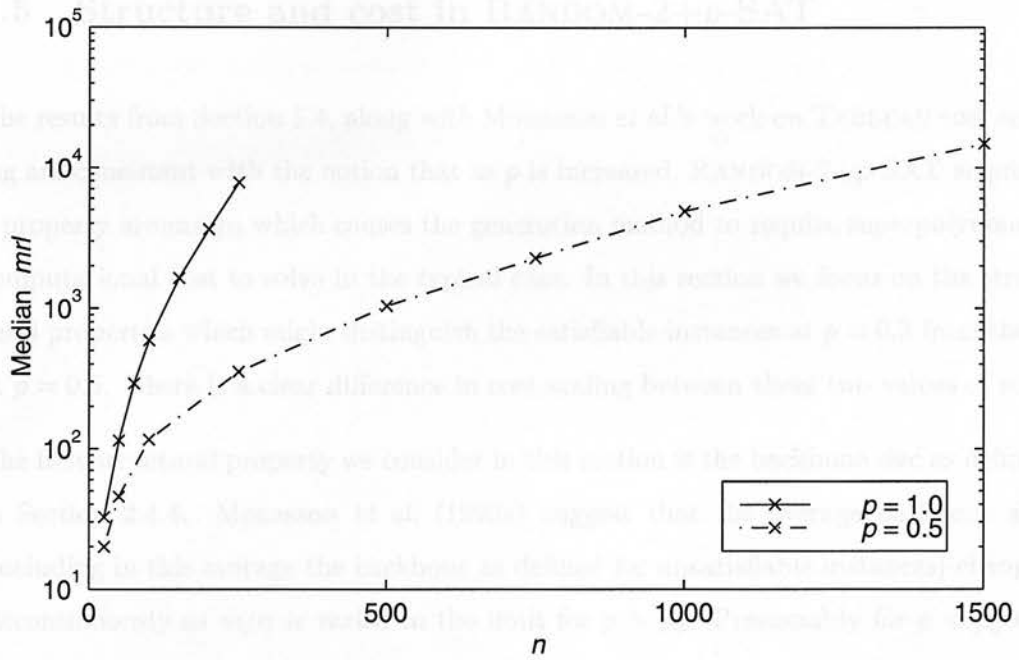

Figure 5.4: Typical cost for WSAT/NOVELTY$^+$ on RANDOM-2+p-SAT, log scale.

5.4.3 $0.5 \leq p \leq 1$: Typical cost appears superpolynomial

The data in Figure 5.2 is consistent with a possible qualitative change in the scaling nature between $p = 0.3$ and $p = 0.5$. The log-log scale (Figure 5.3) reveals a slight upward curve for $p \geq 0.5$ suggesting that the scaling is superpolynomial. For $p \leq 0.3$ the typical cost is approximately a straight line (although for $p = 0.3$ there are some small deviations on both sides), which confirms the polynomial dependence on n.

Plotting the growth for $p \geq 0.5$ on a log scale (Figure 5.4) gives a downward curve which shows that it is (at least for this range of n) slower than a simple exponential. This is similar to the experimental average case scaling results found by Parkes and Walser for the closely related WSAT/SKC algorithm (Parkes and Walser 1996). In these two important respects (superpolynomial and sub-simple-exponential) scaling for $p = 0.5$ and 1.0 is similar.

5.5 Structure and cost in RANDOM-2+p-SAT

The results from Section 5.4, along with Monasson *et al.*'s work on TABLEAU cost scaling are consistent with the notion that as p is increased, RANDOM-2+p-SAT acquires a property around p_0 which causes the generation method to require superpolynomial computational cost to solve in the typical case. In this section we focus on the structural properties which might distinguish the satisfiable instances at $p = 0.3$ from those at $p = 0.5$. There is a clear difference in cost scaling between these two values of p.

The first structural property we consider in this section is the backbone size as defined in Section 2.4.4. Monasson *et al.* (1999a) suggest that the average backbone size (including in this average the backbone as defined for unsatisfiable instances) changes discontinuously as m/n is varied in the limit for $p > p_0$. Presumably for $p < p_0$ the change is continuous, reflecting the continuous change in the satisfiability probability. If this carries over to the satisfiable phase, the possibly different nature of the backbone either side of p_0 could be one reason for the differences in local search cost.

We calculated backbone sizes for the $p = 0.3$ and $p = 0.5$ collections introduced in Table 5.1. The average *bsize* over each collection is plotted against n in Figure 5.5. The data shows that the mean *bsize* at $p = 0.5$ is slightly larger for this range of n than at $p = 0.3$. The median *bsize* over each collection is plotted in Figure 5.6. For the median, there is no consistent difference between the backbone sizes.

As demonstrated experimentally by Parkes (1997), at $p = 1$, the distribution of backbone sizes becomes bimodal for large n. As discussed in Section 2.4.4 RANDOM-3-SAT satisfiable threshold instances are a mixture of large backbone instances (with *bsize* around $0.9n$) and instances with much smaller backbones. Monasson *et al.* (1999a) also mention this and in addition report (although without giving data) that for $p = 0$ (RANDOM-2-SAT) this bimodal pattern does not occur: rather there is a single peak at a low value of *bsize*. This suggests that this bimodal feature may be connected to the computational hardness of the generation methods.

In our backbone size data, evidence of a qualitative difference between the distributions of *bsize* for $p = 0.3$ and $p = 0.5$ was inconclusive at best. Figure 5.7 gives histograms of our *bsize* data with n ranging up to 1500. In each case the backbone sizes have been

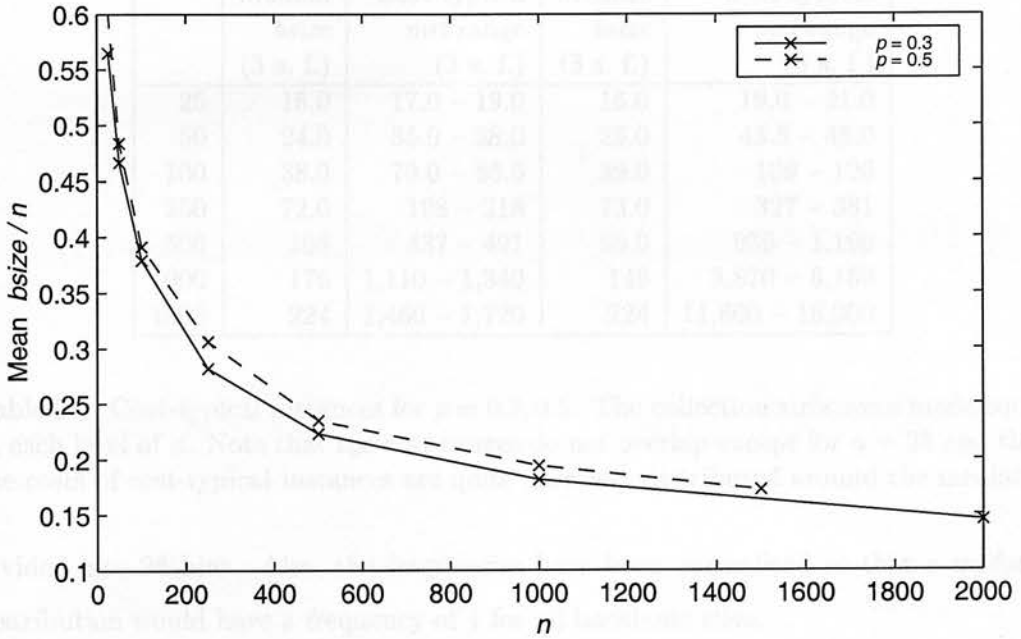

Figure 5.5: Average backbone size in RANDOM-2+p-SAT relative to n, $p = 0.3, 0.5$.

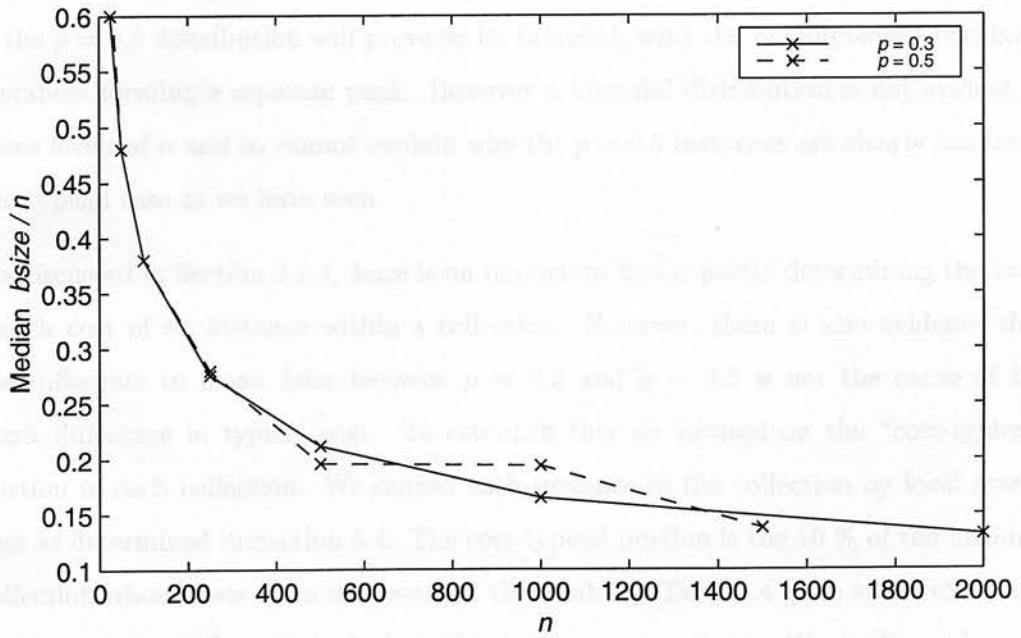

Figure 5.6: Median backbone size in RANDOM-2+p-SAT relative to n, $p = 0.3, 0.5$.

n	$p = 0.3$		$p = 0.5$	
	Median *bsize* (3 s. f.)	Cost-typical *mrl* range (3 s. f.)	Median *bsize* (3 s. f.)	Cost-typical *mrl* range (3 s. f.)
25	15.0	17.0 – 19.0	16.0	19.0 – 21.0
50	24.0	35.0 – 38.0	25.0	43.5 – 48.0
100	38.0	79.0 – 85.5	39.0	109 – 126
250	72.0	198 – 218	73.0	327 – 381
500	108	437 – 491	95.0	936 – 1,190
1000	176	1,110 – 1,340	149	3,870 – 6,160
1500	224	1,460 – 1,720	224	11,600 – 18,900

Table 5.4: Cost-typical instances for $p = 0.3, 0.5$. The collection sizes were made equal at each level of n. Note that the cost ranges do not overlap except for $n = 25$ and that the costs of cost-typical instances are quite narrowly distributed around the median.

divided into 26 bins. Also, the frequencies have been normalised so that a uniform distribution would have a frequency of 1 for all backbone sizes.

We note that the $p = 0.3$ distribution (on the left-hand side of the figure) remains clearly unimodal as n increases. The $p = 0.5$ distribution (on the right-hand side) is also unimodal, but becomes skewed, with a small number of medium-sized backbone instances (*bsize* around $0.5n$) persisting as n increases. It is possible that with larger n the $p = 0.5$ distribution will prove to be bimodal, with the medium-sized backbone instances forming a separate peak. However a bimodal distribution is not evident at these levels of n and so cannot explain why the $p = 0.5$ instances are clearly harder in the typical case as we have seen.

As discussed in Section 2.4.4, *bsize* is an important factor partly determining the local search cost of an instance within a collection. However, there is also evidence that the difference in mean *bsize* between $p = 0.3$ and $p = 0.5$ is not the cause of the stark difference in typical cost. To establish this we focused on the "cost-typical" portion of each collection. We ranked each instance in the collection by local search cost as determined in Section 5.4. The cost-typical portion is the 10 % of the instance collection whose costs lie in the centre of this ranking. Table 5.4 gives statistics on the backbone sizes of the cost-typical portion instances at various settings of n and p.

The data suggest that for larger n, where the cost differences are greater, the backbones of the $p = 0.3$ cost-typical instances are generally no smaller than those at $p = 0.5$

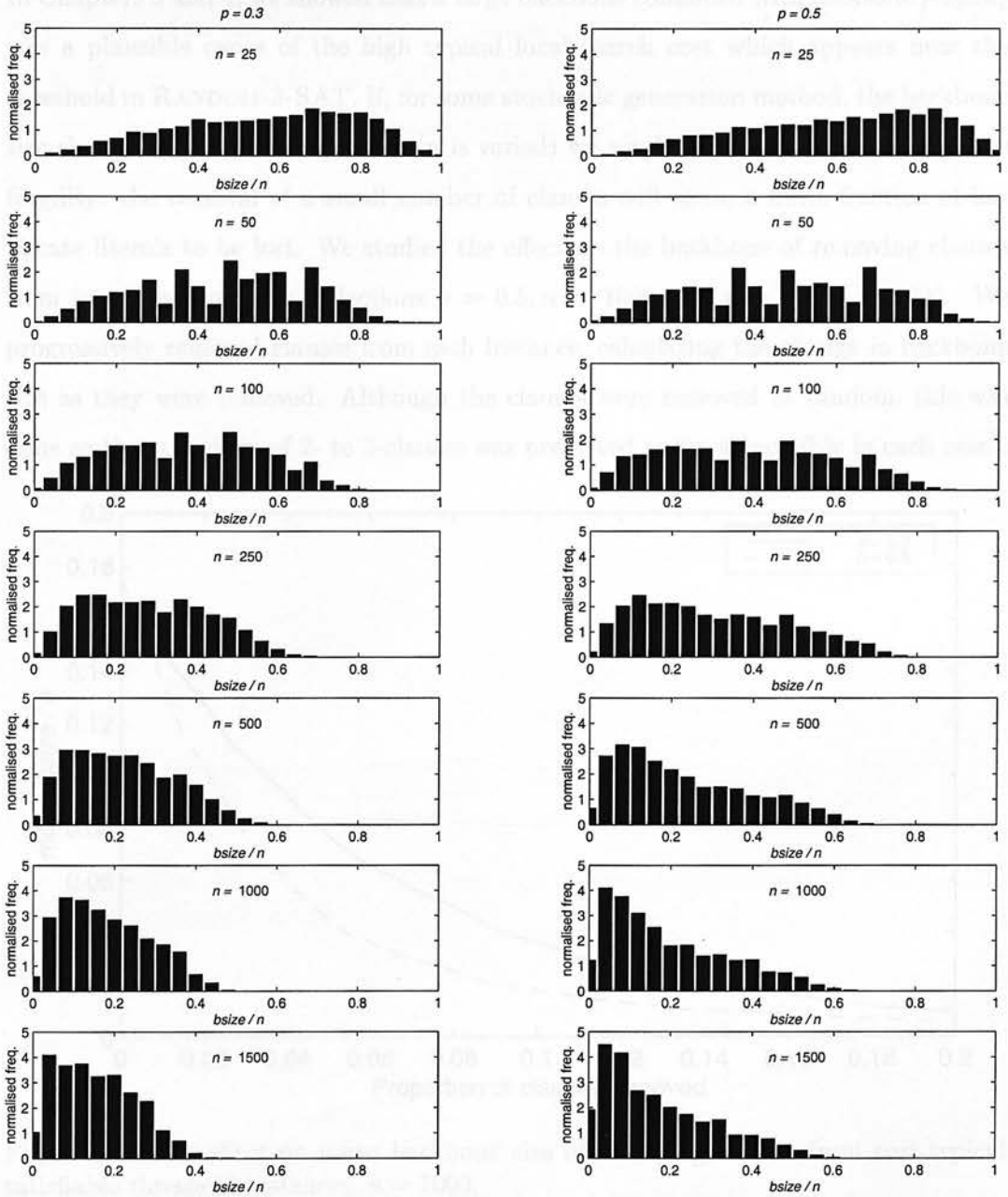

Figure 5.7: Distribution of backbone sizes in RANDOM-2+p-SAT relative to n for $p = 0.3, 0.5$.

even though they are much less costly. This would imply that backbone size is not the reason that typical cost grows so much more quickly for $p = 0.5$.

In Chapters 3 and 4, we showed that a large backbone combined with *backbone fragility* was a plausible cause of the high typical local search cost which appears near the threshold in RANDOM-3-SAT. If, for some stochastic generation method, the backbone size changes discontinuously (as m/n is varied) we would expect to observe backbone fragility: the removal of a small number of clauses will cause a finite fraction of implicate literals to be lost. We studied the effect on the backbone of removing clauses from instances from the collections $p = 0.5, n = 1000$ and $p = 0.3, n = 1000$. We progressively removed clauses from each instance, calculating the change in backbone size as they were removed. Although the clauses were removed at random, this was done so that the ratio of 2- to 3-clauses was preserved as far as possible in each case[3].

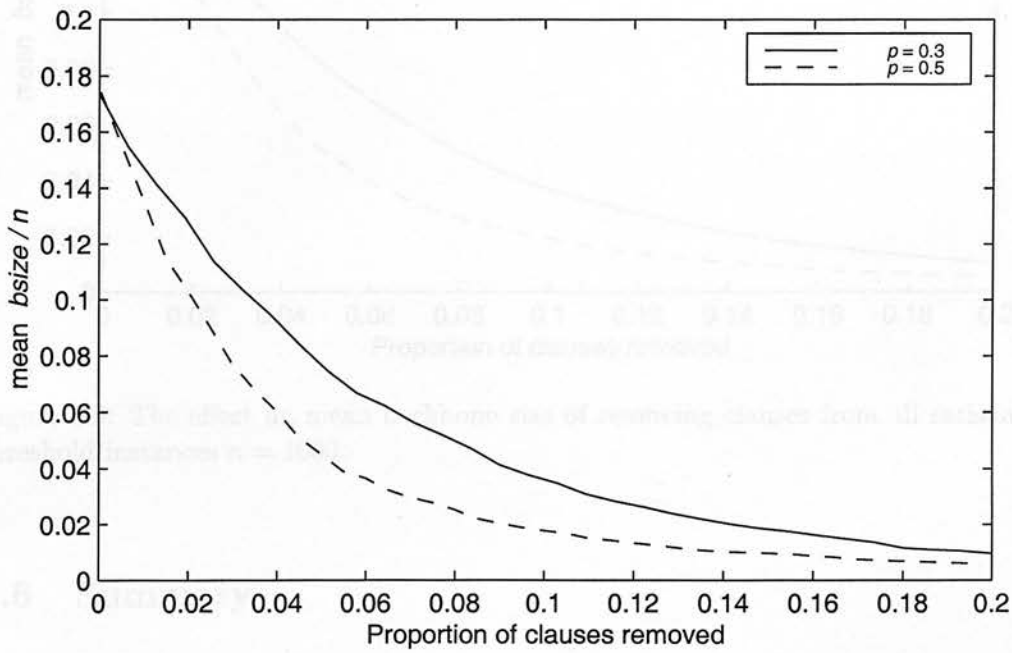

Figure 5.8: The effect on mean backbone size of removing clauses from cost-typical satisfiable threshold instances, $n = 1000$.

Figure 5.8 shows the effect on average backbone size of removing clauses from the cost-typical instances. The original average backbone size (seen at 0 clauses removed) is very

[3] This was done by removing a 3-clause if the current p was above the desired p and a 2-clause otherwise.

similar, but as we remove clauses the backbone decays much more sharply at $p = 0.5$ than at $p = 0.3$. The $p = 0.5$ instances are more backbone-fragile. This preliminary result combined with the work from Chapters 3 and 4 suggests that backbone fragility may also be the cause of the change in cost scaling between $p = 0.3$ and $p = 0.5$. The same effect was also seen with respect to the number (rather than the proportion) of clauses removed. An almost identical pattern is seen when data is averaged over all satisfiable instances rather than just the cost-typical ones (as shown in Figure 5.9).

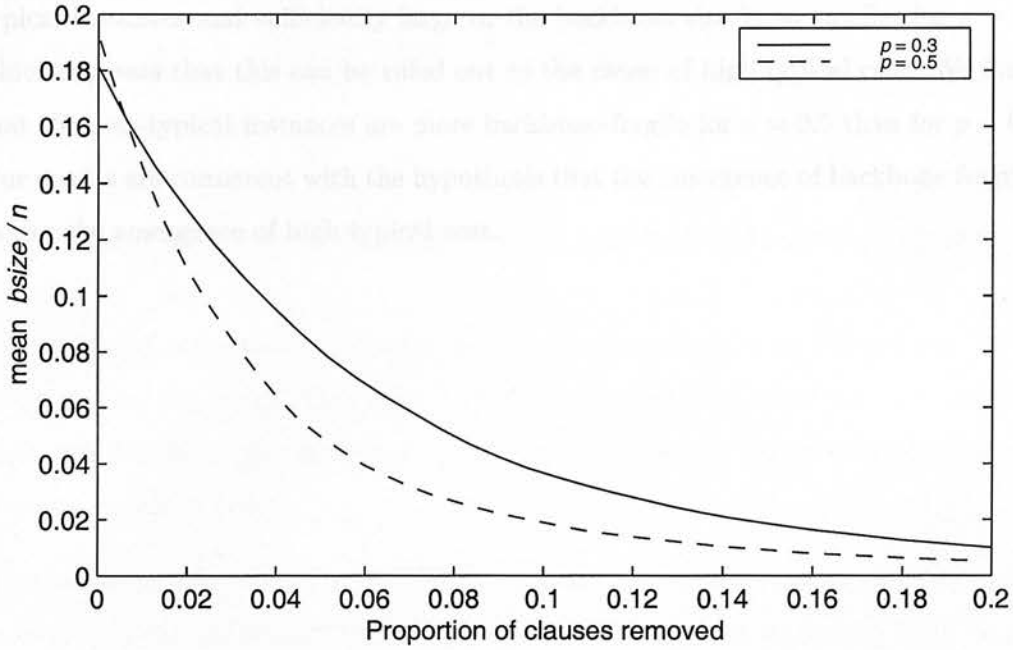

Figure 5.9: The effect on mean backbone size of removing clauses from all satisfiable threshold instances $n = 1000$.

5.6 Summary

We tested the local search algorithm WSAT/NOVELTY$^+$ at optimal noise levels on the RANDOM-2+p-SAT generation method. The experimental data suggested some significant conclusions. Typical WSAT/NOVELTY$^+$ cost appears to scale as a low polynomial up to $p = 0.3$. However, solving all instances using this algorithm appears to be costly for $p > 0$. For $p = 0.4$ the results were inconclusive: the scaling was superlinear, but we could not determine whether it was superpolynomial. For $p = 0.5$ typical cost appears to grow superpolynomially but more slowly than a simple exponential: in this

sense cost scaling for $p = 0.5$ is similar to that for $p = 1$. The apparent cross-algorithm nature of the change in typical cost scaling from low polynomial to superpolynomial suggests that some 'hardness property' of instances may become common enough to affect the median between 0.3 and 0.5. This may be near p_0 which is predicted at about 0.41.

We also reported on some experiments designed to identify the hardness property. The average backbone size is larger for $p = 0.5$ than it is for $p = 0.3$. However, for cost-typical instances and sufficiently large n, the backbone size is no smaller for $p = 0.3$ which suggests that this can be ruled out as the cause of high typical cost. We found that the cost-typical instances are more backbone-fragile for $p = 0.5$ than for $p = 0.3$. Our results are consistent with the hypothesis that the emergence of backbone fragility causes the emergence of high typical cost.

Chapter 6

Local Search on NOT-ALL-EQUAL 3-SAT

6.1 Introduction: When implicate literals are impossible

The evolution of the backbone as can be seen as the accumulation of implicate literals as constraints are added. Our hypothesis from Chapter 4 proposes that aspects of an instance's backbone (size and fragility) are important factors determining the difficulty of instances for WSAT.

However, in certain problems, literals are never implicates. For example, this was found to be true of the underconstrained RANDOM-3-SAT instances studied by Bayardo and Schrag (1996). Their empirical study found that implicate literals do not emerge until near the satisfiability transition. One other major reason for the lack of implicate literals is symmetry in the solution set which arises because of structural regularities in the instance. For example there has been a considerable amount of work encoding quasigroup existence problems from finite mathematics into SAT and solving them using SAT solvers e.g. Slaney *et al.* (1995); Zhang *et al.* (1996). These problems are often such that for a given problem, if there is a solution, then there are other essentially equivalent solutions to the problem which are obtained by symmetry. In the usual SAT encodings of these problems, although a proposition may be true in one solution, there will be another symmetric solution where the proposition is false. Such propositions and their negations cannot therefore be implicate literals of these SAT instances.

So, if the backbone cannot include these propositions, how can our hypothesis, which is based on the backbone, possibly generalise? The approach we take in this chapter is to broaden the definition of the backbone. Rather than being based simply on literals, which in certain SAT instance classes cannot be implicates, we allow the backbone to be based on other kinds of constraints which can be implicates.

In this chapter we generalise our hypothesis to a problem which cannot have implicate literals. The problem is NOT-ALL-EQUAL 3-SAT, in which implicate literals are impossible, but which is otherwise similar in many respects to 3-SAT.

The remainder of this chapter is as follows. In Section 6.2 we introduce the NOT-ALL-EQUAL 3-SAT problem and the "double clause" encoding which represents a NOT-ALL-EQUAL 3-SAT instance as a 3-SAT instance. We also describe how the backbone can be defined in the absence of implicate literals. In Section 6.3 we investigate the behaviour of WSAT/SKC on double clause encodings of random NOT-ALL-EQUAL 3-SAT instances and note certain patterns. In order to explain this behaviour, in Section 6.4 we formulate an analogous hypothesis to that put forward in Chapter 4. This is based again on backbone fragility, but this time in terms of the new definition of the backbone. We then test one important prediction of this hypothesis. Finally, Section 6.5 summarises the chapter.

6.2 NOT-ALL-EQUAL 3-SAT

In this chapter we study the NP-complete NOT-ALL-EQUAL (NAE) 3-SAT problem, which is given by Garey and Johnson (1979) as follows:

Instance A set V of variables and a collection C of clauses over V such that each clause has exactly 3 literals.

Question Is there a truth assignment for V such that each clause has at least one true and at least one false literal?

Note that the only difference with 3-SAT is that as well as one true literal, there must also be one false literal in every clause. Hence, moving from 3-SAT to NAE

3-SAT we are changing simply the decision question. We will describe a CNF as being *NAE satisfiable* if the answer to the NAE 3-SAT decision question is "yes" and *NAE unsatisfiable* otherwise. Similarly we will talk of clauses being *NAE (un)satisfied* under assignments and assignments *NAE satisfying* clauses. Assignments which prove an instance to be NAE satisfiable are *NAE solutions* of the instance. Two instances are *NAE equivalent* when every assignment is an NAE solution of one iff it is an NAE solution of the other.

Recall from Section 2.4.4 that in SAT an implicate of an instance C is defined as any clause c such that $C \rightarrow c$ is valid. Equivalently, c is an implicate of C when any assignment which is a solution of C also satisfies c. Similarly we may define *NAE implicates* as follows. The clause c is an NAE implicate of the instance C when any assignment which is an NAE solution of C also NAE satisfies c.

The NAE 3-SAT problem was suggested by Culberson (1999) as a test case for generalisation of our hypothesis for the reason that it is similar to 3-SAT except for the lack of implicate literals.

6.2.1 The "double clause" encoding of NOT-ALL-EQUAL 3-SAT

We now show how an instance of NAE 3-SAT may be very simply encoded as an instance of 3-SAT. The aim is to take an arbitrary instance of NAE 3-SAT C and construct an instance of 3-SAT C' such that C' is satisfiable iff C is NAE satisfiable. There are different possible encodings, but the double clause encoding has a particularly useful property that each solution of the encoding C' is an NAE solution of the NAE 3-SAT instance C.

Suppose C consists of clauses $c_1, \ldots, c_m$. Each clause c_i in C is a disjunction of 3 literals $l_{i1} \vee l_{i2} \vee l_{i3}$. Here, $\mathrm{neg}(l)$ denotes the negation of literal l as defined in Section 2.5.1. We start with C' as the empty formula. Then for each clause c_i in C we add two clauses c_{i1} and c_{i2} to C': $c_{i1} = l_{i1} \vee l_{i2} \vee l_{i3}$ and $c_{i2} = \mathrm{neg}(l_{i1}) \vee \mathrm{neg}(l_{i2}) \vee \mathrm{neg}(l_{i3})$.

For any assignment $\mathcal{T}$, note that c_i is NAE satisfied under $\mathcal{T}$ iff both c_{i1} and c_{i2} are satisfied under $\mathcal{T}$. It follows not only that C' is satisfiable exactly when C is NAE satisfiable, but also that $\mathcal{T}$ is a solution to C' exactly when $\mathcal{T}$ is an NAE solution to

C.

6.2.2 Implicates and the backbone in NOT-ALL-EQUAL 3-SAT

Observe that literals (clauses of length 1) can never be NAE satisfied. No matter what value is assigned to the variable mentioned in the literal, it is never the case that the literal contains both a true and a false literal. This means that if an instance C is NAE satisfiable, any literal c cannot be an NAE implicate of C.

However, there are some simple properties which may be implicates of NAE satisfiable instances, for example 2-clauses. As we shall see, these can be used to define a backbone. This method of generalising the backbone was identified first in graph colouring by Joe Culberson and Ian Gent (Culberson and Gent 2000).

In NAE SAT, 2-clauses are essentially disequality relations between literals. The meaning of the clause $l_1 \vee l_2$ in the NAE SAT context is that the literals l_1 and l_2 must have different values. The clause is NAE satisfied precisely by assignments under which l_1 takes the opposite value to l_2. The clause $l_1 \vee \mathrm{neg}(l_2)$ is the negation of this: it asserts that l_1 and l_2 have the same value as it is NAE satisfied precisely when $l_1 \vee l_2$ is NAE unsatisfied.

As an example of how a 2-clause can be an NAE implicate of an NAE satisfiable instance, consider the instance of NAE 3-SAT below.

$$x \vee y \vee z$$
$$x \vee y \vee \neg z$$

This is a NAE satisfiable instance of NAE 3-SAT. An example NAE solution to this instance sets x true, y false and z true. The clause $x \vee y$ (which in the NAE context asserts that x and y must take opposite truth values) is an NAE implicate of the above instance. To demonstrate this, we add the negation $x \vee \neg y$ to the formula and show that this results in NAE unsatisfiability. Adding this clause gives:

$$x \vee y \vee z$$
$$x \vee y \vee \neg z$$

$$x \vee \neg y$$

This is now NAE unsatisfiable. To see this, observe that due to the third clause, we must set x and y to the same truth value, t_1 say. Let t_2 be the opposite truth value. By the first clause, the three variables must not have the same truth value and so z must be set to the value opposite to the one given to x and y, so we set z to t_2. But this then contradicts the second clause which, given that x and y are both set to t_1, effectively requires z also to be set to t_1. Since adding $x \vee \neg y$ causes NAE unsatisfiability, the clause $x \vee y$ must be an NAE implicate of the original instance.

This example also demonstrates that testing whether a disequality between literals is an NAE implicate of an NAE SAT instance is straightforward: we add the clause requiring equality between the literals to the formula and test whether it is then NAE unsatisfiable.

What is less clear, given the set of NAE implicate disequalities of the instance, is how to define the backbone size. One possibility is simply the number of possible NAE implicate disequalities which are actual NAE implicates. This is analogous (although it is not the only analogy) to the backbone size in SAT, which is the number of possible implicate literals which are actual implicates. One problem with this is that in NAE 3-SAT, as the set of NAE implicates evolves when clauses are added, many NAE implicate disequalities appear due to transitivity. For example if $l_1 \vee l_2$ and $l_2 \vee l_3$ are NAE implicates, then $l_1 \vee \text{neg}(l_3)$ must also be an NAE implicate by transitivity. Using this interpretation of backbone size may give us a view of the evolution of the backbone which is heavily influenced by transitivity.

An alternative which we use was advocated by Culberson and Gent (2000) in the context of graph colouring. This avoids the effect on the backbone of transitivity. Suppose the clause $l_1 \vee l_2$ is an NAE implicate of an NAE 3-SAT instance. This dictates that l_1 and l_2 must take different values, or that l_1 always equals $\text{neg}(l_2)$. Given this NAE implicate, we can eliminate l_2 and its negation entirely from the instance. We replace all occurrences of l_2 with $\text{neg}(l_1)$ and all occurrences of $\text{neg}(l_2)$ with l_1. The resulting formula must be NAE equivalent to the original and must mention one fewer variables. We continue eliminating variables in this way according to disequality

NAE implicates until no more variables can be eliminated. The backbone size is then defined as the number of variables which can be eliminated because of NAE implicate disequalities. This is also a valid analogy with the backbone in SAT, which can be seen as the number of variables which can be eliminated because of implicate literals. The backbone size ranges from 0 to $n-1$ as the last variable can never be eliminated.

The backbone size according to this method is not affected by transitivity. If $l_1 \vee l_2$ is an NAE implicate, we must eliminate either l_1 and its negation or l_2 and its negation. Either way, the clauses $l_2 \vee l_3$ and $l_1 \vee \text{neg}(l_3)$ are then NAE equivalent and so it is not the case that if one of these is an NAE implicate, we also have a third distinct NAE implicate because of transitivity.

The backbone size defined as the maximal number of variables which can be eliminated due to NAE implicates is independent of the order in which the variable eliminations are performed. The NAE implicates are equality and disequality relations between variables which respect the axioms of equality and disequality given that the instance is satisfiable. The variables therefore form subsets such that between any two variables in different subsets, there are no equality or disequality relations. Each subset is divided into two halves, each variable is constrained to be equal to every variable in the same half and constrained to be not equal to every variable in the other half. Figure 6.1 illustrates this situation. The nodes are variables and the edges denote equality and disequality relations as labelled. The dashed lines indicate subsets divided into halves.

We only merge literals whose variables appear in the same subset, so the elimination procedure stops when each subset contains exactly one variable. Whichever order the variables are eliminated in, the number eliminated is the same: the total number of variables minus the number of subsets.

6.2.3 Determining the backbone in NOT-ALL-EQUAL SAT

We now give some notes on the procedure we developed to determine the backbone of an NAE SAT instance. As in SAT, the basic idea is to begin with a set of possible NAE implicates and attempt to confirm or eliminate each of these. There are $n(n-1)$ possible disequality NAE implicates, so it is important for the sake of efficiency to

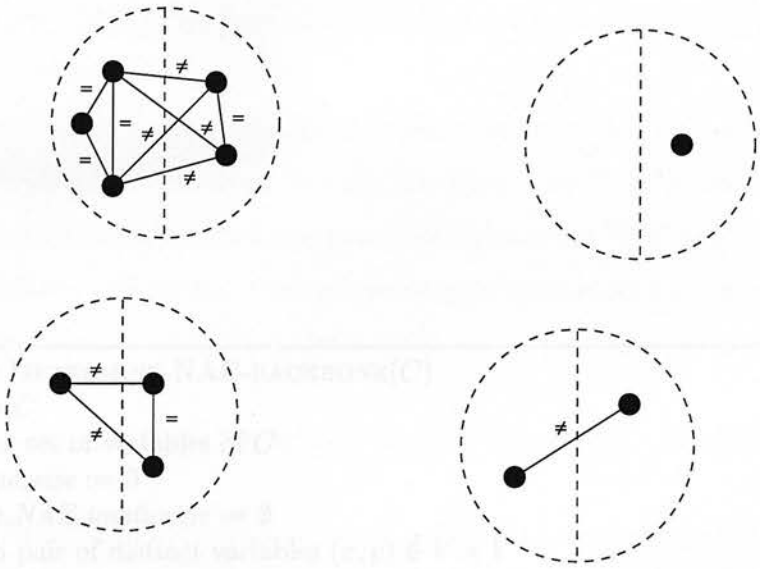

Figure 6.1: Equality and disequality relations between variables in an NAE SAT instance.

reduce this where possible when determining the backbone of an NAE SAT instance. Our method uses two principles to do this:

1. Once a disequality NAE implicate is established, the associated variable elimination is carried out immediately and all future searches use instances with this variable eliminated. All possible NAE implicates which mention the eliminated variable are disregarded, since these are NAE equivalent to other possible NAE implicates.

2. If an NAE solution to the instance is found during an attempt to establish an NAE implicate, this NAE solution may be used to eliminate certain disequality NAE implicates. For example if the NAE solution makes l_1 and l_2 equal, the possible NAE implicate $l_1 \vee l_2$ may be eliminated.

Pseudocode for the DETERMINE-NAE-BACKBONE algorithm is given in Figure 6.2. It assumes we have a procedure NAE-SOLUTION which takes an NAE instance and returns a NAE solution cylinder if there is one. In practice, this subroutine runs DPLL-SOLUTION on the double clause encoding of the NAE SAT instance.

```
procedure DETERMINE-NAE-BACKBONE(C)
    C_temp = C
    V := the set of variables of C
    Backbone_size := 0
    Possible_NAE_implicates := ∅
    for each pair of distinct variables (x, y) ∈ V × V
        Possible_NAE_implicates := Possible_NAE_implicates ∪ {x ∨ y, x ∨ ¬y}
    end for
    while Possible_NAE_implicates ≠ ∅
        Current_NAE_implicate := a member of Possible_NAE_implicates
        Remove Current_NAE_implicate from Possible_NAE_implicates
        C_test := C_temp ∧ the negation of Current_NAE_implicate
        if NAE-SOLUTION(C_test) returns a solution cylinder T_cyl
            for each clause Clause ∈ Possible_NAE_implicates
                if there is some extension T_sol of T_cyl such that Clause is
                                                NAE unsatisfied under T_sol
                    Remove Clause from Possible_NAE_implicates
                end if
            end for
        else
            l_1 := first literal of Current_NAE_implicate
            l_2 := second literal of Current_NAE_implicate
            C_temp := C_temp with l_2 substituted by neg(l_1) and neg(l_2) substituted by l_1
            Backbone_size := Backbone_size + 1
        end if
    end while
    return Backbone_size
```

Figure 6.2: The DETERMINE-NAE-BACKBONE procedure.

6.3 WSAT/SKC on NOT-ALL-EQUAL RANDOM-3-SAT

The remainder of this chapter attempts to extend our hypothesis about WSAT/SKC cost to the NAE 3-SAT problem. We use RANDOM-3-SAT instances generated exactly as in previous chapters, but interpreted as instances of NAE 3-SAT. We refer to these as NAE RANDOM-3-SAT. We tried where possible to reproduce the experimental conditions which were used in Chapters 3 and 4.

6.3.1 The satisfiability threshold in NOT-ALL-EQUAL RANDOM-3-SAT

Our hypothesis relates to satisfiable random instances at the threshold of satisfiability and so the first task was to establish the location of the NAE satisfiability threshold. An upper bound which in other problems has been found to be close to the true threshold is established by considering the expected number of solutions (Gent *et al.* 1996).

Assuming that there is an asymptotic threshold in NAE satisfiability, setting the expected number of NAE solutions equal to 1 gives an upper bound on the location of the asymptotic threshold. In NAE 3-SAT each of the m clauses is NAE satisfied by three quarters of the 2^n assignments so we have:

$$\text{Expected no. of NAE solutions} = 1$$

$$\text{So we have} \qquad 2^n(\tfrac{3}{4})^m = 1$$

Re-arranged, this gives:

$$\frac{m}{n} = \frac{\log_2(\frac{1}{2})}{\log_2(\frac{3}{4})} = 2.4096$$

To experimentally determine the threshold location we set $n = 100$ and varied $\frac{m}{n}$ between 1 and 3 in steps of 0.01 (i.e. one clause). We used 5000 instances at each point to give an estimate of the probability of NAE satisfiability. Figure 6.3 shows this estimate as a function of $\frac{m}{n}$. At this level of n, the whole number of clauses nearest to the point of 50% NAE satisfiability was 205, where 48.98% of instances were NAE satisfiable.

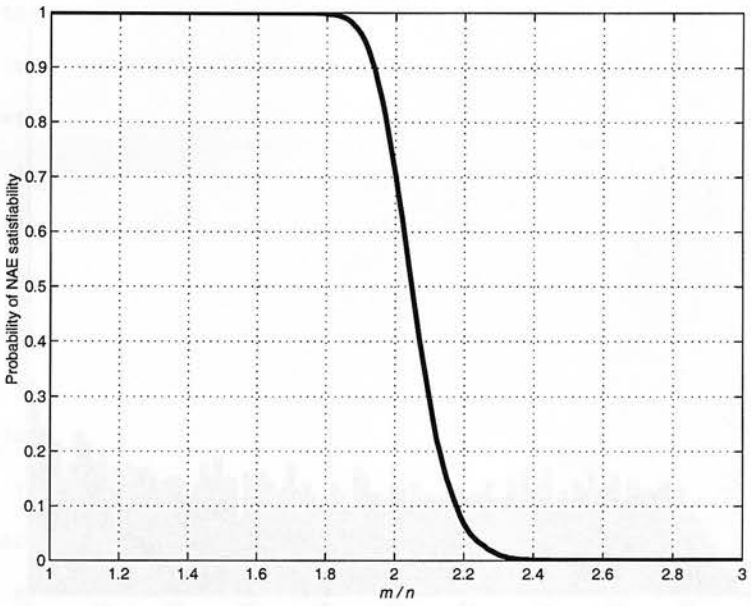

Figure 6.3: Probability of NAE satisfiability, $n = 100$.

6.3.2 The evolution of backbone size in NOT-ALL-EQUAL RANDOM-3-SAT

We also needed a detailed picture of how backbone size evolves near the threshold in NOT-ALL-EQUAL RANDOM-3-SAT, so we conducted some experiments to determine this. Figures 6.4, 6.5 and 6.6 show the evolution of backbone size as m/n is increased through the point of 50% solvability (2.05). We set m/n at 1.95, 2.05 and 2.15. We used 10,000, 50,000 and 50,000 instances respectively. The large backbone sizes at the higher levels of m/n meant that more variables and possible implicates could be eliminated and so more data could be collected. The histograms have been normalised so that a uniform distribution would be level at 1.0.

There are some similarities with the analogous sequence of distributions in RANDOM-3-SAT (see Figure 2.10) and also differences. Remarkably, in both sequences a mode of the distribution appears near $0.9n$. However, in NAE RANDOM-3-SAT this group of instances is more numerous at the point of 50% solvability, where there are very few small-backbone instances. In NAE RANDOM-3-SAT, at $n = 100$, the "leap" in backbone size occurs at a more underconstrained point relative to the threshold than in RANDOM-3-SAT. There seems to be no evidence of a weak secondary peak such as

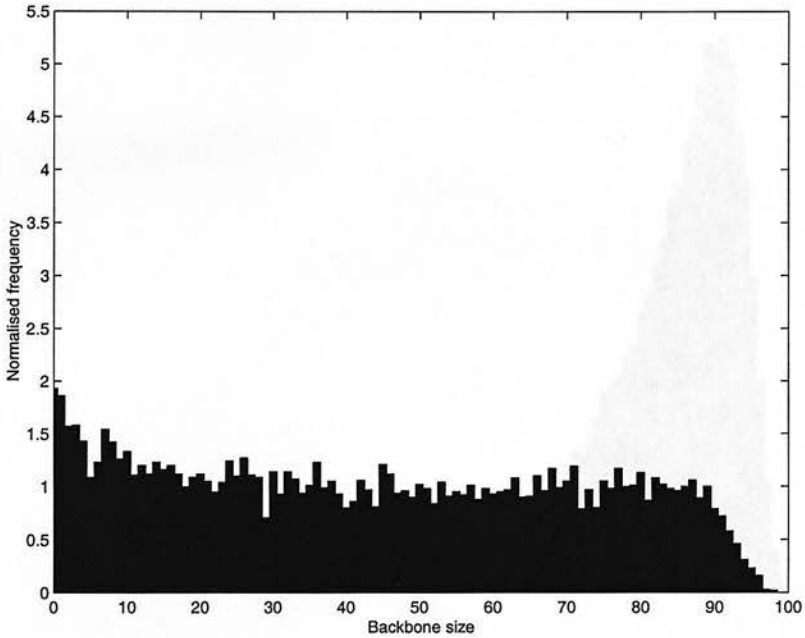

Figure 6.4: Backbone size distribution in NAE RANDOM-3-SAT, $n = 100$, $m = 195$.

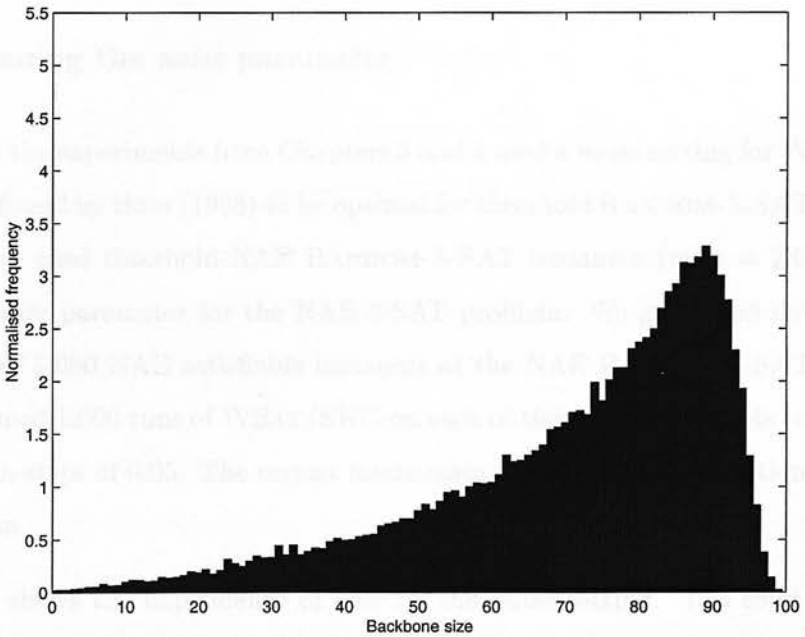

Figure 6.5: Backbone size distribution in NAE RANDOM-3-SAT, $n = 100$, $m = 205$.

that observed by Parkes (1997), so that could be peculiar to RANDOM-3-SAT, or it may only emerge at higher levels of n.

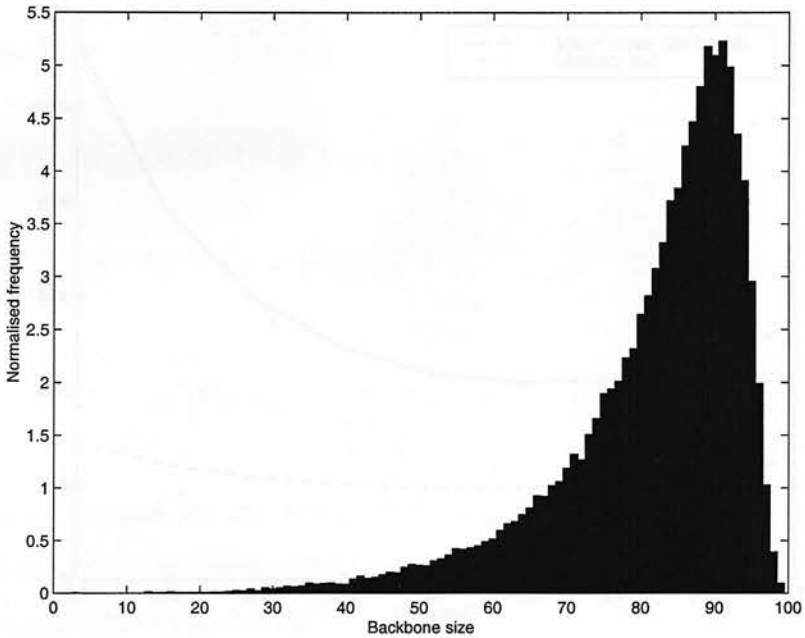

Figure 6.6: Backbone size distribution in NAE RANDOM-3-SAT, $n = 100$, $m = 215$.

6.3.3 Tuning the *noise* parameter

Recall that the experiments from Chapters 3 and 4 used a *noise* setting for WSAT/SKC which was found by Hoos (1998) to be optimal for threshold RANDOM-3-SAT instances. Similarly we used threshold NAE RANDOM-3-SAT instances ($m/n = 2.05$) to optimise the *noise* parameter for the NAE 3-SAT problem. We generated double clause encodings of 5,000 NAE satisfiable instances at the NAE RANDOM-3-SAT threshold and performed 1,000 runs of WSAT/SKC on each of these, at *noise* levels varying from 0.3 to 0.7 in steps of 0.05. The restart mechanism was not used. A solution was found in every run.

Figure 6.7 shows the dependence of cost on the *noise* setting. The solid line is the mean over all 5,000 instances of the mean run length for each instance and the dashed line is the median over all 5,000 instances of the *mrl* for each instance. In both cases the optimum is at 0.55. Interestingly this is the same as the threshold optimum in RANDOM-3-SAT, which may indicate a degree of similarity between the WSAT/SKC search spaces induced by the two problems.

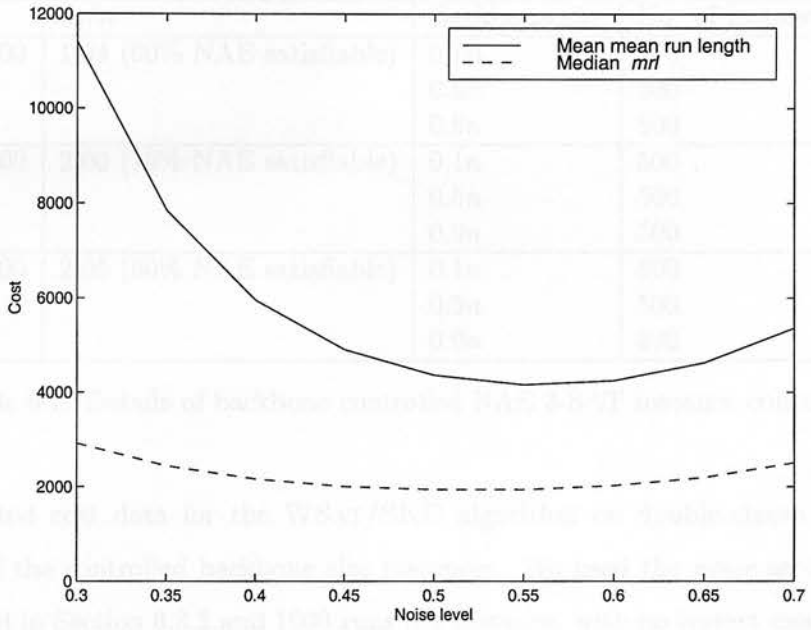

Figure 6.7: Dependence of WSAT/SKC cost on the *noise* parameter setting for double clause encodings of NAE satisfiable threshold NAE RANDOM-3-SAT instances.

6.3.4 Cost with backbone size controlled

For the remainder of this chapter we study NAE RANDOM-3-SAT instances with backbone size controlled, following our methodology from earlier chapters. The results from Section 6.3.2 show that the backbone transition in this problem occurs at a more underconstrained location than in RANDOM-3-SAT. So, in the overconstrained region of NAE RANDOM-3-SAT, small-backbone instances are extremely rare. In addition, because of the $n(n-1)$ possible implicates, determining the backbone in NAE RANDOM-3-SAT is considerably more computationally expensive than in RANDOM-3-SAT. However we wanted to study the properties of small, medium and large backbone instances as m/n is varied near the threshold. This made necessary certain compromises. Firstly we reduced the sample size to 500 instances. Secondly we studied a region of the control parameter range which is slightly more underconstrained than that which was studied in RANDOM-3-SAT, since we needed a region where a range of backbone sizes were frequent. Thirdly we studied only 3 values of m/n rather than the 8 studied in RANDOM-3-SAT. Details of the backbone controlled instances used are given in Table 6.1.

n	m/n	Backbone size	No. of instances
100	1.94 (90% NAE satisfiable)	$0.1n$	500
		$0.5n$	500
		$0.9n$	500
100	2.00 (70% NAE satisfiable)	$0.1n$	500
		$0.5n$	500
		$0.9n$	500
100	2.05 (50% NAE satisfiable)	$0.1n$	500
		$0.5n$	500
		$0.9n$	500

Table 6.1: Details of backbone controlled NAE 3-SAT instance collections

We collected cost data for the WSAT/SKC algorithm on double-clause 3-SAT encodings of the controlled backbone size instances. We used the *noise* setting of 0.55 established in Section 6.3.3 and 1000 runs per instance, with no restart mechanism. A solution was found in every run. Our cost measure for each instance was the *mrl* of these 1000 runs. Figure 6.8 shows the dependence of median *mrl* on m/n with the backbone size controlled at different values. The bars at each point give the interquartile range.

We compare this data with that presented in Figure 3.3 to determine which patterns of cost behaviour carry over to NAE RANDOM-3-SAT. The overall range of cost is similar, suggesting that the double clause encodings of these NAE RANDOM-3-SAT instances are neither easier nor harder for WSAT/SKC than their RANDOM-3-SAT counterparts. The data is also consistent with an exponential decay in cost for controlled backbone size but there are clearly too few points to extrapolate this. We observe a similar positive effect of backbone size: in this case the effect may be larger as there is a greater difference between the cost of medium and large backbone sizes.

We observe a large variation in cost for large backbone size instances, but the variation is not quite so large for small and medium backbone sizes. Overall, the cost patterns are very similar to those in RANDOM-3-SAT.

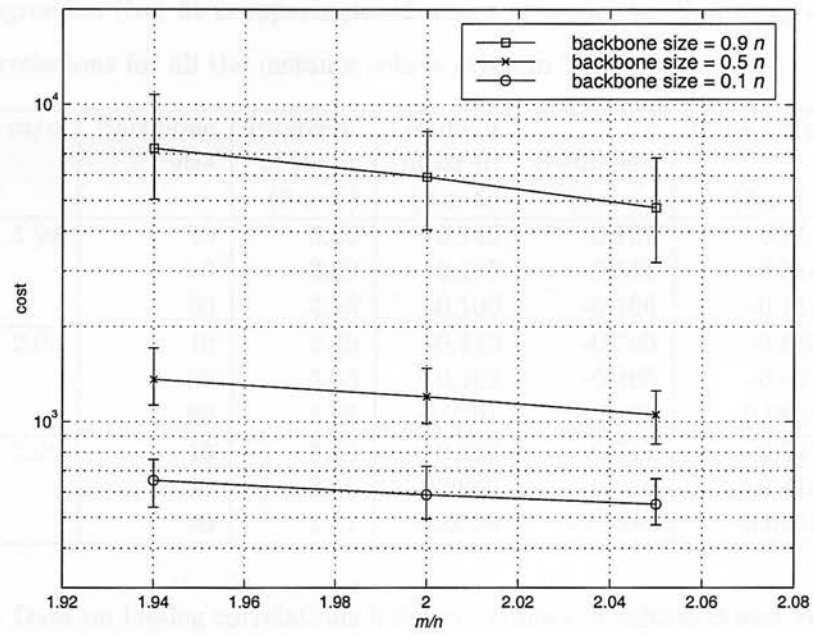

Figure 6.8: Dependence of WSAT/SKC cost (median *mrl*) on m/n for instances with backbone size controlled at different values. The instances are double clause 3-SAT encodings of the backbone controlled NAE RANDOM-3-SAT instances. The length of the bars is the interquartile range (25th – 75th percentiles).

6.4 Explaining cost behaviour in NAE RANDOM-3-SAT

Given the similar cost patterns in NAE RANDOM-3-SAT we now consider whether these can be explained using a similar explanation to that given in Chapters 3 and 4 for RANDOM-3-SAT.

6.4.1 Cost and the number of solutions

Recall that for the RANDOM-3-SAT problem we investigated in Chapter 3 the extent to which the number of solutions accounts for the unexplained variation in cost. The number of solutions accounted for most of the variation for small backbone instances, but with increasing backbone size, the number of solutions accounted for less variation. Here we used our solution-counting software on the double clause 3-SAT encodings of the NAE SAT instances, since these have the same number of solutions as the NAE SAT instances themselves. Figure 6.9 shows three log-log plots of the number of solutions against cost where m/n is 2.05 and backbone size is $0.1n$, $0.5n$ and $0.9n$. A linear least

squares regression (*lsr*) fit is superimposed where appropriate. Summary data on the log-log correlations for all the instance sets is given in Table 6.2.

m/n	Backbone size	Intercept of *lsr* fit (3 s. f.)	Gradient of *lsr* fit (3 s. f.)	r coefficient (3 s. f.)	Rank corr. (3 s. f.)
1.94	10	3.66	-0.149	-0.737	-0.719
	50	3.69	-0.117	-0.531	-0.514
	90	4.13	-0.100	-0.106	-0.112
2.00	10	3.49	-0.130	-0.700	-0.680
	50	3.55	-0.102	-0.465	-0.451
	90	3.86	-0.0391	-0.0461	-0.0608
2.05	10	3.44	-0.128	-0.717	-0.693
	50	3.45	-0.0949	-0.459	-0.418
	90	3.81	-0.0538	-0.0584	-0.0553

Table 6.2: Data on log-log correlations between number of solutions and WSAT/SKC cost for NAE RANDOM-3-SAT backbone controlled instances.

The pattern is the same in important ways to that in RANDOM-3-SAT. The number of solutions accounts for much of the cost variation for small backbone instances but much less for large backbone instances. There is no clear change in the r values as m/n is varied.

6.4.2 Cost and backbone fragility in NAE RANDOM-3-SAT

We now examine the extent to which the variation in cost for large backbone instances and the decay in cost for backbone controlled instances near the threshold in NAE RANDOM-3-SAT can be explained in terms of backbone fragility.

Recall that in Chapter 4 we measured backbone robustness in RANDOM-3-SAT using *tbbr*. Preliminary experiments with the NAE analogue of the *tbbr* metric indicated that establishing this with any accuracy on our NAE 3-SAT instances was too computationally expensive to be practical. This is because of the very large number of possible implicate disequalities in NAE 3-SAT. We therefore switched to an alternate measure of backbone fragility called *fixed fraction removal based backbone fragility (ffrbbf)*. This was a feasible compromise, but measurements were still very computationally expensive.

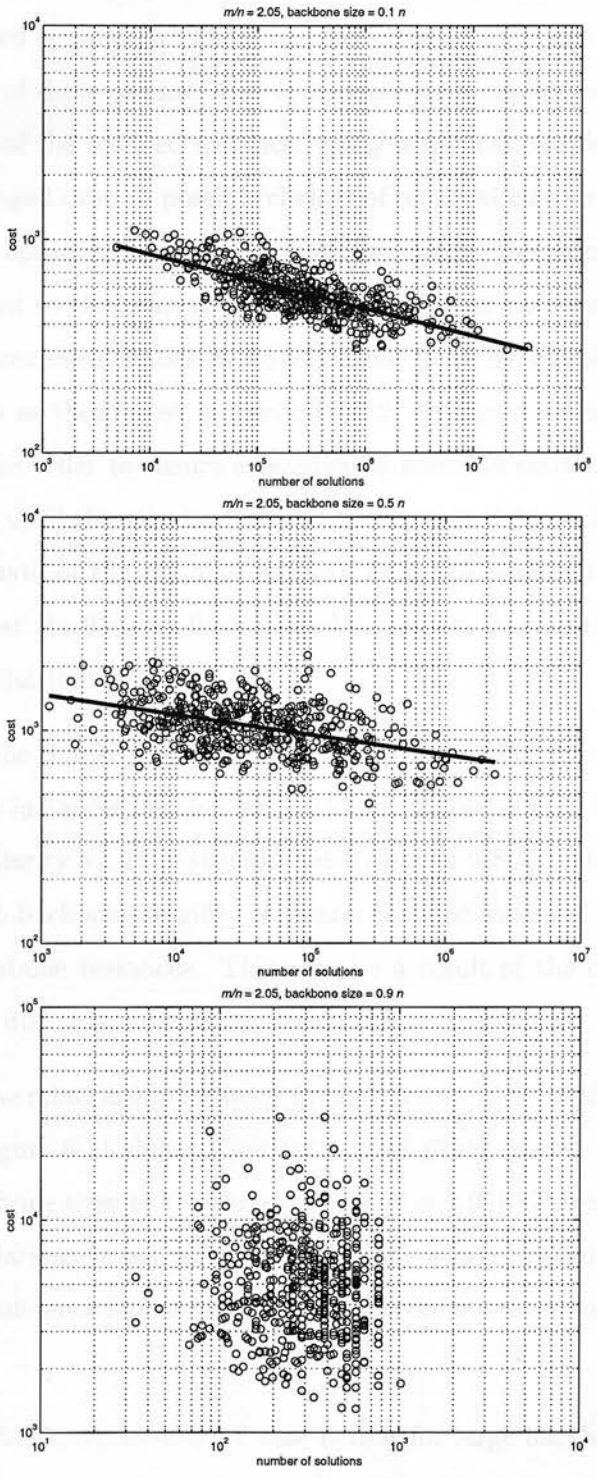

Figure 6.9: Log-log scatter plots of number of solutions against cost for NAE RANDOM-3-SAT threshold ($m/n = 2.05$) instances with backbone size controlled at $0.1n$ (top) $0.5n$ (middle) and $0.9n$ (bottom).

The *ffrbbf* metric is calculated as follows. Suppose a fixed fraction of the clauses of an instance are removed at random. There is a reduction in backbone size (possibly zero) due to the removal of these clauses. This is because some implicates of the full instance are not implicates of the reduced instance. *ffrbbf* is defined as the mean reduction in backbone size averaged over all possible choices of removed clauses. In all experiments reported in this chapter we removed 2.5% of the clauses i.e. 5 clauses in our range of m/n (we rounded to the nearest whole clause). It was not feasible to examine all possible choices of removed clauses so we estimated *ffrbbf* by sampling, using the same statistical methods as those used in Section 4.2.2. We used at least 100 samples on each instance and in order to ensure a reasonably accurate estimate, we continued to take more samples until the standard error was less than 0.05 $\times$ the sample mean. In this case our estimate of the population mean from the sample mean is accurate to within about 10% at the 95% confidence level, under the assumption that the sample mean is normally distributed.

Figure 6.10 shows the change in median *ffrbbf* as m/n is varied. As expected, controlled backbone instances instances become less backbone fragile as m/n is increased. This is one important similarity with the situation in RANDOM 3-SAT. One difference is that at $m/n = 2.00, 2.05$ backbone fragility is greater for medium-sized backbone instances than for large backbone instances. This may be a result of the change from *tbbr* to *ffrbbf* or it may be due to a difference between the two problems.

We now examine the relationship between *ffrbbf* and cost with backbone size controlled and m/n fixed. Figure 6.11 shows a scatter plot of *ffrbbf* against log of WSAT/SKC cost for three backbone sizes at the threshold ($m/n = 2.05$). Table 6.3 gives summary data for the correlations in all collections of NAE 3-SAT instances, including rank correlations and confidence intervals obtained using the bootstrap method as in Section 4.3.3.

A similarity with the RANDOM-3-SAT case is that for large backbone instances there is a moderate correlation with r values around 0.5. Note that the existence of this correlation is a key correct prediction of the hypothesis that the decay in backbone fragility underlies the decay in cost. One difference with the RANDOM-3-SAT case is that there is also a correlation for medium and small backbone instances which is

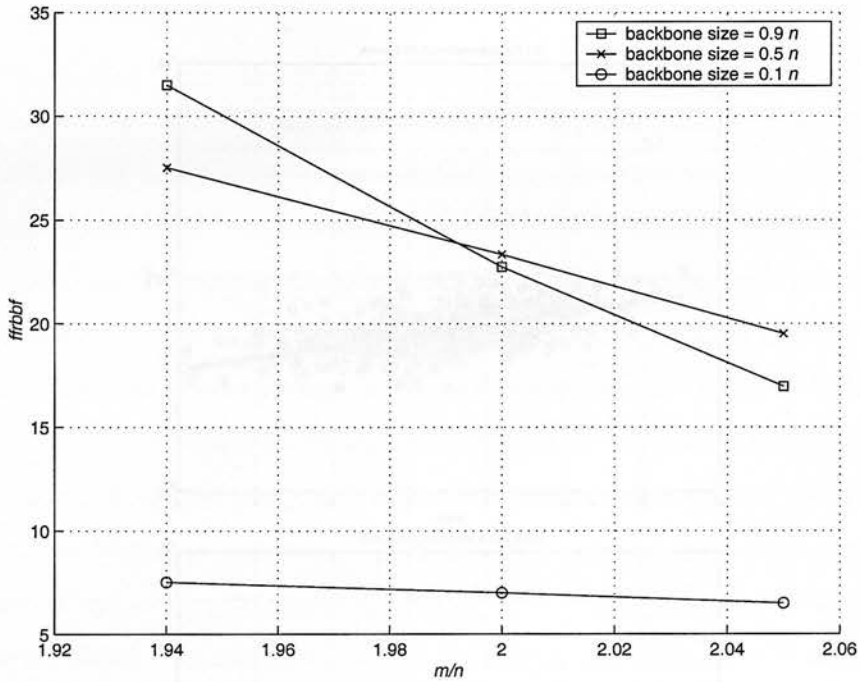

Figure 6.10: Plot of median *ffrbbf* as m/n is varied for NAE RANDOM-3-SAT instances with backbone size controlled at $0.1n$, $0.5n$ and $0.9n$.

possibly stronger than that for large backbone instances. Again, this may be apparent because of the change in our measure of backbone fragility or it may be a difference between the two problems.

m/n	Backbone size	Intercept of *lsr* fit (3 s. f.)	Gradient of *lsr* fit (3 s. f.)	r (3 s. f.)	$r^{-95\%}$ (3 s. f.)	$r^{+95\%}$ (3 s. f.)	Rank corr. coefficient (3 s. f.)
1.94	10	2.34	0.0646	0.610	0.552	0.666	0.590
	50	2.80	0.0124	0.666	0.614	0.713	0.660
	90	3.52	0.0108	0.553	0.490	0.615	0.565
2.00	10	2.42	0.0521	0.538	0.474	0.590	0.532
	50	2.80	0.0120	0.671	0.621	0.718	0.665
	90	3.54	0.00950	0.440	0.364	0.510	0.447
2.05	10	2.42	0.0510	0.557	0.497	0.615	0.550
	50	2.77	0.0129	0.661	0.612	0.706	0.647
	90	3.42	0.0139	0.534	0.478	0.595	0.532

Table 6.3: Data on correlations between *ffrbbf* and log of WSAT/SKC cost for NAE RANDOM-3-SAT backbone controlled instances.

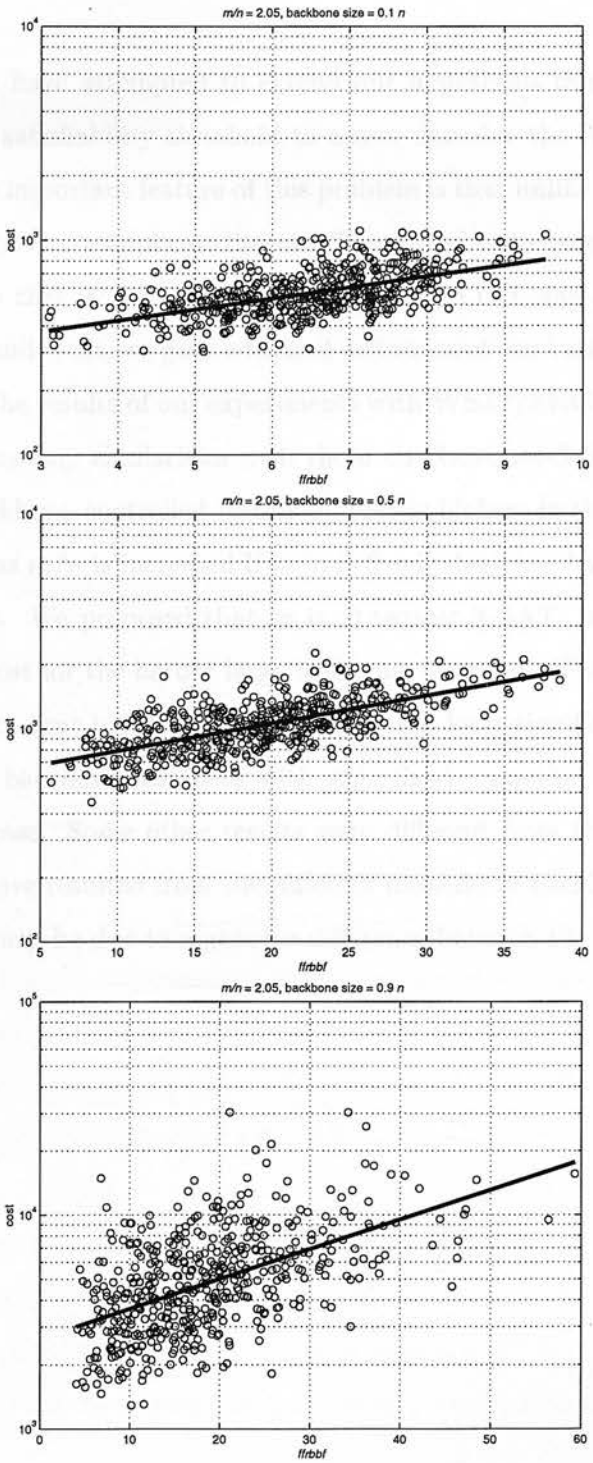

Figure 6.11: Scatter plot of *ffrbbf* against log of WSAT/SKC cost for NAE RANDOM-3-SAT threshold ($m/n = 2.05$) instances with backbone size controlled at $0.1n$ (top) $0.5n$ (middle) and $0.9n$ (bottom).

6.5 Summary

In this chapter we have attempted to extend our hypothesis from Chapter 4 about WSAT cost at the satisfiability threshold to a new domain: the NOT-ALL-EQUAL 3-SAT problem. The important feature of this problem is that unlike in 3-SAT, instances of NAE 3-SAT cannot have implicate literals. This forced us to generalise our definition of the backbone so that it is meaningful in NAE 3-SAT. Using a method based on that of Culberson and Gent, we gave a revised definition of the backbone which applies to NAE 3-SAT. The results of our experiments with WSAT/SKC on NAE RANDOM-3-SAT showed some key similarities with those on RANDOM-3-SAT from earlier in the thesis. For backbone controlled instances, cost is highest in the underconstrained region and decays as m/n is increased If m/n is fixed, there is a large variation in cost for these instances. We proposed that as in RANDOM-3-SAT, backbone fragility is the cause of high cost for the harder large backbone instances. This hypothesis makes a correct prediction, that backbone fragility accounts for a significant portion of cost variation for large backbone instances with m/n fixed – another similarity with the RANDOM-3-SAT case. Some other results were different from the RANDOM-3-SAT case. These may have resulted from the different measure of backbone fragility which was used, or they may be due to a genuine difference between the two problems.

Chapter 7

Conclusions and Further Work

In this last chapter we summarise the implications of the research work presented in the thesis and give suggestions for future work.

Recall that the aim of the thesis was to explain WSAT behaviour on random SAT instances. By elucidating this behaviour we developed a speculative account of the behaviour of WSAT and, inspired by this, a featural theory centred around the new concept of backbone fragility. This theory is the major contribution of the thesis. Section 7.1 summarises the speculative account of behaviour, the featural theory and the evidence for it. Sections 7.2 and 7.3 suggest areas of further work where research could build on the thesis results, emphasising particularly where the theory complements the results of other work. Section 7.4 is a brief epilogue.

7.1 The contribution of the thesis

Our summary of the contribution of the thesis is divided into two sections. In Section 7.1.1 we summarise our speculative account of the behaviour of WSAT on RANDOM-3-SAT threshold instances. In Section 7.1.2 we summarise the featural theory and give pointers to evidence for the theory which has been presented earlier in the thesis.

7.1.1 A speculative account of WSAT behaviour

Here we summarise our own intuitions about the behaviour of WSAT at the threshold of RANDOM-3-SAT, purely to illustrate how the featural theory originated. We should

emphasise that the propositions in this section are much more conjectural than those in the following section.

1. We speculate that WSAT cost on a RANDOM-3-SAT instance is largely determined by the likelihood of the algorithm to be at a low Hamming distance from a solution shortly after the early hill-climbing part of the search. We think that in the second, longer plateau-like phase, WSAT is unguided towards the solutions and is searching blindly among the quasi-solutions. So we think the length of the second phase will be largely determined by the Hamming distance between the algorithm's position and a solution at the end of the hill-climbing phase.

2. We conjecture that if solutions are widely-distributed, since this means they occur in different regions, a solution is likely to be present near the random initial assignment. Hill-climbing is then likely to position the search near to a solution and the plateau-like phase will be shorter. On the other hand, we think that if solutions are narrowly-distributed, the random initial assignment is not likely to be near to the solutions and so hill-climbing may (depending on other features of the instance) position the search at a point distant from the solutions, causing the plateau-like phase to be longer.

3. We speculate that on RANDOM-3-SAT threshold instances, the number of variables not mentioned in the backbone corresponds to a large extent to the average Hamming distance between pairs of solutions, which is one measure of how widely distributed the solutions are. This is why we think that small-backbone instances are easier than large-backbone instances.

4. For small-backbone RANDOM-3-SAT instances, the extent to which solutions are widely-distributed presumably varies. There is probably more variation in this factor in small-backbone instances than in large-backbone instances. We speculate that for RANDOM-3-SAT instances of a given backbone size the number of solutions corresponds to how widely-distributed solutions are.

5. We speculate that the large differences in WSAT cost between large backbone instances are not due to differences in how widely solutions are distributed. We think that large backbone instances can have other features which, if present,

cause WSAT to initially hill-climb to assignments that are Hamming distant from the solutions, so hindering search success. It is the presence or lack of these features which accounts for the differences in search cost, rather than how widely solutions are distributed.

6. We speculate that the immediate cause of cost variation in large-backbone instances is variation in the extensiveness of the quasi-solution area. We think that when the quasi-solution area is extensive, cost is high because hill-climbing from a random initial assignment is ineffective[1]. We think that hill-climbing is ineffective in this case because quasi-solutions are available at locations Hamming-near to the initial assignment, and these are more readily visited. The plateau phase then begins with the current assignment still at some Hamming distance from the nearest solution. When the quasi-solution area is less extensive, cost is lower because the hill-climbing phase, though longer, is more effective and ends at a quasi-solution which is Hamming-near a solution.

7. We speculate that in RANDOM-3-SAT, just as we think backbone size corresponds to how widely distributed solutions are, backbone fragility corresponds to how widely distributed quasi-solutions are compared to solutions. This is why we think that backbone fragility represents the cause of cost variation in large-backbone instances.

7.1.2 The backbone fragility theory of WSAT cost

The above intuitions led us to the formulation of backbone fragility as part of our featural theory which predicts WSAT's behaviour on RANDOM-3-SAT threshold instances. A substantial part of the work was dedicated to testing this theory. The main points of the featural theory are summarised here (in italics), along with pointers to the evidence from the thesis and from other work which supports each theory element.

1. *Among satisfiable instances, as m/n is increased, there is a transition from small- to large-backbone instances near the satisfiability threshold* . Evidence: Parkes

[1] This assumption could feasibly be checked using experiments, although this was not done as part of our work.

(1997).

2. *Backbone size is positively correlated with cost near the threshold.* Evidence: Parkes (1997), and data in Section 3.3.3.

3. *Near the threshold, for instances of a given backbone size, cost decays as m/n is increased.* Evidence: Parkes (1997), and data in Section 3.3.3.

4. *Near the threshold, if m/n is fixed, for instances of a given backbone size there is a large variation in cost.* Evidence: data in Section 3.3.3.

5. *For instances of a small given backbone size near the threshold, if m/n is fixed, variation in cost is accounted for by variation in the number of solutions.* Evidence: data in Section 3.3.4.

6. *For instances of a large given backbone size near the threshold, if m/n is fixed, variation in cost is not accounted for by variation in the number of solutions.* Evidence: data in Section 3.3.4.

7. *For instances of a large given backbone size near the threshold, if m/n is fixed, variation in cost is caused by variation in a factor represented adequately by backbone fragility.* This causal hypothesis makes three correct predictions which we tested in Chapter 4:

   (a) *For instances of a large given backbone size, if m/n is fixed, variation in cost is accounted for by variation in backbone fragility.* Evidence: data in Section 4.3.

   (b) *If* RANDOM-3-SAT *instances are altered so as to become more backbone fragile, their cost increases.* Evidence: data in Section 4.5.

   (c) *If the size of the backbone is highly dependent on the presence of a clause, then the clause will be more often unsatisfied during search than average.* This is a prediction of the model of search which motivated backbone fragility. Evidence: data in Section 4.6

8. *The backbone fragility theory can be applied to explain* WSAT *cost behaviour in different scenarios:*

(a) *The* WSAT *cost peak in* RANDOM-3-SAT *instances.* The evidence for this is summarised in Section 4.8.

(b) *The difference in* WSAT *cost between cost-typical* RANDOM-2+p-SAT *threshold instances at different values of* p. Preliminary experiments in Section 5.5 indicated that a plausible explanation for this difference is backbone fragility.

(c) *Differences in* WSAT *cost in double-clause encodings of* (NAE) RANDOM-3-SAT *instances when backbone size is large and* m/n *is fixed and the fall in cost when* m/n *is varied.* Evidence that backbone fragility causes this appears in Section 6.4.2. The results from Chapter 6 are also evidence that the theory generalises to threshold instances of at least one constraint problem where the definition of the backbone must be broadened.

7.2 Extending knowledge about hard instances

This section outlines suggestions for further work possibilities, focusing on promising ways in which our knowledge about the nature of hard instances could be extended in the light of the new results from this thesis and other recent work.

This thesis has concentrated on a single algorithm and a rather narrow range of problem instance types. Ultimately we would like to understand the operation of many different search algorithms on as broad a range of instances as possible.

WSAT is just one of a vast array of techniques which has been applied to the SAT problem. Many other techniques have radically different architectures and so it will be necessary to see what elements of the theory apply to these. Sections 7.2.1 and 7.2.2 suggest two initial directions which could be followed for this purpose.

The RANDOM-k-SAT model, although it appears to be universally hard, is in some respects unrepresentative. For example, in real world SAT instances, it may be that clauses are likely to be of different lengths and that clauses involving one particular set of variables are more likely to appear than clauses involving another set. Therefore, it will be important to find out which results from this domain generalise to other kinds of problem instance and which do not. Section 7.2.3 suggests an area where research

towards this objective could begin.

7.2.1 A cross-algorithm explanation of hard instances

Gent and Walsh (1996) looked at the probability that an unsatisfiable random SAT instance (not RANDOM-3-SAT in this case) became satisfiable if a fixed number of clauses are removed at random. The unsatisfiable threshold instances which had the highest computational cost for their version of DPLL were found to be those which were unsatisfiability-fragile – their unsatisfiability was sensitive to the random removal of clauses. It may therefore be that the fragility of an instance's unsatisfiability is correlated with computational cost of unsatisfiable threshold instances for complete procedures such as DPLL.

Recent work such as that by Mammen and Hogg (1997) and Culberson and Gent (2000) has suggested that the cause of hard instances for complete algorithms at the solvability thresholds of colouring and SAT is closely related to the nature of the MUSs of the instances. For a complete algorithm, it is thought that the hardest instances are those where the smallest MUSs are large[2], involving many constraints and variables. It is also thought that the number of MUSs may be important in the cost. Both these hypotheses are reasonable, since to prove unsolvability, an algorithm must identify an MUS, and if these are large and scarce, this should be difficult.

One way of viewing unsatisfiability fragility as measured by Gent and Walsh (1996) is as a proxy for the properties of the MUSs of an instance. If the smallest MUS of an instance is large, and there are few MUSs, then unsatisfiability is more likely to depend on a particular constraint or small set of constraints. Hence unsatisfiability is more likely to be lost if a random set of constraints is removed, and so the instance in this case would be more unsatisfiability-fragile.

Similarly, backbone fragility may be seen as a proxy for the properties of an instance's BMSs, or for the properties of those minimal sub-instances which induce individual

[2] However, this only appears to hold if the algorithm has some intelligent features such as backjumping. For standard DPLL (which has none of these features), the hardest instances are those with a very small number of very small MUSs – these are the so-called exceptionally hard instances. There is also some evidence that even for the more intelligent algorithms these instances are exceptionally hard (Bayardo and Schrag 1996).

implicate literals. Again, if these sub-instances are large and scarce, an instance will be backbone-fragile. The backbone fragility theory therefore represents an interesting link between the two classes of algorithms. In fact, unsatisfiability can be seen as the case when the empty clause is an implicate. So computational cost for both classes may be dependent on the nature of the sub-instances which induce implicates.

This suggests the possibility that although local search and complete search algorithms are completely different in architecture, they may be susceptible to the same instance properties. This in turn implies that these instance properties may be fundamental to the nature of hard distributions.

7.2.2 Non-random artificially backbone-fragile instances

The hypothesis proposes a causal link between backbone fragility and WSAT cost for large backbone instances. One prediction from this statement is that if instances are made to have large backbones and be very backbone fragile they should be difficult for WSAT. This prediction was tested in Section 4.5 when random instances were altered so as to be more backbone-fragile. Another possibility is the construction of *non-random* SAT instances, specifically designed to be backbone-fragile. The hypothesis predicts that these should be difficult for WSAT variants. A class of such instances was suggested by Steven Prestwich (Prestwich 2000a). These are constructed as follows:

$$
\begin{gathered}
x_1 \wedge x_2 \wedge \\
[(x_1 \wedge x_2) \rightarrow (x_3 \wedge x_4)] \wedge \\
[(x_3 \wedge x_4) \rightarrow (x_5 \wedge x_6)] \wedge \\
[(x_5 \wedge x_6) \rightarrow (x_7 \wedge x_8)] \wedge \\
\cdots \\
[(x_{2i+1} \wedge x_{2i+2}) \rightarrow (x_{2i+3} \wedge x_{2i+4})] \wedge \\
\cdots \\
[(x_{2M+1} \wedge x_{2M+2}) \rightarrow (x_{2M+3} \wedge x_{2M+4})]
\end{gathered}
$$

The index i ranges from 0 to M. M governs the size of the formula. The formulas are easily translated into conjunctive normal formulas with $O(M)$ clauses. They have

a maximal backbone consisting of all the variables. This is due to the chain of implications starting at $x_1 \wedge x_2$ and adding two more literals to the backbone with each new line. However, the removal of a few clauses is likely to leave a break in the chain quite early on. All literals added to the backbone after this break are then no longer implicates. The instances are therefore also quite backbone-fragile.

Some initial experiments (Hoos and O'Neill 2000; Prestwich 2000a) showed that these instances are very difficult for a range of existing variants of WSAT, which lends credibility to our hypothesis that these algorithms are adversely affected by backbone fragility. It may also show that the effects of backbone fragility are not limited to the domain of random instances.

Holger Hoos and Kevin O'Neill have devised an algorithm WWSAT which combines WSAT with a clause weighting scheme. This solves the above instances quickly (Hoos and O'Neill 2000). Another algorithm CLS (Prestwich 2000b) which combines local and constructive search also appears to solve the instances without difficulty (Prestwich 2000a). This prompts the conclusion that a large backbone combined with backbone fragility is not a sufficient condition for an instance to be hard for every local search algorithm, since we have local search algorithms which solve some large backbone, backbone-fragile instances without difficulty. However, the combination may be a sufficient condition for an instance to be hard for a certain class of local search algorithms encompassing the standard WSAT variants.

This line of research may have some interesting implications for future extension of the theory presented in Section 7.1.2. Assuming that the new algorithms are adversely affected by the RANDOM-3-SAT satisfiability threshold and that there is variation in their cost at the threshold, there are two possibilities:

1. They find the same threshold instances hard as the standard WSAT variants do i.e. backbone size accounts for much of the variance and backbone fragility accounts for much of the remaining variance. This would imply that the combination of backbone size and backbone fragility is a latent factor which is itself correlated with some other feature which causes hardness more directly.

2. They find a different set of instances at the satisfiability threshold hard. This

would reinforce the idea that the combination of backbone size and backbone fragility is really only relevant for some subclass of local search algorithms.

7.2.3 A rival explanation of the cost peak

Recent work by Achlioptas *et al.* (2000) has investigated the behaviour of WSAT on a novel class of SAT instances: encodings of the QUASIGROUPS-WITH-HOLES (QWH) problem. Achlioptas *et al.* devised an efficient method by which only satisfiable instances are generated, but where the number of constraints can be varied as in RANDOM-3-SAT. One interesting result was that they found a peak in WSAT cost which coincided with the point at which large backbone instances emerge. The authors speculate:

> "The reasons for the correlation between problem hardness and the appearance of the backbone are not fully understood at this time. [...] For local search procedures, an explanation might be developed by considering the relationship between the backbone and set of solutions to the instances. When the backbone is small, there are many solutions widely distributed in the search space, and so local search may quickly find one. When the backbone is near 1 [i.e. large], the solutions are tightly clustered, so that all clauses "vote" to push the search in the same direction. A partial backbone, however, may indicate that solutions are in different clusters that are widely distributed, with different clauses pushing the search in different directions [...]"

Thus the authors suggest an alternative hypothesis to the backbone fragility theory. Their key conjecture above is that hardness emerges because of the presence of medium-sized backbone instances, which do indeed occur most frequently near the satisfiability threshold in RANDOM-3-SAT. However, in RANDOM-3-SAT this conjecture is contradicted by the positive correlation between an instance's backbone size and the cost for WSAT to solve the instance; at the threshold, the large backbone instances are the hardest, not the medium-sized backbone instances.

The correspondence between the emergence of the large backbone instances and the WSAT cost peak in QWH is also consistent with the backbone fragility theory, because we would expect the peak at the point where the backbone size changes. This is because the determinant of cost changes at this point from the number of solutions to backbone fragility. QWH would therefore be an interesting future case study for testing the backbone fragility theory against the conjecture of Achlioptas *et al.*

7.3 Exploiting knowledge about backbone-fragility

This section covers further work with a different objective: that of engineering better algorithms. The knowledge that backbone fragility is a difficulty may allow us to direct the design of new algorithms to mitigate it. Here two possibilities along these lines are outlined.

7.3.1 Intelligent initialisation

Given the very strong correlation between *hci* and cost, we concluded that the hard instances are those where the assignment found as a result of hill-climbing is generally Hamming distant from the nearest solution. A further experiment could investigate the effect of varying the Hamming distance between the initial assignment and the nearest solution. A very possible result is a positive correlation between this and the run length. This would indicate that the success of a particular run is dependent on the location of the initial assignment.

Current local search algorithms typically begin each try with a random assignment. The initial assignment could be chosen more intelligently, so as to reduce the Hamming distance between it and the nearest solution. This may then cause the length of the run to be reduced. The benefit from a reduction in run length may outweigh the computational cost of the intelligent initialisation. Non-random initialisation strategies have been investigated before. For example, in OSAT (Gent and Walsh 1993b) the initial assignment was greedily constructed so as to minimise the number of unsatisfied clauses i.e. it found a quasi-solution. However, the results from this thesis suggest that the difficulty is perhaps not finding a quasi-solution but instead finding an assignment

which is at a low Hamming distance from the nearest solution. This may suggest why OSAT did not perform much better than GSAT.

How tries are initialised more intelligently is an open research question. One recent advance in tree-search style techniques is *discrepancy-based* search (Harvey and Ginsberg 1995; Walsh 1997). Discrepancy-based search effectively probes promising but radically different regions of the search space at each iteration. Each iteration could be used to produce an initial assignment for local search. Hence subsequent tries can begin in different promising regions. This may increase the likelihood of a try beginning at an assignment which is at a low Hamming distance to the nearest solution.

7.3.2 Adding implicates to reduce backbone fragility

Rather than altering the algorithm, it may be possible to modify the instance to avoid the effects of backbone fragility. Recall from Section 4.2.3 that as m/n is increased in RANDOM-3-SAT, backbone fragility decreases. It was proposed in Section 4.2.1 that the main reason for this is the addition of clauses which allow backbone literals to be derived from the instance via alternative proofs. Hence, the backbone fragility of an instance can be reduced by the addition of clauses with this property.

However, computing the amount by which a given new clause reduces backbone fragility is unlikely to be cost-effective. The approach would be to add clauses to the instance with the hope that these will sufficiently reduce backbone fragility. Furthermore it may be preferable that the added clause does not contradict any existing solutions to the problem. Implicates are good candidates for this since they are guaranteed not to contradict any existing solutions. Certain classes of implicates can be computed efficiently, for example the Horn closure (Molony 1999). A method whereby implicates are added with the aim of improving local search is given by Cha and Iwama (1996). It may prove that local search is made easier because of the reduction in backbone fragility.

7.4 Epilogue

The research question was why solutions can be hard to find. In this thesis we have concentrated on a specific algorithm and problem instance type. We formulated and tested a featural theory to answer the question for this particular scenario, based on a new feature of instances, backbone fragility. We have also conducted some initial experiments aimed at broadening the range of instances covered and given some suggestions at how further research establishing the scope of the theory may be conducted. Thus, while many questions remain unanswered, we have made a contribution to the overall knowledge about problem instance hardness in general by studying this particular scenario. We hope that this thesis will serve as a starting point for the broadening and exploitation of this knowledge.

Bibliography

Achlioptas, D. and Sorkin, G. B. (2000). Optimal Myopic Algorithms for Random 3-SAT . In *Proceedings of FOCS-2000*, pages 590–600.

Achlioptas, D., Kirousis, L. M., Kranakis, E., and Krizanc, D. (1997). Rigorous Results for (2+p)-SAT. In *Proceedings of RALCOM 97*, pages 1–10.

Achlioptas, D., Gomes, C., Kautz, H., and Selman, B. (2000). Generating Satisfiable Problem Instances. In *Proceedings of AAAI-00*, pages 256–261. AAAI Press / The MIT Press.

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

Bayardo, R. J. and Schrag, R. (1996). Using CSP Look-Back Techniques to Solve Exceptionally Hard SAT Instances. In *Proceedings of the Second International Conference on the Principles and Practice of Constraint Programming*, pages 46–60. Springer.

Beame, P., Karp, R. M., Pitassi, T., and Saks, M. E. (1998). On the Complexity of Unsatisfiability Proofs for Random k-CNF Formulas. In *Proceedings of the 30th Annual ACM Symp. on the Theory of Computing*, pages 561–571.

Cha, B. and Iwama, K. (1996). Adding New Clauses for Faster Local Search. In *Proceedings AAAI-96*, pages 332–337. AAAI Press / The MIT Press.

Cheeseman, P., Kanefsky, B., and Taylor, W. (1991). Where the Really Hard Problems Are. In *Proceedings of IJCAI-91*, pages 331–340. Morgan Kaufmann.

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

Clark, D., Frank, J., Gent, I. P., MacIntyre, E., Tomov, N., and Walsh, T. (1996). Local Search and the Number of Solutions. In *Proceedings of the Second International Conference on the Principles and Practice of Constraint Programming*, pages 119–133. Springer.

Cohen, P. (1995). *Empirical Methods for Artificial Intelligence*. The MIT Press.

Cook, S. A. (1971). The Complexity of Theorem-Proving Procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158.

Cook, S. A. and Mitchell, D. G. (1997). Finding Hard Instances of the Satisfiability Problem: A Survey. In *Satisfiability Problem: Theory and Applications*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society.

Crawford, J. M. and Auton, L. D. (1996). Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence*, **81**, 31–57.

Culberson, J. (1999). Personal communication.

Culberson, J. and Gent, I. P. (1999a). On the Completeness of WalkSAT for 2-SAT. Technical Report APES-15-1999, APES Research Group.
Available from `http://apes.cs.strath.ac.uk/apesreports.html`.

Culberson, J. and Gent, I. P. (1999b). Well out of reach: Why hard problems are hard. Technical Report APES-13-1999, APES Research Group.
Available from `http://apes.cs.strath.ac.uk/apesreports.html`.

Culberson, J. and Gent, I. P. (2000). Frozen Development in Graph Coloring. Technical Report APES-19-2000, APES Research Group.
Available from `http://apes.cs.strath.ac.uk/apesreports.html`.

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

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

Dubois, O., Boufkhad, Y., and Mandler, J. (2000). Typical random 3-sat formulae and the satisfiability threshold. In *Proceedings of the Eleventh ACM-SIAM Symposium on Discrete Algorithms*, pages 124–126.

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

Frank, J. (1996a). Learning Short-Term Weights for GSAT. Technical Report CSE-96-14, Department of Computer Science, University of California, Davis.

Frank, J. (1996b). Weighting for Godot: Learning Heuristics for GSAT. In *Proceedings of AAAI-96*, pages 338–343. AAAI Press / The MIT Press.

Frank, J., Cheeseman, P., and Stutz, J. (1997). When Gravity Fails: Local Search Topology. *J. Artificial Intelligence Research*, **7**, 249–281.

Friedgut, E. (1999). Sharp thresholds of graph properties, and the k-sat problem. *Journal of the American Mathematical Society*, **12**, 1017–1054. (Note: appendix by J. Bourgain).

Frieze, A. and Suen, S. (1996). Analysis of Two Simple Heuristics on a Random Instance of k-SAT. *Journal of Algorithms*, **20**(2), 312–355.

Garey, M. and Johnson, D. (1979). *Computers and intractability : a guide to the theory of NP-completeness*. W. H. Freeman.

Gent, I. P. (1998). On the Stupid Algorithm for Satisfiability. Technical Report APES-02-1998, APES Research Group, Leeds University / Strathclyde University.

Gent, I. P. and Walsh, T. (1993a). An Empirical Analysis of Search in GSAT. *J. Artificial Intelligence Research*, **1**, 47–59.

Gent, I. P. and Walsh, T. (1993b). Towards an Understanding of Hill-climbing Procedures for SAT. In *Proceedings of AAAI-93*, pages 28–33. AAAI Press / The MIT Press.

Gent, I. P. and Walsh, T. (1995). Unsatisfied Variables in Local Search. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions (Proceedings of AISB-95)*, pages 73–85, Amsterdam. IOS Press.

Gent, I. P. and Walsh, T. (1996). The satisfiability constraint gap. *Artificial Intelligence*, **81**, 59–80.

Gent, I. P., MacIntyre, E., Prosser, P., and Walsh, T. (1996). The Constrainedness of Search. In *Proceedings of AAAI-96*, pages 246–252. AAAI Press / The MIT Press.

Gent, I. P., MacIntyre, E., Prosser, P., and Walsh, T. (1997). The Scaling of Search Cost. In *Proceedings of AAAI-97*, pages 315–320. AAAI Press / The MIT Press.

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

Guerra e Silva, L., Miguel Silveira, L., and Marques-Silva, J. P. (1999). Algorithms for Solving Boolean Satisfiability in Combinational Circuits. In *Proceedings of the IEEE/ACM Design, Automation and Test in Europe Conference*. IEEE Computer Society.

Harvey, W. and Ginsberg, M. (1995). Limited Discrepancy Search. In *Proceedings of IJCAI-95*, pages 607–613. Morgan Kaufmann.

Hirsch, E. A. (2000). SAT Local Search Algorithms: Worst-Case Study. *Journal of Automated Reasoning*, **24**(1/2), 127–143.

Hogg, T. and Williams, C. P. (1994). The hardest constraint problems: a double phase transition. *Artificial Intelligence*, **69**, 359–377.

Hooker, J. N. and Vinay, V. (1995). Branching Rules for Satisfiability. *Journal of Automated Reasoning*, **15**, 359–383.

Hoos, H. (1998). *Stochastic Local Search - Methods, Models, Applications*. Ph.D. thesis, Darmstadt University of Technology.

Hoos, H. (1999a). On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT. In *Proceedings of AAAI-99*, pages 661–666. AAAI Press / The MIT Press.

Hoos, H. (1999b). SAT-Encodings, Search Space Structure, and Local Search Performance. In *Proceedings of IJCAI-99*, pages 296–302. Morgan Kaufmann.

Hoos, H. and O'Neill, K. (2000). Personal communication.

Hoos, H. and Stützle, T. (1998). Characterising the Run-time Behaviour of Stochastic Local Search. Technical Report AIDA-98-01, Darmstadt University of Technology.

Jeroslow, R. G. and Wang, J. (1990). Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, **1**, 167–187.

Kautz, H. and Selman, B. (1992). Planning as Satisfiability. In *Proceedings of ECAI-92*, pages 359–363. John Wiley & Sons.

Kautz, H. and Selman, B. (1996). Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. In *Proceedings of AAAI-96*, pages 1194–1201. AAAI Press / MIT Press.

Kautz, H., Selman, B., and Jiang, Y. (1997). General Stochastic Approach to Solving Problems with Hard and Soft Constraints. In *Satisfiability Problem: Theory and Applications*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society.

Koutsoupias, E. and Papadimitriou, C. H. (1992). On the greedy algorithm for satisfiability. *Information Processing Letters*, **43**(1), 53–55.

Larrabee, T. and Tsuji, Y. (1992). Evidence for a Satisfiability Threshold for Random 3CNF Formulas. Technical Report UCSC-CRL-92-42, Jack Baskin School of Engineering, University of California, Santa Cruz.

Mammen, D. L. and Hogg, T. (1997). A New Look at the Easy-Hard-Easy Pattern of Combinatorial Search Difficulty. *J. Artificial Intelligence Research*, **7**, 47–66.

Mazure, B., Saïs, L., and Grégoire, É. (1997). Tabu search for SAT. In *Proceedings of AAAI-97*, pages 281–285. AAAI Press / The MIT Press.

McAllester, D., Selman, B., and Kautz, H. (1997). Evidence for Invariants in Local Search. In *Proceedings of AAAI-97*, pages 321–326. AAAI Press / The MIT Press.

Mitchell, D. G., Selman, B., and Levesque, H. J. (1992). Hard and Easy Distributions of SAT Problems. In *Proceedings of AAAI-92*, pages 459–465. AAAI Press / The MIT Press.

Molony, J. (1999). *Symmetry Arguments in Automated Reasoning*. Ph.D. thesis, (submitted) Division of Informatics, University of Edinburgh.

Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. (1999a). 2+P-SAT: Relation of Typical-Case Complexity to the Nature of the Phase Transition. *Random Structures and Algorithms*, **15**, 414–440.

Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. (1999b). Determining computational complexity from characteristic 'phase transitions'. *Nature*, **400**, 133–137.

Morris, P. (1993). The Breakout Method for Escaping from Local Minima. In *Proceedings of AAAI-93*, pages 40–45. AAAI Press / The MIT Press.

Papadimitriou, C. H. (1991). On selecting a satisfying truth assignment. In *Proc. 32nd IEEE Symp. on the Foundations of Comp. Sci.*, pages 163–169.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

Parkes, A. (1997). Clustering at the Phase Transition. In *Proceedings of AAAI-97*, pages 340–345. AAAI Press / The MIT Press.

Parkes, A. and Walser, J. (1996). Tuning Local Search for Satisfiability Testing. In *Proceedings of AAAI-96*, pages 356–362. AAAI Press / The MIT Press.

Prestwich, S. (2000a). Personal communication.

Prestwich, S. (2000b). A Hybrid Search Architecture Applied to Hard Random 3-SAT and Low-Autocorrelation Binary Sequences. In *Proceedings of the Sixth International Conference on the Principles and Practice of Constraint Programming*, pages 337 – 352. Springer.

Schrag, R. and Crawford, J. (1996). Implicates and prime implicates in Random 3-SAT. *Artificial Intelligence*, **81**, 199–222.

Selman, B. and Kautz, H. (1993a). An Empirical Study of Greedy Local Search for Satisfiability Testing. In *Proceedings of AAAI-93*, pages 46–53. AAAI Press / The MIT Press.

Selman, B. and Kautz, H. (1993b). Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proceedings of IJCAI-93*, pages 290–295. Morgan Kaufmann.

Selman, B., Levesque, H. J., and Mitchell, D. G. (1992). A New Method for Solving Hard Satisfiability Problems. In *Proceedings of AAAI-92*, pages 440–446. AAAI Press / The MIT Press.

Selman, B., Kautz, H., and Cohen, B. (1994). Noise Strategies for Improving Local Search. In *Proceedings of AAAI-94*, pages 337–343. AAAI Press / The MIT Press.

Selman, B., Mitchell, D., and Levesque, H. J. (1996). Generating Hard Satisfiability Problems. *Artificial Intelligence*, **81**(1–2), 17–29.

Singer, J., Gent, I. P., and Smaill, A. (2000a). Backbone Fragility and the Local Search Cost Peak. *Journal of Artificial Intelligence Research*, **12**, 235–270.

Singer, J., Gent, I. P., and Smaill, A. (2000b). Local Search on Random 2+p-SAT. In *Proceedings of ECAI-2000*, pages 113–117. IOS Press.

Slaney, J., Fujita, M., and Stickel, M. (1995). Automated reasoning and exhaustive search: quasigroup existence problems. *Computers and Mathematics with Applications*, **29**, 115–132.

van Gelder, A. and Okushi, F. (1999). A propositional theorem prover to solve planning and other problems. *Annals of Mathematics and Artificial Intelligence*, **26**, 87–112.

Walsh, T. (1997). Depth-bounded Discrepancy Search. In *Proceedings of IJCAI-97*, pages 1388–1393. Morgan Kaufmann.

Yokoo, M. (1997). Why Adding More Constraints Makes a Problem Easier for Hill-Climbing Algorithms: Analysing Landscapes of CSPs. In *Proceedings of the Third International Conference on the Principles and Practice of Constraint Programming*, pages 356–370. Springer.

Zhang, H. and Stickel, M. (1994). Implementing the Davis-Putnam Algorithm by Tries. Technical Report 94-12, Department of Computer Science, University of Iowa.

Zhang, H., Bonacina, M. P., and Hsiang, J. (1996). PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems. *Journal of Symbolic Computation*, **21**(4), 543–560.

Appendix A

Bootstrap Methods

We summarise the methods as used in this context. Further explanation of resampling methods is given in Cohen (1995).

A.1 Bootstrap method for testing whether two medians are equal

This method was taken from David Howell's web pages on resampling:

`http://www.uvm.edu/~dhowell/StatPages/Resampling/Boot2Medians/bootstrapping_two_medians.html`

A bootstrap method can be used to test the hypothesis that the medians of two populations are equal given a pair of samples, one from each population.

We have two original samples $X_{sample} = \langle x_1, x_2, \ldots x_N \rangle$ and $Y_{sample} = \langle y_1, y_2, \ldots y_N \rangle$. We assume that the sample sizes are equal for simplicity. A *pseudo-sample* from one of these original consists of N data. A datum in a pseudo-sample from X_{sample} is x_q where q is a random number between 1 and N. Each datum in the pseudo-sample is chosen independently i.e. data are sampled from the original with replacement. We assume that our original samples are representative of the underlying population. Given this, composing many pseudo-samples and taking the median of each pseudo-sample gives us the sampling distribution of the median for the underlying population. If we take pairs of pseudo-samples, consisting of one pseudo-sample from X_{sample} and the other from Y_{sample} and collate the differences in the medians between the two pseudo-samples in each pair, this will give us the sampling distribution of the difference between the medians in the underlying population. If we use a large number of pairs of pseudo-samples, we can then study the sampling distribution of the difference between the medians to see whether it is consistent with the hypothesis that the difference is zero. If it is consistent with this hypothesis at the 95% level, then zero should fall between the 2.5^{th} and the 97.5^{th} percentiles. Otherwise, we can reject the hypothesis with 95% confidence.

A.2 Bootstrap estimation of confidence intervals for correlation coefficients

We have an original sample $\langle (x_1, y_1), (x_2, y_2), \ldots (x_N, y_N) \rangle$ of N pairs. A pseudo-sample from the original in this case consists of N pairs. The j^{th} pair in the pseudo-sample $(x_j^b, y_j^b) = (x_q, y_q)$ where q is a random number between 1 and N. Each pair in the pseudo-sample is chosen independently i.e. pairs are sampled from the original with replacement. We assume that our original sample of pairs of data is representative of the whole population of such pairs. Given this, composing pseudo-samples is just like sampling from the whole population. Therefore by measuring the correlation coefficient of many pseudo-samples, we can study what the correlation coefficient would have looked like had we taken many sets of samples from the whole population. From the distribution of the correlation coefficient among many pseudo-samples (the bootstrap sampling distribution) we can infer bounds on the confidence interval for the observed correlation coefficients.

Many pseudo-samples are taken, and the correlation coefficient is calculated for each of the pseudo-samples. This gives the bootstrap sampling distribution of the correlation coefficient. The 97.5^{th} percentile of this distribution is an upper bound on the 95% confidence interval for the correlation coefficient, and the 2.5^{th} percentile is a lower bound.

Appendix B

The Relationship Between BMSs and MUSs

Let C be a satisfiable SAT instance and $\{l_1, l_2, \ldots, l_k\}$ be the set of all literals entailed by C. Let d be the clause $\neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_k$.

Theorem C' is a BMS of C iff $C' \wedge d$ is an MUS of $C \wedge d$ $\square$

PROOF Suppose C' is a BMS of C. Then $C' \wedge d$, which is a sub-instance of $C \wedge d$, must be unsatisfiable, as d violates every literal in the backbone of C'. If d is removed from $C' \wedge d$, the result C' is satisfiable. If any other clause c is removed from $C' \wedge d$, there must be some literal from the backbone of C', l_i say, such that $(C' - \{c\}) \wedge \neg l_i$ is satisfiable. Therefore, since $\neg l_i$ is also a literal of d, $(C' - \{c\}) \wedge d$ is satisfiable. Therefore $C' \wedge d$ is an MUS of $C \wedge d$.

Conversely, suppose $C' \wedge d$ is an MUS of $C \wedge d$. Since $C' \wedge d$ is minimally unsatisfiable, C' is satisfiable. Since C' is a sub-instance of C, the backbone of C' must be a subset of the backbone of C. Suppose there were some literal l_j which was in the backbone of C but not in the backbone of C'. Then there would be a solution to $C' \wedge \neg l_j$. This would then also be a solution to $C' \wedge d$, since $\neg l_j$ is one literal of d. This contradicts $C' \wedge d$ being unsatisfiable and so there can be no l_j i.e. C' and C must have the same backbone.

$C' \wedge d$ is minimally unsatisfiable. Therefore for any clause c of C', $(C' - \{c\}) \wedge d$ is satisfiable. Any solution to $(C' - \{c\}) \wedge d$ must make some literal $\neg l_k$ of d true, and must therefore also be a solution to $(C' - \{c\}) \wedge \neg l_k$. Therefore l_k, which is in the backbone of C', is not in the backbone of $(C' - \{c\})$. Hence C' is a BMS of C $\square$