Understanding Stochastic Local Search Algorithms

An Empirical Analysis of the Relationship Between Search Space Structure and Algorithm Behaviour

by

Kevin R. G. Smyth

B.Sc., The University of British Columbia, 2002

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 27, 2004

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

KEVIN SMYTH
Name of Author *(please print)*

30/09/2004
Date (dd/mm/yyyy)

Title of Thesis: UNDERSTANDING STOCHASTIC LOCAL SEARCH ALGORITHMS: AN EMPIRICAL ANALYSIS OF THE RELATIONSHIP BETWEEN SEARCH SPACE STRUCTURE AND ALGORITHM BEHAVIOUR

Degree: MASTER OF SCIENCE Year: 2004

Department of COMPUTER SCIENCE
The University of British Columbia
Vancouver, BC Canada

Abstract

Combinatorial optimisation problems are an important and well-studied class of problems, with applications in most areas of the computing sciences. Because of their prominence, combinatorial optimisation problems and their related decision problems have been the focus of extensive research for several decades. The propositional satisfiability problem (SAT), in particular, has been the focus of a vast amount of research, and a class of algorithms known as stochastic local search (SLS) algorithms has emerged as the state-of-the art on a variety of SAT problem classes.

Much of the recent progress in algorithm development has been facilitated by an improved understanding of the properties of SAT instances and of high-performance SAT algorithms. This thesis studies the search space features underlying the behaviour of stochastic local search algorithms for SAT, extending existing results from the literature and providing novel contributions. Search space features such as plateaus and the interconnectivity between plateaus are defined and studied on a variety of SAT instances, and it is empirically demonstrated that features such as these are responsible for the wide range in instance hardness observed in distributions of syntactically identical SAT instances. Furthermore, the novel concept of the performance criticality of variables in SAT instances is introduced, and the connections between search space structure and performance criticality are investigated. Finally, methods for practically exploiting the knowledge gained by this search space analysis are briefly explored, with the goal of improving the state-of-the-art in SAT solving.

Contents

Abstract ii

Contents iii

List of Tables vi

List of Figures viii

List of Algorithms x

Acknowledgements xi

1 Introduction 1

1.1 Problem Definition and Motivation 2

1.2 Existing Work 3

1.2.1 Number of Solutions 3

1.2.2 Solution Clustering 4

1.2.3 Backbone Fragility 5

1.2.4 Local Minima 6

1.2.5 Effect of Search Space Structure on Algorithm Behaviour 7

1.2.6 Plateau Distribution 8

1.2.7 Predicting Local Search Cost 9

1.3 Thesis Outline 11

2 Preliminaries 12

2.1 Propositional Formulae 12

2.2 Search Space Components 14

2.3 SLS Algorithms for SAT 15

2.4 Empirical Methodology 17

3 Plateau Characteristics . . . 20
3.1 Definitions and Motivation . . . 21
3.1.1 State Types . . . 21
3.1.2 Plateaus . . . 22
3.2 Plateau characteristics of the UF-3-SAT distribution . . . 23
3.2.1 Experimental Methodology . . . 24
3.2.2 Fraction of Closed Plateaus . . . 24
3.2.3 Plateau Size . . . 25
3.2.4 Discussion . . . 27
3.3 Plateau characteristics of individual UF-3-SAT instances . . . 28
3.3.1 Experimental Methodology . . . 28
3.3.2 Number of Plateaus . . . 32
3.3.3 Fraction of Closed Plateaus . . . 33
3.3.4 Plateau Size . . . 35
3.4 Plateau characteristics of structured instances . . . 37
3.4.1 Number of Plateaus . . . 37
3.4.2 Plateau Size . . . 39
3.5 Internal Plateau Structure . . . 41
3.5.1 Branching Factor . . . 46
3.5.2 Diameter . . . 48
3.5.3 LMIN State Fraction . . . 48
3.5.4 Exit Distance . . . 50
3.6 Summary . . . 52

4 Plateau Connectivity . . . 54
4.1 Definitions and Motivation . . . 54
4.2 PCG Properties . . . 60
4.2.1 Out-Degree . . . 62
4.2.2 Target Depth . . . 63
4.3 Solution Reachability . . . 65
4.4 Properties of Traps in the Search Space . . . 69
4.4.1 Attractivity . . . 70
4.4.2 Escape Difficulty . . . 72
4.4.3 Connection With Mixture Models . . . 75
4.5 Summary . . . 77

5 Performance Critical Variables . . . 79
5.1 Motivation . . . 80
5.2 Oracles and Performance Criticality . . . 81
5.3 Performance Critical Variables for SLS Algorithms . . . 84
5.4 Performance Critical Variables for DPLL Algorithms . . . 89
5.5 Discussion . . . 91
5.5.1 Connection with Search Space Features . . . 92
5.5.2 An Approximate PCV Oracle . . . 95
5.6 Summary . . . 97

6 Conclusions and Future Work . . . 98

Bibliography . . . 102

A Test Suite Description . . . 107

B Binary Decision Diagrams . . . 110

List of Tables

3.1 Number of plateaus in UF-3-SAT instances, counting all plateaus 32
3.2 Number of plateaus in UF-3-SAT instances, counting only open plateaus 34
3.3 Number of plateaus in UF-3-SAT instances, counting only closed plateaus 34
3.4 Size of closed plateaus in UF-3-SAT instances 35
3.5 Fraction of states contained in largest plateau for the UF3SAT test sets 35
3.6 Size of open plateaus in UF-3-SAT instances 36
3.7 Number of plateaus in structured instances 37
3.8 Fraction of closed plateaus in structured instances 38
3.9 Size of closed plateaus in structured instances 39
3.10 Fraction of states contained in largest plateau for the structured instances 40
3.11 Size of open plateaus in structured instances 41
3.12 Summary statistics of the distribution of branching factors for the structured instances 46
3.13 Lower bound on plateau diameters for the structured instances 47
3.14 LMIN state fraction for the structured instances 50
3.15 Exit distance statistics for the structured instances 51

4.1 Summary statistics of the distribution of the out-degree of PCG nodes in UF-3-SAT instances 62
4.2 Summary statistics of the distribution of the out-degree of PCG nodes in structured instances 62
4.3 Summary statistics of the distribution of the average target depth of PCG nodes in UF-3-SAT instances 64
4.4 Summary statistics of the distribution of the maximum target depth of PCG nodes in UF-3-SAT instances 64
4.5 Summary statistics of the distribution of the average target depth of PCG nodes in structured instances 65
4.6 Summary statistics of the distribution of the maximum target depth of PCG nodes in structured instances 65
4.7 Correlation between $p(S \mid \Delta)$ and *lsc* for the UF-3-SAT test sets 68

4.8 $p(S \mid \Delta)$ values for the structured instances 69
4.9 Statistics from the distribution of α_{max} values for the UF-3-SAT test sets 71
4.10 Correlation between α_{max} and lsc for the UF-3-SAT test sets 72
4.11 Summary statistics for the distribution of ϵ_{max} values for the UF-3-SAT test sets 74

5.1 PCV results for SLS algorithms 86
5.2 PCV results for DPLL-type algorithms 90

A.1 Description of the random benchmark sets studied in this paper 108
A.2 Description of the structured benchmark sets studied in this paper 108

List of Figures

2.1 Example cumulative distribution functions 18

3.1 Fraction of closed plateaus sampled at each level 25
3.2 Distribution of plateau sizes (UF-3-SAT, 20 variables) 26
3.3 Distribution of plateau sizes (UF-3-SAT, 50 variables) 26
3.4 Distribution of plateau sizes (UF-3-SAT, 100 variables) 26
3.5 Distribution of the number of plateaus at each level over instances in the *uf30* test set 32
3.6 Scaling of the average number of plateaus with the number of variables in the UF3SAT test sets 33
3.7 Average fraction of closed plateaus at each level for the UF3SAT test suites 34
3.8 P^3_{max} from the instance in the *uf20* test set with median *lsc* 43
3.9 An example level 3 plateau from the `ais6` instance 44
3.10 An example level 5 plateau from the `parity8` instance 45
3.11 Distribution of branching factors for the 20, 50, and 100 variable UF-3-SAT test sets 46
3.12 Lower bound on plateau diameters for 20 variable UF-3-SAT test set 47
3.13 Distribution of the fraction of LMIN states for the 20, 50, and 100 variable UF-3-SAT test sets 49
3.14 Distribution of exit distances for the 20, 50, and 100 variable UF-3-SAT test sets 51

4.1 PCG for an easy *uf20* instance 58
4.2 PCG for the hardest *uf20* instance 59
4.3 PCG for the structured `anom` instance 61
4.4 Correlation between $p(S \mid \Delta)$ and *lsc* for the *uf30* test set 67
4.5 Distribution of α_{max} values for the UF-3-SAT test sets 70
4.6 Correlation between α_{max} and both *lsc* and $p(S \mid \Delta)$ for the *uf-30* test set 71
4.7 Shifted RLDs from instances from the *uf30* test set 73
4.8 Distribution of ϵ_{max} values for the UF-3-SAT test sets 74
4.9 Correlation between α_{max} and ϵ_{max} for the *uf30* test set. 75
4.10 RLD for hard instance from the *uf30* test set, and approximation by mixed exponential distribution 76

5.1 Distribution of criticality factors with respect to Novelty$^+$ for variables in the hardest *uf100* instance 85
5.2 Correlation between search cost and PCV fraction for the *uf100* benchmark set using the Novelty$^+$ algorithm 87
5.3 Correlation of criticality factors with respect to different SLS algorithms 88
5.4 Additivity of PCVAs using dynamic ranking for Novelty$^+$ 88
5.5 Correlation of criticality factors with respect to different algorithms on the bwa instance . . . 90
5.6 Correlation between average criticality factor and two search space features for Novelty$^+$ on the *uf25* test set 92
5.7 Partial, weighted PCG for the hardest *uf25* instance 93
5.8 Effect on the PCG when instantiating most critical variable in hardest *uf25* instance 94
5.9 Correlation between γ and criticality factor for Novelty$^+$ on instance bwa 96

B.1 Comparing Truth Tables, Binary Trees, and Binary Decision Diagrams 110

List of Algorithms

2.1 GSAT($\mathcal{F}$, *maxTries*, *maxSteps*) 16
2.2 WalkSAT/SKC($\mathcal{F}$, *maxTries*, *maxSteps*, p) 17
3.1 Levelset($\mathcal{F}$, *maxLev*) 29
3.2 Partition($Levelset^k$) 31

Acknowledgements

This thesis would have never been completed without the support of many people. First and foremost, I would like to thank Lisa Wong, whom I love deeply and truly, for being a constant source of motivation, for reminding me to take time to enjoy the small things in life, and for putting up with all of the late nights and weekend hours required to see this thesis through to completion. My thanks go out to my family for their constant interest in my research, even though I never took the time to give a satisfying explanation of it.

Many thanks are due to my supervisor Holger Hoos for his support and encouragement throughout my young academic career, for giving me the freedom to pursue all of those wild ideas, and for the midnight oil burnt during joint work. Thanks to both Holger Hoos and Nando de Freitas for improving this thesis with insightful comments and suggestions.

Finally, there are many people in the computer science department who have made my life easier and more enjoyable over the years. I would like to thank all of the members of the β-lab for their help and for interesting discussions, in particular Dave Tompkins, Kevin Leyton-Brown, Steph Durocher, and Dan Tulpan. Finally, I am indebted to Valerie McRae for all of her invaluable assistance.

Chapter 1

Introduction

Combinatorial optimisation problems are an important and well-studied class of problems, with applications in most areas of the computing sciences [GJ79]. Combinatorial optimisation problems can generally be described as computational problems in which all variables are discrete, and the goal is to find the best possible configuration among the set of variables, where quantifying what constitutes the "best" configuration is highly problem-dependent [PS82]. Because of their prominence, combinatorial optimisation problems have been the focus of extensive research for several decades. Unfortunately, many interesting combinatorial optimisation problems are known to be $\mathcal{NP}$-hard. Therefore, under the widely held assumption that $\mathcal{P}$!= $\mathcal{NP}$, any exact algorithm solving many of these problems must, in the worst case, require at least an exponential amount of time in the problem size.

Combinatorial optimisation problems (finding solutions that minimise a cost metric) are closely related to the corresponding decision problems (deciding whether there exists a solution with a given cost) [SW02]. In this thesis, we study a conceptually simple decision problem: the propositional satisfiability problem (SAT). Over the past decade, much progress has been made in solving both large and hard instances of the propositional satisfiability problem, in spite of the fact that the problem is $\mathcal{NP}$-complete. Two main classes of algorithms have emerged as the state-of-the art for SAT solving, stochastic local search (SLS) algorithms [HS04] and systematic search algorithms based on the Davis-Putnam procedure (DPLL) [DLL62]. The former are generally incomplete, *i.e.* they cannot decide unsatisfiability. Beyond this fundamental difference, however, there are certain classes of problem instances which can be handled more efficiently by either of the two classes of algorithms. For example, it seems that state-of-the-art DPLL type algorithms have a significant performance advantage on certain classes of SAT-encoded structured instances [SBH02].

Much of the progress in algorithm development has been facilitated by an improved understanding of the properties of SAT instances and of high-performance SAT algorithms (see, *e.g.*, [BS03, SBH02, KS03]). With this thesis, we hope to complement and extend existing approaches for understanding the complex interplay between properties of SAT instances and the behaviour of the algorithms used to solve them. The majority of the thesis focuses on the analysis of SLS algorithms, though we also discuss the implications of our findings on the design of systematic search algorithms and some of our analysis is done on both SLS and systematic algorithms (*c.f.* Sections 5.3 and 5.4). Furthermore, though the thesis exclusively stud-

ies the properties of SAT instances, most of the definitions and observations can be generalised to other combinatorial optimisation problems.

The remainder of this chapter is structured as follows. We first define the central problem of this thesis, and motivate its study. We then review previous work that has been done in this area, and comment on how the work relates to this thesis. Finally, we conclude this introduction with an outline of the remainder of the thesis.

1.1 Problem Definition and Motivation

The amount of research that has been undertaken towards an *understanding* of how and why SLS algorithms behave the way they do is surprisingly much less than the amount of work spent *developing* said algorithms. Much of the research has focused solely on pushing the bounds of the set of instances which may be feasibly solved, and on reducing the time required to solve those instances. While this is, and will continue to be, an important goal, many of the approaches have been fairly ad-hoc, with more emphasis on tweaking algorithms than in understanding the underlying reasons for their performance. The main problem addressed in this thesis is how to extend the current understanding the relationship between search space features and algorithm behaviour. The ultimate goal of this research is to use this knowledge to improve existing algorithms, and to provide insights into how better algorithms can be developed.

In order to simplify our search space analysis and abstract away irrelevant details, we restrict our analysis to the propositional satisfiability problem. The propositional satisfiability problem (SAT) can be stated as follows: Given a propositional formula $\mathcal{F}$, find an assignment of truth values to the variables in $\mathcal{F}$ under which $\mathcal{F}$ is satisfied, *i.e.*, equivalent to true, or determine that no such variable assignment exists. In the former case, $\mathcal{F}$ is called satisfiable, and in the latter case it is called unsatisfiable. Most widely studied SAT algorithms are restricted to formulae in conjunctive normal form (CNF), *i.e.*, to formulae of the form $\bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{k_i} l_{ij} \right)$ where the l_{ij} are literals, *i.e.*, variables or their negations.

As alluded to above, there are two main classes of state-of-the-art algorithms for solving SAT for CNF formulae: Stochastic local search (SLS) algorithms and systematic search algorithms based on DPLL-type algorithms. This thesis focuses almost exclusively on the study of SLS algorithms for SAT. Stochastic local search algorithms operate in the space of complete variable assignments, and iteratively make local modifications to the complete assignment while trying to optimise a given objective function.[1] In the SAT case, complete variable assignments correspond to a mapping from the binary truth values $\{\top, \bot\}$ to the set of variables of the formula $Var(\mathcal{F}) = \{x_1, x_2, \cdots, x_n\}$ (n is the number of propositional variables appearing in the formula). The local modifications generally correspond to flipping the truth values of a small set

[1] Note that in this context it is useful to regard SAT as an optimisation problem rather than a decision problem.

(typically of size 1) of variables.

One of the simplest SLS algorithms for SAT fitting this framework is a simple random walk: starting from a randomly chosen assignment, iteratively choose a random variable and flip the truth value assigned to that variable, and repeat this process until a solution is found or a given time bound is exceeded. It should be clear that the expected running time of such an algorithm on an instance with a single solution would be on the order of $O(2^n)$, which quickly becomes computationally intractable as n grows large.[2]

In practice, SLS algorithms employ some type of heuristic guidance. The most natural (and common) way to quantify the "quality" of a variable assignment is simply to use the number of unsatisfied clauses under the assignment. Thus, we define the objective function of each SLS algorithm as a function over variable assignments $g : \{\top, \bot\}^n \mapsto \mathbb{R}$. The triple consisting of the set of assignments, the neighbourhood relation, and the objective function comprises the *search space* of a problem instance with respect to a given SLS algorithm. Section 2.3 presents more rigorous definitions of these (and related) concepts.

Clearly, the behaviour of any SLS algorithm is highly related to underlying features of the search space. For example, if there are many local minima in the search space, this may (and in fact does) significantly impede the progress of SLS algorithms. We might expect that instances with many solutions are easier than instances with few — or that instances with a large number of solutions that are distributed uniformly throughout the space of all assignments are easier than instances with the same number of solutions, but in which all of the solutions appear in a tight cluster. The central problem of this thesis is to understand how these types of search space features affect algorithm behaviour.

1.2 Existing Work

We now briefly review the most relevant literature related to the search space analysis of SAT instances. Many different aspects of the search spaces have been analysed, so we attempt to group the research into general categories. Some of the research follows different approaches while attempting to answer the same questions, while other research is very closely related to the work done in this thesis. The remainder of this section describes the existing work, and characterises the relationship between the existing work and the work done in this thesis.

1.2.1 Number of Solutions

One search space feature that is known to affect the performance of SLS algorithms for SAT is the number of solutions. In 1996, Clark *et al.* investigated the correlation between the number of solutions and the

[2]Note that non-trivial upper bounds on running time (*i.e.* bounds lower than $O(2^n)$) have been established for other simple SLS algorithms. See, *e.g.*, [Pap91, Sch99].

hardness of an instance [CFG+96]. For test instances, they used 1000 soluble random 3-SAT problems with 100 variables and 430 clauses (for a description of random 3-SAT problems, refer to Appendix A), and one of the first SLS algorithms introduced for SAT, GSAT.[3]

In order to quantify the hardness of each instance, Clark *et al.* ran GSAT 100 times on each instance and used the fraction of successful runs as a measure of local search cost. Note that because GSAT is a very greedy algorithm, it can get permanently "stuck" in local minima regions of the search space, which is why not all of the runs were successful. A strong but noisy positive correlation was found between the number of solutions and this estimated local search cost, indicating that (not surprisingly) instances with fewer solutions tend to be harder than instances with many.

Hoos extended this analysis in 1998 using an improved methodology [Hoo98]. Hoos used the GWSAT algorithm [SK93], which is a more robust, better performing variant of GSAT, and which can be guaranteed to find a solution (if one exists) given enough time [Hoo98]. He measured average local search cost (*lsc*) by measuring the average number of GWSAT steps required to find a solution rather than simply measuring the fraction of successful runs (we will use this definition of local search cost throughout this thesis).

Hoos also found a strong positive correlation between the number of solutions and *lsc*. In addition, he noted that while for instances having high numbers of solutions the variability in *lsc* is very small, it increases considerably for instances with lower numbers of solutions. Hoos also notes that with growing problem size, the dependency of *lsc* on the number of solutions gets weaker.

While these results are significant, using the number of solutions to predict local search cost has a number of drawbacks. First, it is obviously very computationally expensive to count the number of solutions — counting all the solutions to a SAT instance is in the $\#\mathcal{P}$-C complexity class [Val79], making this approach suitable only for *a posteriori* analyses. Note that this observation applies to many of the other metrics discussed later in the section. Second, and more problematic, we have found that even if we consider only single solution instances, there can still be a very large range in *lsc*. This agrees with Hoos' findings for small numbers of solutions, and suggests that there must be other search space factors affecting the hardness of instances. This thesis extends these results by presenting a model of the search space which can account for this large variance in *lsc*, even when accounting for the number of solutions.

1.2.2 Solution Clustering

In 1997, Parkes extended the work done by Clark *et al.* by discovering that, for random 3-SAT instances near the phase-transition region [CKT91], solution states are found in clusters [Par97]. He studied this clustering phenomenon indirectly, by counting the unary prime implicates, or UPI's (*i.e.* literals that are logically entailed by the formula) of the instances. Parkes observed that for clause-to-variable ratios α less

[3] GSAT was conceived by Selman, Levesque, and Mitchell [SLM92] in 1992, and is described in detail in Section 2.3.

than the critical clause to variable ratio α_c,[4] most instances have few UPI's but as the transition is made to $\alpha > \alpha_c$, most instances have many UPI's. This phenomenon was observed for problem sizes ranging from 50 to 300 variables.

Parkes then described how instances that have many UPI's have no solutions outside of the region of the search space that is compatible with the UPI's, and that in this region there are few constraints and the density of solutions is high. He refers to such a region as a cluster, and proceeds to show that WalkSAT performs badly on these "clustered instances" (WalkSAT is another well known SLS algorithm for SAT — see [SKC93, SKC94] as well as Section 2.3).

Finally, Parkes conjectured that the addition of a single clause to a satisfiable instance could convert a "solution cluster" into a "failed cluster", which would be a very attractive region of the search space, from which it is impossible to reach a solution. This was strictly a conjecture, and he did not provide a convincing argument for it. In this thesis, we confirm Parkes' conjecture by empirically demonstrating that such search space features do in fact exist, and that they have a very significant impact on the behaviour of local search algorithms.

Parkes' findings seem to indicate that the number of solutions should not have such a dominating effect on *lsc* as was previously believed. If all of the solutions of most instances are found in a single cluster, then the difficulty faced by SLS algorithms is to find the solution cluster. Because the solution density is high in these clusters, the number of solutions should not be particularly relevant. This apparent contradiction with earlier results is addressed in the following section.

1.2.3 Backbone Fragility

Singer, Gent and Smaill studied the effect of the backbone of SAT instances on the *lsc* of WalkSAT [SGS00]. The backbone of a SAT instance is simply the set of UPI's. In this study, they used the WalkSAT algorithm and random 3-SAT instances with 100 variables and a clause to variable ratio varying from 4.0 to 4.5. Their findings are closely related to those presented in the previous section.

Singer *et al.* found that the number of solutions is highly correlated with *lsc* for instances with small-backbones, but is much less relevant for large-backbone instances. To explain this they conjectured that for small-backbone instances, finding the backbone is straightforward and the main difficulty is encountering a solution once the backbone has been satisfied. In this case, the density of solutions in the region satisfying the backbone is then important. For larger backbone sizes, the main difficulty is satisfying the backbone and the number of solutions becomes less important.

This result is also related to another of Hoos' observations in [Hoo98]. Hoos found that the correlation

[4] The critical clause to variable ratio is more commonly known as the phase transition point for random 3-SAT instances, and is the point at which a randomly chosen instance is satisfiable with probability 0.5[CKT91].

between the number of solutions and local search cost becomes weak in the over-constrained region (the over-constrained region consists of all soluble SAT instances having a clause-to-variable ratio $\alpha > 4.26$). Singer *et al.* observed that large-backbone instances dominate the over-constrained region. Since the correlation between number of solutions and *lsc* is not strong for large-backbone instances, this implies that most instances in the over-constrained region will not exhibit this correlation.

Singer, Gent, and Smaill also introduced a novel metric that they demonstrated was highly correlated with *lsc*. For a fixed backbone size, they generated 1000 WalkSAT runs, which terminated as soon a level 5 or lower state was found (please refer to the next chapter for a definition of the level of a state — in short, the level of an assignment is the number of clauses which are unsatisfied by the assignment). For each of these low level states, they measure the Hamming distance to the nearest solution, s, and find the average, which they label $hdns(T_{f5}, s)$. They find a strong positive correlation between $hdns(T_{f5}, s)$ and *lsc* when backbone size is fixed.

Motivated by these observations, Singer *et al.* make some conjectures about the search spaces of the instances they were examining. They conjecture that the "quasi-solution area" (this corresponds in this thesis to the set of all low level plateaus,[5] defined in Chapter 3) form interconnected areas of the search space such that WalkSAT can always reach a solution from them without moving to higher-level states. Further, they conjecture that in high-cost instances, this quasi-solution area extends to parts of the search space that are Hamming-distant from solutions, whereas in lower cost instances the area is less extensive.

Our results indicate that this is indeed the case. However, Singer, Gent, and Smaill claim that variance in $hdns(T_{f5}, C)$ accounts for almost all of the variance in cost at three different backbone sizes. We do not believe that such a simple measure can account for all of the variance in search cost. We have found that the hardest instances seem to be those that have very attractive closed plateau regions in combination with few low-level plateau regions that are well connected to solutions. We believe that while $hdns(T_{f5}, C)$ does seem to show promise as a method for estimating local search cost, a complete understanding of SLS algorithm behaviour requires a more detailed analysis of the respective search spaces.

1.2.4 Local Minima

Yokoo was interested in how the search space of random 3-SAT instances changes through the phase transition [Yok97]. In particular, he investigated the effect that adding clauses had on the number of local minima states in the search space. For his analysis, he used a deterministic, greedy, lexographical tie-breaking algorithm, and 20 variable random 3-SAT instances. In this analysis, Yokoo exhaustively explored the entire search space of the problem instances, which explains why his analysis was restricted to such small instances.

[5] Informally, a plateau is a set of connected states all having the same objective function value.

Yokoo pointed out a seemingly paradoxical situation: the average number of solutions to the problems decreases as clauses are added, but soluble problems at the phase transition are harder than those beyond it. To explain this, Yokoo measured the average ratio of "solution-reachable" states, where solution-reachable states are states having the property that if his (deterministic) algorithm was initialized in those states, it would eventually reach a solution.

Yokoo found that by adding more clauses, the ratio of solution-reachable states actually increased beyond the phase transition. Next, he found that the size of closed plateaus decreases as more clauses are added, while the number of closed plateaus increases as clauses are added up to the phase transition point, and then slowly decreased. Thus, Yokoo conjectured that the reason problems get easier as more clauses are added is that the added clauses either break up closed plateau regions, or eliminate them altogether. Finally, Yokoo repeated the above analysis on 12 variable 3-colouring problems, with similar results.

1.2.5 Effect of Search Space Structure on Algorithm Behaviour

In 1993, Gent and Walsh performed an analysis of the the behaviour of GSAT on 500 variable random 3-SAT problems [GW93]. Their experiments consisted of measuring how the level of visited states and "poss-flips" (poss-flips is the number of neighbours of a state which all have the best possible score) vary during a GSAT try (a single try is a series of hill-climbing steps, and ends when a maximum number of steps have been completed or a solution is found).

Gent and Walsh found that the level initially decreases rapidly, but then flattens out until much search is needed to improve the level. To explain this, they show that GSAT tries can be divided into a series of distinct phases, which they label $H_0, \cdots, H_m$. Each phase H_i corresponds to the portion of the search trajectory in which a typical variable flip results a net increase of i satisfied clauses. GSAT starts in phase H_m, then moves to phase H_{m-1}, etc. until the search is in the H_0 phase. When the search is in the H_0 phase, they refer to this as the plateau-search phase because almost every move is 'sideways', with no change in level. Early search phases have a short duration, but the level decreases rapidly.

As we will see, the results from our plateau-based search space analysis agree with these results, and extend them in a non-trivial way. We find that the search space of random 3-SAT instances consists of a single, large plateau region which is very easy to escape from at high levels. In regions such as this, a greedy algorithm such as GSAT makes rapid progress and the level decreases very quickly. In addition, we find that at high levels in the search space, many of the exits from these plateau regions have neighbours which are a number of levels lower than the current state, but as the level decreases, this difference in the level of neighbours tends to decrease. These results are discussed further in Chapter 4.

1.2.6 Plateau Distribution

In early SLS algorithms for SAT, a restart mechanism was used to address the problem of getting stuck in closed plateau regions. A fixed number of greedy hill-climbing steps were performed, and then the algorithm would be reinitialised to a random assignment. In 1995, Hampson and Kibler investigated how much time should be spent searching plateau regions before restarting [HK95]. To do this, they used a greedy hill-climbing algorithm that tracked the number of consecutive sideways moves made on a given plateau region, and they also used breadth-first search (BFS) to exhaustively search plateau regions to a limit of 100 000 states. They ran their experiments on random 3-SAT instances with 64 to 512 variables and a clause to variable ratio $\alpha = 4.3$.

Hampson and Kibler made a number of discoveries that are significant to this study. They demonstrated that the size of plateau regions increases with increasing level. However, they also note that the density of open states[6] in the plateau regions increases rapidly with increasing level. These observations are confirmed by our analysis in Chapter 3.

Finally, Hampson and Kibler found that the size of lower level plateau regions seemed to grow exponentially with the number of variables (n), while the density of open states in them decreased inversely with n, and that the number of lower level plateau regions increased linearly with n. From this, they conjectured that if the search could be initialized randomly from states in lower level plateau regions, then the average search time would increase linearly with n. To our knowledge, this conjecture was never tested, and remains an open question.

In 1997, Frank, Cheeseman, and Stutz performed a fairly detailed analysis of the search space of Random 3-SAT instances, with a focus on finding the distribution of plateau region sizes, as well as a preliminary examination of the structure of these plateau regions [FCS97]. They used GSAT and random 3-SAT problems with 100 variables and $\alpha = 3.8$ to 4.6 for their analysis.

Frank *et al.* generated 1000 instances with $\alpha = 3.8, 3.9, \ldots, 4.5, 4.6$, and ran GSAT once per instance until it encountered a state at each level $1 \ldots 5$. Then, each of these states were expanded using breadth-first-search (BFS) to find the entire plateau containing the starting state, and they report properties such as the number of plateaus and how large they are. This work is very closely related to our work in Chapter 3, where we attempt to replicate their experiments as well as extend their analysis in interesting directions. Interestingly, our data differs from theirs dramatically, though the experiments were intended to be identical. Because of these differences, we draw different conclusions from our experiments, and are able to explain conflicting observations that Frank *et al.* were not able to. Further discussion of these results are left for Chapter 3.

[6] An *open* state is a state having at least one neighbours at a lower level.

1.2.7 Predicting Local Search Cost

There has been a significant amount of research invested in attempting to find search space features which are easy to measure, and which can also be used to predict the local search cost (*lsc*) of problem instances. We have already discussed measures which correlate well with *lsc*, such as the number of solutions [CFG+96] or $hdns(T_{f5}, C)$ [SGS00], but these are *a posteriori* measures for explaining algorithm performance. The following measures are computationally cheap to compute, and can be unsed *a priori* to estimate the expected local search cost.

In 1998, Holger Hoos contributed to the literature with a chapter of his Ph.D. Dissertation devoted to the search space structure of SAT instances [Hoo98]. The first section of the chapter is devoted to analysing how the number of solutions affects local search cost, and is discussed earlier in this paper. The remaining sections are devoted to novel metrics that Hoos developed for measuring search space features.

The first metric that Hoos introduces is the standard deviation of the number of unsatisfied clauses, *sdnclu*. This is a measure of the variance of the objective function. The motivation behind this is the intuition that the ruggedness or smoothness of the objective function across the search space should have an effect on the behaviour of SLS algorithms.

Hoos measures *sdnclu* by exhaustive sampling for Random 3-SAT problems of size $n = 20$, but for $n = 50$ and $n = 100$ he resorts to randomly sampling 100 000 states from the search space. One concern that we have with this random sampling method is that randomly sampling states from large instance will almost exclusively find open states at relatively high levels. It is well known that SLS algorithms spend very little time searching these higher level states, so it is not immediately clear why properties of these higher level states will have a significant impact on SLS behaviour in general.

Nevertheless, Hoos does report a noisy negative correlation between *sdnclu* and local search cost, as well as a positive correlation between *sdnclu* and number of solutions for 20, 50, and 100 variable Random 3-SAT instances. Hoos also finds that the correlation is stronger for under-constrained instances, and gets weaker as the number of clauses increases. Thus, *sdnclu* must be capturing global search space features which affect SLS algorithm performance, even though only high level states are being sampled.

Next, Hoos compares *sdnclu*, local search cost, and number of solutions of structured instances to Random 3-SAT instances of equivalent size. He finds that his observations seem to hold for Random Graph Colouring problems and Blocks World Planning problems, but he finds inconsistent results when he examines All-Interval-Series problems. Note that this is one of the only pieces of existing work which has studied the search space properties of structured problem instances. In this thesis, we also give equal precedence to the study of random and structured instances whenever possible.

Following this, Hoos introduces a set of state types and investigates their distributions. Hoos defines the following state types: BPLAT states are those that have neighbours at higher, lower, and equal levels.

SLOPE states only have neighbours at higher or lower levels. LMIN states have neighbours at the same level or higher. LMAX states have neighbours at the same level or lower. SLMIN and SLMAX states are LMIN and LMAX states respectively, which do not have any neighbours at the same level. Finally, IPLAT states are those that only have neighbours at the same level.

By exhaustive sampling of Random 3-SAT instance with $n = 20$, Hoos measures the exact state distributions. He finds that almost all states are BPLAT, followed by a much smaller number of SLOPE states, and even fewer LMIN and LMAX states. Hoos found very few SLMIN or SLMAX states, and no IPLAT states.

Because it is infeasible to exhaustively enumerate the states of larger instances and random sampling finds almost exclusively BPLAT states, Hoos then sampled the state distribution along GWSAT trajectories. The results of this analysis show that GWSAT is very strongly attracted to LMIN states (which is not surprising because GWSAT uses hill-climbing to move through the search space), and that the only types of states that significantly affect GWSAT's behaviour are BPLAT and LMIN states. Hoos also performed this analysis on structured instances, with similar results.

Finally, Hoos analyses another novel metric, the branching factor of local minima states, *blmin*. The motivation here is that it is well known that local minima (or closed) states have a major impact on SLS performance, and *blmin* is a way to study these states. Because local minima states only account for a small fraction of the states in the search space, they cannot be randomly sampled so instead Hoos samples *blmin* along GWSAT trajectories. He again uses Random 3-SAT, Graph Colouring, Blocks World Planning, and All-Intervals-Series instances.

Hoos finds a negative correlation between *blmin* and local search cost, and a positive correlation between *blmin* and both *sdnclu* and number of solutions. Hoos notes that the *blmin* values are fairly low, indicating that the plateau structures are quite "brittle" and that most neighbours of local minimum states are at higher levels. Along with his earlier observation that IPLAT states do not seem to exist, this can help to explain the effectiveness of simple escape strategies such as random walk.

Finally, Nudelman *et al.* recently investigated how features of SAT instances can be used to build models useful for predicting search cost [NLBD+04]. Their approach is to use machine learning to automatically determine which of a set of given features are useful in building compact models for predicting local search cost. They study a large set of random 3-SAT instances with varying clause to variable ratios. Though they study only complete algorithms, a similar analysis could theoretically be performed using stochastic local search algorithms. Our work is related to Nudelman *et al.*'s in several ways. Firstly, while they currently only consider features of the instances which can be computed efficiently (the analyses we perform are very computationally expensive), it may be possible to incorporate some of the features coming out of our work in order to improve their models. Secondly, the features automatically determined to be useful in predicting the search cost of an instance are obviously interesting objects of study. It should be possible to take those

interesting features, and analyse them carefully to determine the relationship between those features, and features of the search space of the instance.

1.3 Thesis Outline

The remainder of the thesis is structured as followed. In the next chapter, we introduce necessary concepts and definitions required for the remaining chapters. Chapter 3 studies the distribution of plateaus in the search space, and how the properties of these plateaus are related to SLS algorithm behaviour. Chapter 4 discusses the connectivity of plateaus, illustrates how we can build simplified models of the search space that aid in the understanding of both search space structure, and demonstrates how this structure affects the performance of SLS algorithms. Chapter 4 also characterises and identifies and studies important search space features that have a negative impact on SLS algorithm performance - so called "traps" in the search space. Chapter 5 takes a slightly different direction, and identifies propositional variables that play a critical role when solving SAT instances. We characterise the relationship between critical variables and search space structure, which gives novel insights into the local search paradigm. Finally, we summarise our main results and suggest interesting directions for future work in Chapter 6. The two appendices at the end of the thesis give a description of the test-sets used in the thesis, and an introduction to binary decision diagrams.

Chapter 2

Preliminaries

In this chapter, we introduce many of the notations and definitions used throughout the thesis. Many of our definitions are meant to be functionally equivalent to those presented in [Hoo98, HS04], though they may be presented differently in this thesis. Throughout the thesis, the following conventions are followed. Predicates are capitalised and italicised (*E.g.* $LMIN(s)$). Sets are italicised, with the first letter of each word in the name capitalised, and optional subscripts and superscripts allowed (*E.g.* $Levelset^{k}_{\mathcal{F}}$). Functions are also capitalised and have the first letter of each word in the name capitalised, but are required to be followed by a list of parameters in parentheses and are not allowed to have subscripts or superscripts in the name (*E.g.* $PickTwoClauses(\mathcal{F})$). Literals are italicised, and must be a single lowercase symbol with optional subscripts but only primes allowed as superscript (*E.g.* l'_{ij}). Finally, we use $S \backslash P$ and $\#S$ to denote the set theoretical difference of S and P, $\{x \mid x \in S \wedge x \notin P\}$, and the cardinality of S, respectively.

The remainder of this chapter is structured as follows. First, we formalise the definition of propositional formulae and related definitions such as variable assignments and satisfiability. Then, we formalise many of the definitions used to define the search space of SAT instances. Following this, we introduce Stochastic Local Search algorithms for SAT and describe two reference algorithms used throughout the thesis. Finally, we describe the empirical methodology used throughout the thesis.

2.1 Propositional Formulae

A propositional formula is, in its most general sense, a well-formed formula defined over a number of Boolean variables $\{x_1, x_2, \cdots, x_n\}$ and the standard operators $\{\neg, \wedge, \vee, \rightarrow, \cdots\}$. Throughout this thesis, we focus solely on propositional formulae of a syntactically restricted class, called Conjunctive Normal Form. The next definition formalises this notion.

Definition 2.1. (Propositional Formula, Conjunctive Normal Form)
A *propositional variable* is an element of the set $V = \{x_i \mid i \in \mathbb{N}\}$, and can be assigned a value from the set of *truth values* $C = \{\top, \bot\}$. A *literal* is either a propositional variable x_i, or its negation $\neg x_i$. Given a literal l, $Variable(l)$ represents the associated variable.

A *propositional sentence* is a sentence over the alphabet $P := V \cup C \cup O \cup \{(,)\}$, where O represent the *propositional operators* $\{\neg, \wedge, \vee\}$.[1] A propositional sentence F_S is in *Conjunctive Normal Form* if it can be written as a conjunction($\wedge$) of disjunctions($\vee$) of literals:

$$F_S = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{k_i} l_{ij} \right)$$

Here, each disjunction $\bigvee_{j=1}^{k_i} l_{ij}$ is called a *clausal sentence* involving k_i literals.

Because conjunction and disjunction are associative, we can disregard the order of the clausal sentences and of literals within a single clausal sentence. Thus, for each clausal sentence $\bigvee_{j=1}^{k_i} l_{ij}$ we define the set $C(i) := \bigcup_{j=1}^{k_i} l_{ij}$, and call this set a *clause*. The number of literals in clause C is given by $Length(C) := \#C$. We also define the set $\mathcal{F} := \bigcup_{i=1}^{m} \{C(i)\}$, and call this set a *CNF formula*. The number of clauses in $\mathcal{F}$ is given by $Clauses(\mathcal{F}) := \#\mathcal{F}$. Finally, a CNF formula $\mathcal{F}$ is in k-CNF if and only if $\exists k \in \mathbb{N}\ \forall C \in \mathcal{F} : Length(C) = k$. Note that, according to this definition, clauses cannot contain duplicate literals, and CNF formulae cannot contain duplicate clauses — this is not a serious restriction, since both duplicate literals and clauses are redundant. □

We should note that restricting our attention to this subset of the propositional formulae is without loss of generality, since any propositional formula may be transformed into an equivalent formula in CNF. In fact, it is possible to convert any propositional formula into an equi-satisfiable CNF formula with only a polynomial increase in the size of the formula [PG86, dlT90].

Given a propositional formula with n variables, it should be easy to see that there are exactly 2^n different ways to assign binary values to the variables. The propositional formula is defined in such a way that each one of those assignments evaluates to either true or false; the goal of the SAT problem is to find an assignment that evaluates to true for a given formula. The following definition formalises the concepts of variable assignments and the method of evaluating the value of each assignment under a given formula.

Definition 2.2. (Assignment)

The *variable set* $Var(\mathcal{F})$ *of formula* $\mathcal{F}$ is defined as the set of all variables appearing in $\mathcal{F}$. More formally:

$$Var(\mathcal{F}) = \{x_j \mid \exists C \in \mathcal{F} : x_j \in C \vee \neg x_j \in C\}$$

A *variable assignment of formula* $\mathcal{F}$ is a mapping $a : Var(\mathcal{F}) \mapsto \{\top, \bot\}$ of the variable set of $\mathcal{F}$ to the truth values. The set of all possible variable assignments is denoted by $Assign(\mathcal{F})$.

[1] Note that O may be extended to include other operators such as $\rightarrow$, $\leftrightarrow$, and $\otimes$, but that these additional operators are redundant and are useful only for writing formulae more succinctly.

The *value* $Val_a(\mathcal{F})$ *of formula* $\mathcal{F}$ *under assignment* a is defined inductively:

$$
\begin{aligned}
Val_a(b) &:= b && b \in \{\top, \bot\} \\
Val_a(\neg b) &:= \begin{cases} \top \text{ if } b = \bot \\ \bot \text{ if } b = \top \end{cases} && b \in \{\top, \bot\} \\
Val_a(x_i) &:= Val_a(a(x_i)) && x_i \in V \\
Val_a(\neg x_i) &:= Val_a(\neg a(x_i)) && x_i \in V \\
Val_a(C) &:= \begin{cases} \top \text{ if } \exists\, l \in C : Val_a(l) = \top \\ \bot \text{ otherwise.} \end{cases} && C \in \mathcal{F} \\
Val_a(\mathcal{F}) &:= \begin{cases} \top \text{ if } \forall\, C \in \mathcal{F} : Val_a(C) = \top \\ \bot \text{ otherwise.} \end{cases}
\end{aligned}
$$

Note that $Val_a(\mathcal{F}) = \top$ if and only if all clauses contain at least one literal l that is consistent with a, *i.e.* $Val_a(l) = \top$. □

Finally, we are in a position to formally state the Propositional Satisfiability Problem. The Propositional Satisfiability Problem is to determine for a given Propositional formula if there exists an assignment a that *satisfies* the formula:

Definition 2.3. (Satisfiability)
A variable assignment a is a *model of formula* $\mathcal{F}$ if and only if $Val_a(\mathcal{F}) = \top$; in this case we say that a *satisfies* $\mathcal{F}$. A formula $\mathcal{F}$ is *satisfiable* if and only if there exists at least one model of $\mathcal{F}$. □

2.2 Search Space Components

Given a formula $\mathcal{F}$, we say that the set of variable assignments, $Assign(\mathcal{F})$, forms the *search space* $\mathcal{S}(\mathcal{F})$. We also define a *neighbourhood relation* on $\mathcal{S}(\mathcal{F})$, $\mathcal{N}(\mathcal{F}) \subseteq \mathcal{S}(\mathcal{F}) \times \mathcal{S}(\mathcal{F})$. For brevity, we typically do not include the reference to $\mathcal{F}$ when referring to $\mathcal{S}(\mathcal{F})$ and $\mathcal{N}(\mathcal{F})$, but implicitly assert the dependence on the problem instance, and write $\mathcal{S}$ and $\mathcal{N}$, respectively.

Taken together, $\mathcal{S}$ and $\mathcal{N}$ form the *search graph* $G_N = (\mathcal{S}, \mathcal{N})$. G_N determines the space of assignments that a given SLS algorithm is searching, as well as the transitions available to the algorithm. Typically, however, SLS algorithms do not perform random walks on this graph; they are generally guided by some form of greedy heuristic. This heuristic information is captured in the *evaluation function* $g(\mathcal{F}) : \mathcal{S}(\mathcal{F}) \mapsto \mathbb{R}$, mapping search states to real numbers while ensuring that global optima in g correspond to satisfying assignments. In most SLS algorithms for SAT, we simply define $g(\mathcal{F}, a) := \#\{c \in \mathcal{F} \mid Val_a(c) = \bot\}$, that is, the evaluation function value of an assignment is just the number of clauses that are not satisfied by that

assignment. Minimising g therefore corresponds to satisfying more clauses, and when $g(\mathcal{F}, a)$ is 0 there are no clauses unsatisfied by a, and $\mathcal{F}$ is satisfied.

Finally, combing $\mathcal{S}$, $\mathcal{N}$, and g results in the *search landscape* $L = (\mathcal{S}, \mathcal{N}, g)$. The search landscape fully defines the search space of an instance, and can be conceptualised as a graph in which nodes represent search states, edges represent neighbouring states, and each state has a "fitness" values associated with it. The goal then, of any SLS algorithm, is to perform a (biased) walk in the search landscape attempting to find a state with a fitness value of zero. For a SAT instance involving n variables, the search landscape can be conceptualised as an n-dimensional Boolean hypercube, with the evaluation function value of each state associated with the corresponding vertex. Chapters 3 and 4 of this thesis exclusively study properties of the search landscape in an attempt to understand how and why SLS algorithms behave the way they do on these landscapes.

2.3 SLS Algorithms for SAT

For our purposes in this thesis, we will consider only a small (but well-studied) subset of SLS algorithms for SAT. In order to simplify the analysis done later in the thesis, we restrict our attention to SLS algorithms that search the landscape defined by $\mathcal{S}(\mathcal{F}) = Assign(\mathcal{F})$, $\mathcal{N}(\mathcal{F})$ corresponding to all pairs of states differing in the value of only one variable, and $g(\mathcal{F}, a) := \#\{c \in \mathcal{F} \mid Val_a(c) = \bot\}$. Most modern state-of-the-art SLS algorithms for SAT are compliant with these restrictions, so we are still studying a very interesting class of algorithms. Additionally, much of the theory that we will develop still applies to other types of algorithms (for example, those which use larger local neighbourhoods). For a more general and comprehensive description of SLS algorithms in general, the reader is referred to [Hoo98, HS04].

In this section, we give a brief overview of two "canonical" local search algorithms for SAT. These algorithms are both very well-studied, and also form the basis for many other (better performing) algorithms. The two algorithms are GSAT [SLM92], and WalkSAT/SKC [SKC93, SKC94].

Pseudo-code for the well-known GSAT algorithm is shown in Algorithm 2.1. The algorithm is quite simple. Each run of the algorithm consists of a number of *tries*, each of which is commenced by choosing a random starting assignment (*c.f.* line 3).[2] Once the starting point has been chosen, a fixed number of *steps* are performed. In each step, a variable is selected (*c.f.* line 8) and flipped (*c.f.* line 9). The variable is chosen randomly from all variables that minimise the heuristic function $score^{GSAT}$, defined as $score_a^{GSAT}(v) = g(\mathcal{F}, a') - g(\mathcal{F}, a)$, where a is the current assignment, and a' is the assignment with variable v flipped. This process is repeated until either a solution is found, or the maximum number of steps has been reached.

[2]Note that throughout the thesis, unless states otherwise, random choices are made uniformly over the given set.

Algorithm 2.1: GSAT($\mathcal{F}$, *maxTries*, *maxSteps*)

Input : CNF Formula $\mathcal{F}$, Integers *maxTries* and *maxSteps*
Output: Satisfying assignment or "timeout"

```
1  begin
2      for try := 1, ··· , maxTries do
3          a := randomly chosen element of Assign(F)
4          for step := 1, ··· , maxSteps do
5              if Val_a(F) = ⊤ then return a
6              else
7                  B := {v ∈ Var(F) | ∀v' ∈ Var(F) : score_a^GSAT(v) ≤ score_a^GSAT(v')}
8                  v := randomly chosen element of B
9                  a := a with variable v flipped
10             end
11         end
12     end
13     return "timeout"
14 end
```

As previously mentioned, the basic GSAT algorithm forms the basis for many other SLS algorithms, including (but by no means limited to) GWSAT [SK93], GSAT/TABU [MSK97, MSG97], HSAT [GW93] and HWSAT [GW95], as well as very powerful dynamic local search algorithms such as GLS [MT00], DLM [SW97], and SAPS [HTH02].

Another interesting and influential class of algorithms is based on the WalkSAT framework. Pseudo-code for the archetypical algorithm in this framework, WalkSAT/SKC, is shown in Algorithm 2.2. Structurally, the algorithm is quite similar to GSAT; both algorithms iterate over a fixed number of *tries* and *steps*, randomly reinitialising at the start of every try, and choosing a variable to be flipped in every step. The major difference between the two algorithms, and the defining feature of the WalkSAT framework, is that WalkSAT/SKC uses a two-tiered variable selection mechanism (*c.f.* lines 7 – 16). To select a variable to flip, a currently unsatisfied clause C is chosen and the candidate variable is then restricted to those appearing in C. This guarantees that at the very least, clause C will be satisfied after the variable flip. The set B (*c.f.* line 10) contains all variables that minimise the heuristic function $score^{SKC}$, defined as $score_a^{SKC}(v) = \#\{C \in \mathcal{F} \mid Val_a(C) = \top \wedge Val_{a'}(C) = \bot\}$, where a' is equivalent to the assignment a, with the value of variable v flipped. Note that this heuristic function only counts a negative contribution for the clauses that become unsatisfied when a variable is flipped; it does not count a positive contribution for the clauses that become satisfied (*i.e.* it only assesses the "damage" done by a variable flip). Lines 11 – 16 show how a the variable is actually chosen from among the sets. If a variable can be chosen that does not unsatisfy any new clauses, it is always chosen (*c.f.* line 11). Otherwise, with probability p, the variable is chosen randomly from all of those in the clause, and with probability $1 - p$ the variable is chosen randomly from B (the set of heuristically best variables).

Algorithm 2.2: WalkSAT/SKC($\mathcal{F}$, *maxTries*, *maxSteps*, p)

Input : CNF Formula $\mathcal{F}$, Integers *maxTries* and *maxSteps*, Probability p
Output: Satisfying assignment or "timeout"

begin
  for $try := 1, \cdots, maxTries$ **do**
    $a :=$ randomly chosen element of $Assign(\mathcal{F})$
    for $step := 1, \cdots, maxSteps$ **do**
      if $Val_a(\mathcal{F}) = \top$ **then return** a
      else
        $U := \{C \in \mathcal{F} \mid Val_a(C) = \bot\}$
        $C :=$ randomly chosen element of U
        $C_v := \{v \in Var(\mathcal{F}) \mid \exists l \in C : v = Variable(l)\}$
        $B := \{v \in C_v \mid \forall v' \in C_v : score_a^{SKC}(v) \leq score_a^{SKC}(v')\}$
        if $\forall v \in B : score_a^{SKC}(v) = 0$ **then**
          $v :=$ randomly chosen element of B
        else
          $v :=$ randomly chosen element of $\begin{cases} C_v \text{ with probability } p \\ B \text{ with probability } 1-p \end{cases}$
        end
        $a := a$ with variable v flipped
      end
    end
  end
  return *"timeout"*
end

WalkSAT/SKC has been shown to outperform GSAT (and variants of it) on many problem classes [SKC94, Hoo98]. As with GSAT, a large number of SLS algorithms have been based on the WalkSAT framework. Some notable examples include WalkSAT/Tabu [MSK97, MSG97], and Novelty$^+$ [Hoo99].

2.4 Empirical Methodology

This thesis is very empirical in nature. Throughout the thesis, we perform experiments, typically by running SLS algorithms repeatedly on distributions of test instances, and report empirical results. Because SLS algorithms are inherently randomised, their performance on a given instance can only be characterised by a distribution of run-times for that particular instance. Hoos and Stützle have developed a methodology for empirically analysing SLS algorithms [HS98, HS04], and we follow their approach in this thesis.

To study the distribution of SLS algorithm run-times, we use cumulative distribution functions (CDFs). A CDF is defined simply as the probability that a given random variable X takes a value less than or equal to x, *i.e.* $cdf(x) = Pr[X \leq x]$. For discrete distributions (which we exclusively encounter in the empirical

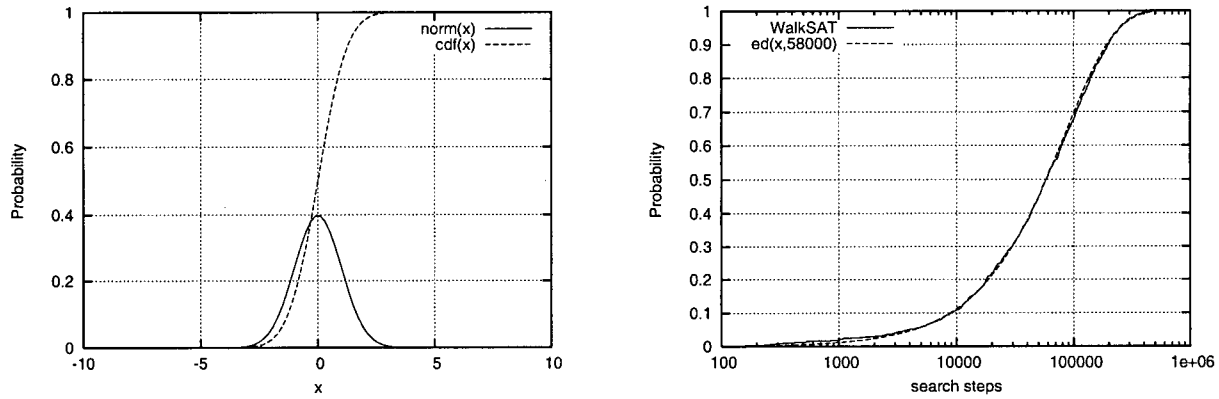

Figure 2.1: Example cumulative distribution function. Left: The standard normal probability density function, labelled $norm(x)$, and the associated CDF, labelled $cdf(x)$. Right: An empirical CDF, the RTD of WalkSAT/SKC on a hard random instance, as well as the best fitting exponential distribution — see text for details.

studies throughout this thesis), this becomes:

$$cdf(x) = \sum_{\{y \mid y \leq x\}} Pr[X = y] \tag{2.1}$$

The left pane of Figure 2.1 illustrates a simple CDF for the standard normal probability density function. For any x, if we would like to find $Pr[X \leq x]$, we simply find the intersection of the CDF with the desired x value. For example, at $x = 0$, we get $Pr[X \leq x] = 0.5$. We use CDFs throughout this thesis, and will assume that the reader is familiar with them.

CDFs are useful for analysing the distribution of run-times of an SLS algorithm on a given instance. To achieve this, we run a given SLS algorithm repeatedly on an instance and measure either the number of search steps or the CPU time required to solve the instance (we assume that the algorithm is given enough resources that it eventually finds a solution in every run). Given this run-time distribution (RTD), it is a simple matter to determine the corresponding CDF. The right pane of Figure 2.1 illustrates such an empirical CDF, obtained by running WalkSAT/SKC 1000 times on a hard Random 3-SAT instance, and recording the number of search steps performed in each run.

The right pane of Figure 2.1 also shows the exponential distribution $ed(x, m) = 1 - 2^{-x/m}$, with $m = 58000$. This exponential distribution fits the RTD very well, which is typical of the RTDs of SLS algorithms (for further discussion of this observation, the reader is referred to [HS98, HS04]). Although it is desirable to study the entire RTD, we often need to quantify the performance of an algorithm on an instance with a single value. Since the RTDs are typically exponentially distributed, summary statistics of the distribution can be meaningfully interpreted. In this thesis, we use the mean of the RTD to quantify the search cost of

an algorithm on a given instance, and refer to this value as the local search cost (*lsc*) of the given algorithm on the given instance.

Because this is an empirical study, we need some benchmark instance sets. Appendix A gives an overview of the instances that we use. When assembling these benchmark sets, we attempted to focus on widely studied sets of instances that were representative of the the instances being studied in the literature. To this end, we include a number of Random 3-SAT test sets from the phase transition region of various sizes [CKT91, GW94]. We also include SAT-encoded instances from other problem domains, such as graph colouring, planning, logistics, and hardware verification and design. Due to the extremely high computational resources required for many of the experiments in this thesis, many of the instances that we study are smaller (and thus correspondingly easier) than those previously found in the literature. However, it is our hope that the properties we observe and the intuitions that we gain by studying these smaller instances carries over to larger instances. This is not without precedence; Yokoo studied the changes in search space across the phase transition for 20 variable Random 3-SAT instances, with very interesting results that did indeed carry over to larger instances [Yok97].

Finally, we briefly mention that a large number of the algorithms that we develop and use in this thesis are based on binary decision diagrams (BDDs). BDDs are essentially a compact method of modelling Boolean functions. In this thesis, we use BDDs to represent sets of search states, in the following way. A given state $\vec{X} = \langle x_1, x_2, \cdots, x_n \rangle$ is treated as the input to a Boolean function f over n variables. The Boolean function returns true if and only if the state $\vec{X}$ belongs to the set represented by f. BDDs are a convenient method of modelling such functions because the BDDs are compact, and many set operations (such as intersection, union, etc.) can be performed very efficiently with the BDDs. Appendix B gives a more detailed overview of BDDs, as well as references to additional resources.

Chapter 3

Plateau Characteristics

One of the most prominent search space features of SAT instances (and many other combinatorial optimisation problems involving discretised evaluation functions) are structures known as plateaus. Simply put, a plateau is a set of states in the search space that are all connected to one another, and are indistinguishable with respect to a given evaluation function. Thus an SLS algorithm has no heuristic guidance when moving between states on a plateau, and must either make a non-improving move or choose uniformly from neighbours on the plateau with the hope of finding a state which has neighbours with a lower evaluation function value (such states are referred to as *exit states*). In this chapter we demonstrate that plateaus are indeed a very prevalent search space feature in a wide variety of SAT instances, and we study many of their properties.

Plateaus are interesting for a variety of reasons. For example, one of the most interesting classes of plateaus are those which consist entirely of states having only neighbours with higher evaluation function values. These states are known as local minima states, and they impede the progress of greedy local search algorithms. If a plateau contains only local minima states, then the plateau is referred to as closed, and it is impossible for an SLS algorithm to improve the evaluation function value without first making at least one non-improving move. Thus, we would like to get a sense of how widespread such closed plateaus are, and what properties the closed plateaus have. Plateaus are also a natural object of study because they fully partition the search space – that is, every state belongs to exactly one plateau. Thus, plateaus allow us to simplify our view of the search space by grouping states together and measuring properties of the encompassing plateaus.

This chapter is structured as follows. The next section presents all of the formal definitions and language needed throughout the chapter, and further motivates the study of plateaus. Next, we study plateaus by sampling the plateaus actually encountered when using an SLS algorithm to solve the instances and look at properties such as the number of plateaus, their size, and the various types of plateaus. Next we describe a method of exhaustively finding all of the plateaus in the search space, and report statistics over all plateaus. Following this, we study the properties of plateaus in the search spaces of structured SAT instances. Next, we study the internal structure of plateaus, and consider how the anatomy of plateaus affects SLS algorithm behaviour. Finally, the chapter concludes with a discussion and summary of the main results.

3.1 Definitions and Motivation

3.1.1 State Types

As previously discussed, the search space $\mathcal{S}$ of a combinatorial optimisation problem is composed of a finite (or countable infinite) set of states. In the case of SAT, given a formula $\mathcal{F}$, the search space $\mathcal{S}(\mathcal{F}) := Assign(\mathcal{F})$, where $Assign(\mathcal{F})$ is simply the set of all complete variable assignments. For convenience, we typically call these assignments *search space states* or, for the sake of brevity, *states*. It should be obvious that there are exactly 2^n states, where $n = \#Var(\mathcal{F})$, since every state involves n variables each having two possible values. Assuming that we are considering only neighbourhood relations $\mathcal{N}(\mathcal{F}) \subseteq Assign(\mathcal{F}) \times Assign(\mathcal{F})$ in which two states are neighbours if and only if they differ in the value of exactly one variable, the search space can be conceptualised as a Boolean hypercube.

In the following, we classify the search space states into different classes, depending on their local neighbourhood. The definition is from [Hoo98, HS04].

Definition 3.1. (State Types)
Let $L = (\mathcal{S}, \mathcal{N}, g)$ be a search landscape. For a state $s \in \mathcal{S}$, we define the following functions which determine the number of upwards, sideways, and downwards steps from s to one of its direct neighbours:

$$\begin{aligned} upw(s) &:= \#\{s' \in \mathcal{N}(s) \mid g(s') > g(s)\} \\ sidew(s) &:= \#\{s' \in \mathcal{N}(s) \mid g(s') = g(s)\} \\ downw(s) &:= \#\{s' \in \mathcal{N}(s) \mid g(s') < g(s)\} \end{aligned}$$

Based on these functions, we define the following state types:

$$\begin{aligned} SLMIN(s) &\Leftrightarrow downw(s) = sidew(s) = 0 \\ LMIN(s) &\Leftrightarrow downw(s) = 0 \wedge sidew(s) > 0 \wedge upw(s) > 0 \\ IPLAT(s) &\Leftrightarrow downw(s) = upw(s) = 0 \\ LEDGE(s) &\Leftrightarrow downw(s) > 0 \wedge sidew(s) > 0 \wedge upw(s) > 0 \\ SLOPE(s) &\Leftrightarrow downw(s) > 0 \wedge sidew(s) = 0 \wedge upw(s) > 0 \\ LMAX(s) &\Leftrightarrow downw(s) > 0 \wedge sidew(s) > 0 \wedge upw(s) = 0 \\ SLMAX(s) &\Leftrightarrow sidew(s) = upw(s) = 0 \end{aligned}$$

The states defined by these predicates are called *strict local minima (SLMIN), local minima (LMIN), plateau interior (IPLAT), ledge (LEDGE), slope (SLOPE), local maxima (LMAX),* and *strict local maxima (SLMAX)* states. □

It should be obvious from the definition that every state falls into exactly one of the above classes, and thus the state types partition the search space. A detailed discussion of the state type distribution in various classes of SAT instances can be found in [Hoo98].

3.1.2 Plateaus

While the distribution of states is an interesting topic of study in and of itself, we are more interested in studying larger search space structures that, as we will demonstrate throughout this thesis, have dramatic effects on the behaviour of SLS algorithms. Rather that studying single states, we study groups of connected states. The following definition formalises the notion of a *region* of the search space.

Definition 3.2. (Region)
Let $L = (\mathcal{S}, \mathcal{N}, g)$ be a search landscape and $G_N = (\mathcal{S}, \mathcal{N})$ be the corresponding search graph. A *region* in G_N is a set $R \subseteq \mathcal{S}$ of search states that induces a connected subgraph of G_N. Formally, R is a region in G_N if and only if $\forall s', s'' \in R\ \exists s_0, s_1, \cdots, s_k \in R . \forall i \in \{0, \cdots, k-1\} : s_0 = s' \wedge s_k = s'' \wedge \mathcal{N}(s_i, s_{i+1})$. □

Regions are interesting because they correspond to portions of the search space in which it is possible for an SLS algorithm to move between any two states belonging to the region without leaving the region. Note that depending on the evaluation function, it may actually be improbable (or even impossible) that an SLS algorithm will actually travel between two states in the region — but such a trajectory is not ruled out by the neighbourhood relation.

However, regions are not interesting in and of themselves. Individual states constitute trivial non-interesting regions. The entire search space typically forms a trivial region as well (though this may not be the case, depending on the neighbourhood relation). However, the next definition formalises a very important and interesting type of region, the *plateau region*, as well as a closely related search space structure, the *plateau*.

Definition 3.3. (Plateau Region, Plateau, Level)
Let $L = (\mathcal{S}, \mathcal{N}, g)$ be a search landscape and $G_N = (\mathcal{S}, \mathcal{N})$ be the corresponding search graph. A region R in G_N is a *plateau region* in L if and only if all states in R have the same evaluation function value, *i.e.*, $\exists l \in \mathbb{R}\ \forall s' \in R : g(s') = l$. In this case, define $Level(R) := l$ to be the *level* of plateau region R (we occasionally refer to the level of a state similarly).

A *plateau* in L is a maximally connected plateau region, *i.e.* a plateau region P is a plateau if and only if $\neg\exists s \in P\ \exists s' \in S \backslash P : \mathcal{N}(s, s') \wedge g(s) = g(s')$. We denote the (unique) plateau containing a state $s \in S$ with $Plateau(s)$. □

The preceding definition was the central definition of the chapter, and we will spend the majority of the remainder of the chapter studying plateaus and their properties. As we have alluded to earlier, every state in the search space belongs to exactly one plateau and so plateaus form a natural partition of the search space. Furthermore, because the evaluation function value is constant for all states in a plateau, the evaluation function cannot be used by SLS algorithms to guide the search when choosing between states in the same plateau. Also, since all states in the same plateau are connected, an SLS algorithm can move between any two states on the plateau without leaving the plateau (though it may, of course, require multiple steps). All of these plateau properties enable us to group the states in a plateau together and treat the entire plateau as a single entity. As we will demonstrate, understanding the properties of plateaus allows us to more fully understand how search space features can affect algorithm behaviour.

We next define two important classes of plateaus — open and closed. Because local search algorithms have no heuristic guidance (w.r.t. the evaluation function) while searching a plateau, it is not obvious how much time should be invested in searching a given (potentially closed) plateau. The answer to this question depends on many factors, but it essentially boils down to determining the expected time that it would take to find a state on that plateau from which the search can progress again; such a state is referred to as an exit. The following definition classifies plateaus into open plateaus, which are plateaus which contain at least one exit (and thus may be worth searching), and closed plateaus which do not contain exits.

Definition 3.4. (Exits, Open and Closed Plateau Regions)
Let $L = (S, \mathcal{N}, g)$ be a search landscape and P a plateau in L. A state $s \in P$ is an *exit* of P if and only if s has a neighbour at a lower level than P, *i.e.* $EXIT(s)$ holds if and only if $\exists s' \in S \backslash P : \mathcal{N}(s, s') \wedge g(s) > g(s')$. In this context, s' is called a *target of exit* s.

A plateau P is called an *open plateau* if and only if it contains at least one exit, otherwise P is called a *closed plateau*. □

The remainder of this chapter is dedicated to an empirical investigation of the properties of plateaus in the search space. We would like to provide answers to questions such as: How many plateaus are there? What is the ratio of open to closed plateaus? How big are the plateaus? What does the internal structure of these plateaus look like?

3.2 Plateau characteristics of the UF-3-SAT distribution

In this section, we study plateaus in the search spaces of instances from the UF-3-SAT distribution (a description of this distribution is given in Appendix A). Rather than study properties of individual instances,

we attempt to probabilistically sample plateaus uniformly and at random from the entire UF-3-SAT distribution, and measure their properties. Thus, this analysis gives insight into what type of plateaus one would expect to encounter in a typical instance from the UF-3-SAT distribution, but does not give any sense of what the search spaces of individual instances look like.

The experimental analyses in this section are intended to replicate those of [FCS97] exactly. We replicated the instance distributions and experimental methodology as accurately as possible from the descriptions in the literature, and performed the same analyses. However, our results differ dramatically from the previously published results, and we draw correspondingly different conclusions. These discrepancies will be discussed throughout this chapter, but we note at this time that the results presented in this section have been rigorously checked, and corroborated by several different methods.

3.2.1 Experimental Methodology

Ideally, we would like to sample plateaus at low levels uniformly and at random from the UF-3-SAT distribution. Unfortunately, this is not possible unless the number of variables is very low. To overcome this problem, we follow the approach of [FCS97] and use GSAT (*c.f.* Section 2.3) to sample plateaus. For each instance in the distribution (1000 instances total), we ran GSAT once for each level from 0 to 5 and output the first state encountered by GSAT at that level. If GSAT did not encounter a state at the desired level during the run, it was run again. This process gave us a distribution of states for each level 0 to 5. Because we used GSAT to sample the states, the distributions are biased — nonetheless, since we are interested in SLS algorithm behaviour, the states form an interesting sample. Also, since GSAT is initialized randomly, the bias is due only to the greedy descent mechanism in GSAT, and not by the starting state.

Each state s was then expanded into $Plateau(s)$ by using breadth-first search. Each state was stored with very low memory overhead in a bit vector, and double-counting of states was avoided by storing them in a hash-table. Because the plateaus were frequently very large, a maximum of 10^6 states were explored (compare this with a maximum of 10 000 states in [FCS97]). We recorded statistics such as the size of the plateau and the number of exits, as well as other statistics which will be discussed in later sections.

3.2.2 Fraction of Closed Plateaus

Figure 3.1 shows the proportion of plateaus that are closed at each level. It is interesting to note that the shape of the curves is similar for all number of variables. The proportion of closed plateaus is (trivially) 1 at level 0, and decreases rapidly as the level is increased — there are vanishingly few closed regions at higher levels. The proportion of closed plateaus decreases faster than exponentially with the level; for example, for the *uf50* test set, the best-fit function was $f(x) = a^{-x^b}$, with $a \approx 2$ and $b \approx 1.35$, and similar functions fit

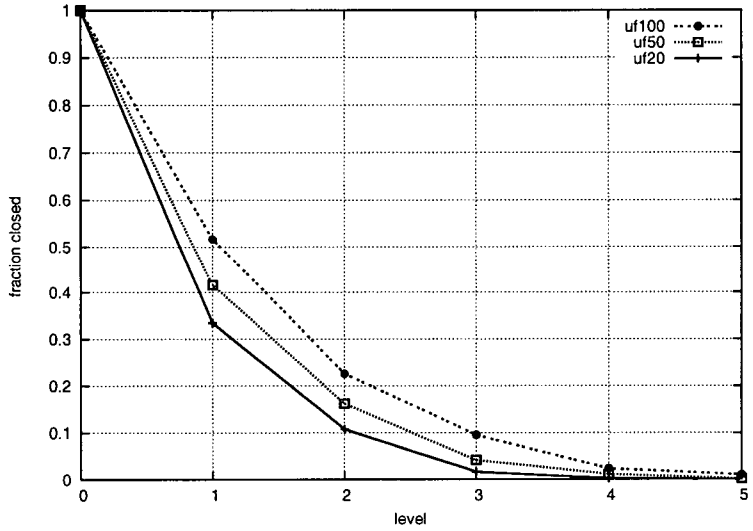

Figure 3.1: Fraction of closed plateaus sampled at each level.

the data well for the other test sets.

Perhaps most interesting is the relatively low proportion of closed plateau regions at level 1 — in fact, over half of the plateaus sampled from the *uf20* and *uf50* distributions were open, and only slightly less than half of those from *uf100* were open as well. This indicates that a solution can be reached from a large fraction of the plateaus sampled at level 1 simply by exploring the plateau until an exit is found. While this may make it sound like these instances should be trivial to solve, many other details must be considered. For example, even though a plateau is open, it is not necessarily the case that it is easy to find an exit — the plateau may be very large and there may be only a few exits, or the exits may be clustered together in a small, fairly unconnected region of the plateau. This is studied in depth in Section 3.5.

As hinted at earlier, there are discrepancies between these results and results of Frank, Cheeseman and Stutz that this experiment was supposed to replicate exactly [FCS97]. Frank *et al.* also show a monotonic decrease in the proportion of closed plateaus with increasing level. However, the values that they report are much higher in some cases. For example, for the *uf100* distribution, Frank *et al.* report that approximately 0.9 of the plateaus samples at level 1 were closed, down to approximately 0.2 at level 5. Their results for smaller test sets are more comparable to ours.

3.2.3 Plateau Size

Figures 3.2, 3.3, and 3.4 show the distribution of the sizes of the plateau regions encountered by GSAT on the *uf20*, *uf50*, and *uf100* test sets, respectively. Our first observation is that plateaus in these distributions are very large, and the size is directly proportional to both the level and the number of variables. The

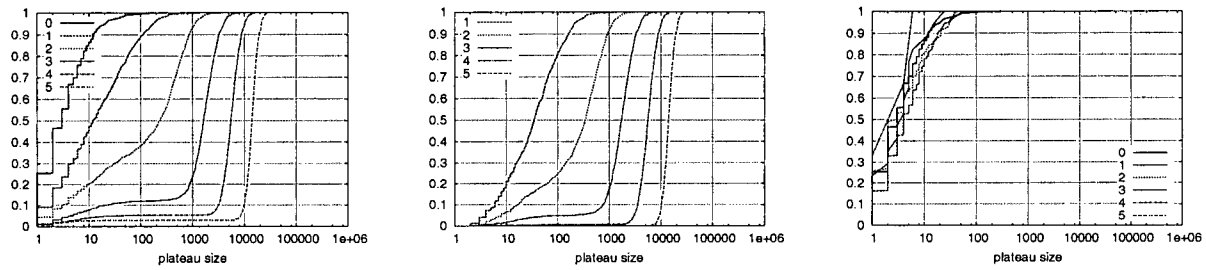

Figure 3.2: Distribution of plateau sizes (UF-3-SAT, 20 variables). Left: all plateaus, Centre: open plateaus, Right: closed plateaus.

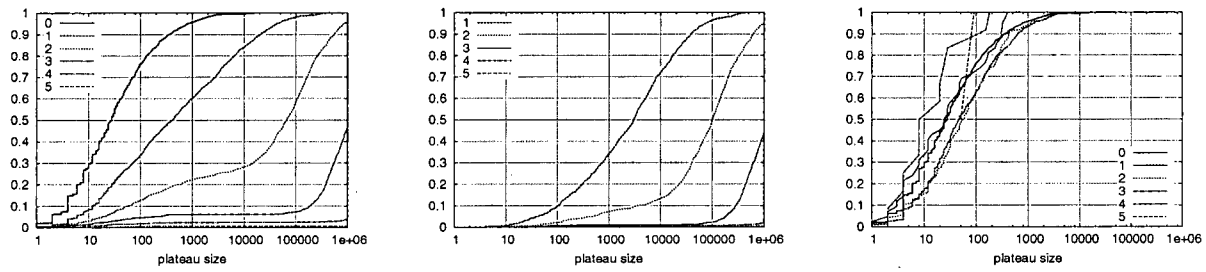

Figure 3.3: Distribution of plateau sizes (UF-3-SAT, 50 variables). Left: all plateaus, Centre: open plateaus, Right: closed plateaus.

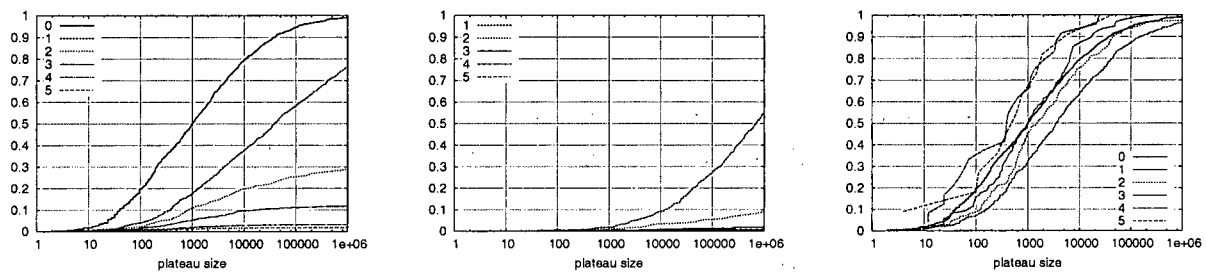

Figure 3.4: Distribution of plateau sizes (UF-3-SAT, 100 variables). Left: all plateaus, Centre: open plateaus, Right: closed plateaus.

variance in the distributions decreases substantially with the level which is quite interesting, but not particularly surprising considering the random nature of the instances. Recall that the plateau exploration was terminated after encountering 10^6 states, explaining why many of the CDFs are truncated.

We first consider the sizes of the closed plateaus, found in the right panes of the figures. The closed plateaus are relatively small, and the sizes increase with the number of variables. There is a slight tendency for higher level closed plateaus to be *smaller* than those at lower levels. Recall that there are also fewer closed plateaus at higher levels, which is why there are so few data points in the CDFs for the high-level distributions. Thus, we observe that closed plateaus are larger and more abundant at lower levels.

The situation looks quite different for open plateaus, which are shown in the centre panes of the figures. We find that the vast majority of the sampled open plateaus are very large, with the size increasing with level. Because a maximum of 10^6 states were explored, the full CDFs are visible only for the *uf20* distribution. In fact, for the *uf100* distribution, only 55% of the level 1 open plateaus, 10% of the level 2 open plateaus, and very few higher level plateaus had less than 10^6 states. The vast majority of the sampled plateaus for the *uf100* test set were very large.

These results are not at all consistent with those of Frank *et al.* [FCS97]. For example, Frank *et al.* report that the median size of level 1 closed plateaus sampled from the *uf100* distribution was 48 states, and that only approximately 4% of the level 1 closed plateaus contained more than 1000 states. In contrast, we find a median size of 3580 states and fully 68% of the level 1 closed plateaus contained more than 1000 states. In fact, 4% of the level 1 closed plateaus that we sampled from the *uf100* had more than 10^6 states. Our results for the open plateaus also differ dramatically from those of Frank *et al.*. For the *uf100* distribution, Frank *et al.* report that, although they found that open plateaus were typically 10-30 times larger than closed plateaus, the vast majority of open plateaus at level 1 consisted of fewer than 1000 states and very few had more than 10 000. In contrast, less than 2% of the level 1 open plateaus that we sampled from the *uf100* distribution were smaller than 1000 states, and fully 45% of them were larger than 10^6 states.

3.2.4 Discussion

It is somewhat troubling that our results from this section differ so dramatically from those of Frank *et al.* [FCS97], which they were intended to exactly replicate. However, we reiterate that all of our results presented in this section were carefully checked, and we are confident that they are correct. All of the algorithms employed were corroborated using completely different methods (the methods from succeeding sections). Because we are reporting much larger plateaus than those from the literature, one might suspect that we are somehow over-representing plateaus — that is, counting states that should not be included in a plateau or double-counting states. However, it is very simple to confirm that a set of states are all elements of a single plateau, and we have performed this check for a large number of the plateaus we sampled.

It is unclear what could cause the difference in results. One possible source of the discrepancy is that, as we will see in the next section, there are in fact many, many small plateaus in the search space. However, when we actually sampled the plateaus that were encountered by GSAT over a large number of runs, none of these small plateaus were encountered. It is possible that Frank *et al.* somehow used an unsound implementation of GSAT that encountered a large number of these small plateaus.

While the analysis in this section paint an interesting picture of what plateaus are typically found in an entire UF-3-SAT distribution, it gives no indication of what the search spaces of individual instances look like. For example, we are most interested in how the search spaces of, say, the easiest and hardest instances from a distribution differ. We would like to know what search space structures are present in the hard instances that cause difficulties for SLS algorithms.

3.3 Plateau characteristics of individual UF-3-SAT instances

In this section, we study the search space of individual UF-3-SAT instances in detail, rather than sampling search space features from the entire instance distribution. For each instance, we find every low level state, and then group the states into their respective plateaus. Thus, we get a detailed look at the complete low-level region of the search space for each instance.

It is critical to study the search space on an instance-by-instance basis in order to understand the behaviour of SLS algorithms. Such an approach makes it possible to study the search space features responsible for making a given instance easy or hard for local search to solve, rather than simply making observations about the entire instance distribution.

3.3.1 Experimental Methodology

In order to study the complete low-level search space of these instances, we need a method to find every single low level plateau. This is quite a difficult task. Sampling cannot be used, because this will not ensure that the entire search space is sampled. The complete search space cannot be enumerated because it is too large. Our approach to the problem in to first find all of the states at a given level, and then to partition this set of states into its component plateaus. The following definition formalises the notion of the complete set of states at a given level, which we refer to as a levelset.

Definition 3.5. (Levelset)
Let $L = (\mathcal{S}, \mathcal{N}, g)$ be a search landscape. The *levelset at level k*, denoted $Levelset^k$ is the set of all states having level k. Formally, $Levelset^k := \{s \in \mathcal{S} \mid g(s) = k\}$. Note that $Levelset^k$ is comprised of all of the states from all plateaus at level k. □

Algorithm 3.1: Levelset($\mathcal{F}$, *maxLev*)

Input : CNF Formula $\mathcal{F}$, Integer *maxLev*
Output: Set of BDDs representing *Levelset*

```
1  begin
2      // Build the partial levelsets w.r.t. each clause
3      LS ← ∅
5      foreach c ∈ F do
7          pls^0 ← BDDSatSet(c)
9          pls^1 ← ¬BDDSatSet(c)
11         for k ← 2 ··· maxLev do
13             pls^k ← ⊥
14         end
16         LS ← LS ∪ {pls}
17     end
18     // Merge the partial levelsets until only one remaining
20     while #LS > 1 do
22         (pls_1, pls_2) ← ChooseTwoPartialLevelsets(LS)
24         LS ← LS \ {pls_1, pls_2}
26         for k ← 0 ··· maxLev do
28             pls^k_1o2 ← ⋁_{i,j|i+j=k} (pls^i_1 ∧ pls^j_2)
29         end
31         LS ← LS ∪ {pls_1o2}
32     end
33     // Only one set of partial levelsets left — it represents the set of complete levelsets
35     return ls, where ls is the single remaining element of LS
36 end
```

Because the sets of states that we are exploring are so large, it is infeasible to store them using conventional means such as simple lists of bit vectors. To avoid these problems, we use Binary Decision Diagrams (BDDs) to store the levelsets and the plateaus. BDDs allow us to represent the sets of states by Boolean functions: a state is an element of the set if and only if it is a minterm of the Boolean function represented by the BDD. Appendix B gives a brief introduction to BDDs, and a description of the BDD functions that we use.

The pseudo-code shown in Algorithm 3.1 elucidates our method for calculating the low level levelsets $Levelset^k$, for $k = 0, \cdots, maxLev$. For all of the experiments in this section, we set $maxLev$ to 5. The input to the algorithm is a CNF Formula, and the maximum level that we would like to calculate the levelset for. Note that the procedure calculates all of the levelsets simultaneously.

A key concept for understanding this process is that of a partial levelset. This is simply a levelset with respect to a subset of the clauses. In the for loop declared at line 5, we build partial levelsets with respect to each clause. At level 0, we simply have all states that satisfy the clause. At level 1, we have all states that do not satisfy the clause. Every state in the search space falls into exactly one of these classes, so for all levels

greater than 1 we set the BDDs to $\perp$, representing the empty set. Note that each *pls* is a set of $maxLev + 1$ BDDs, and that *LS* is a set of sets of BDDs.

Once we have these partial levelsets for each clause, we begin to combine them. At line 22, we call a procedure to heuristically choose two partial levelsets to be combined. Any two choices are correct, but the size of the BDDs explodes if the order of combination is not made carefully. We have found that choosing two partial levelsets according to a heuristic that both maximises the overlap of the support of the two BDDs while simultaneously minimising the BDD sizes gives the best results. The two chosen partial levelsets are removed from the collection *LS*, and then recombined into a new partial levelset.

The levelset combination is shown in line 26. It should be clear that, given two disjoint sets of clauses C_1 and C_2 and their corresponding partial levelsets pls_1 and pls_2, it is fairly straightforward to build the partial levelset $pls_{1\circ 2}$ for $C = C_1 \cup C_2$. In order for a state s to be at level k with respect to C, it will have to be at level i with respect to C_1 and level j with respect to C_2, where i and j are chosen such that $i + j = k$. This is exactly what is accomplished in line 26 (note that the $\wedge$ and $\vee$ operations are being performed on BDDs, and correspond to the BDD *and* and *or* operations, respectively). Finally, $pls_{1\circ 2}$ is added to the collection *LS*. Note that $pls_{1\circ 2}$ is the partial levelset with respect to all of the clauses covered by pls_1 and pls_2. Since we start with a set of partial levelsets, each corresponding to a single clause in the formula, when all of the partial levelsets have been recombined so that there is only one remaining, this last remaining partial levelset is actually the (complete) levelset with respect to the entire formula.

Once we have collected all of the low level levelsets, we must partition each levelset into its component plateaus. The pseudo-code of Algorithm 3.2 illustrates our partitioning process. The input to the algorithm is a BDD representation of $Levelset^k$, for some k, and the output is the set of all plateaus at level k (also represented as BDDs).

In line 4, we choose a single cube from the remaining partition, and add it to the BDD *P*, representing the explored portion of the plateau. Starting from those states, we iteratively expand the plateau using a set-wise breadth-first-search until all of the states in the plateau have been added. Lines 9 – 13 illustrate how the plateau is expanded. For every variable v, the function representing the plateau but with v complemented is calculated, and restricted to level k. The disjunction of all of those functions, *Nbr*, represents all neighbours of the partial plateau that are also elements of the plateau being explored (note that many of the states may already be in *P*). *Nbr* is then added to *P* and removed from *Remaining*. It is trivial to see that when no more new states are generated, the entire plateau has been explored.

Using the two procedures described above, we can find and represent every plateau in the entire search space, or just all plateaus at low levels in the search space. It is no longer necessary to resort to sampling the search space — we get a complete picture of the entire low level portion of the search space. However, the computation is exceptionally expensive (some of the experiments used to collect data for the succeeding

Algorithm 3.2: Partition($Levelset^k$)

Input : BDD Representation of $Levelset^k$
Output: Set of BDDs representing each plateau

```
1  begin
2      Remainder ← Levelset^k                          // Start with entire levelset
3      while Remainder ≠ ⊥ do
4          P ← BDDPickOneCube(Remainder)               // choose one BDD cube to start from
5          fixedPoint ← false
6          // expand the plateau containing that cube (setwise-BFS)
7          while Remainder ≠ ⊥ and fixedPoint = false do
8              // find all neighbours on the current plateau
9              foreach v ∈ Var(F) do
10                 Nbr_v ← BDDNegate(P, v)
11                 NbrLS_v ← Nbr_v ∧ Remainder
12             end
13             Nbr ← ⋁_{v∈Var(F)} NbrLS_v
14             // check for fixed point
15             if P ∧ Nbr = P then
16                 fixedPoint ← true
17             else
18             end
19         end
20         fixedPoint ← false
21         P ← P ∨ Nbr
22         Remainder ← Remainder ∧ ¬Nbr
23         // add the complete plateau to the set of all plateaus
24         PlateauSet ← PlateauSet ∪ {P}
25     end
26     return PlateauSet
27 end
```

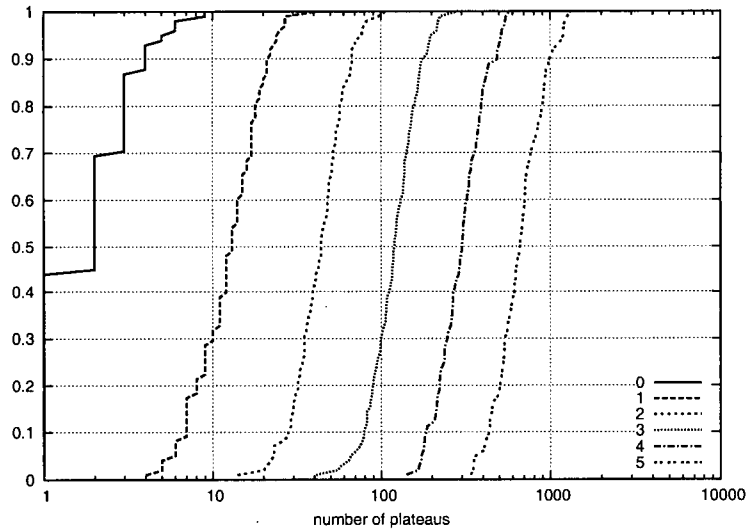

Figure 3.5: Distribution of the number of plateaus at each level over instances in the *uf30* test set.

	level											
	0		1		2		3		4		5	
Test Set	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
uf20	1.7	(0.65)	8.4	(0.43)	22.8	(0.41)	47.8	(0.36)	90.2	(0.30)	155.6	(0.27)
uf25	1.7	(0.56)	10.5	(0.41)	32.0	(0.36)	76.2	(0.33)	168.1	(0.32)	339.3	(0.30)
uf30	2.2	(0.73)	13.7	(0.46)	46.2	(0.36)	127.1	(0.33)	315.9	(0.32)	703.1	(0.33)
uf35	2.3	(0.80)	16.9	(0.51)	63.8	(0.42)	193.1	(0.38)	510.8	(0.35)	1239.1	(0.34)
uf40	2.4	(0.78)	18.6	(0.53)	79.1	(0.45)	270.6	(0.45)	794.2	(0.44)	2088.2	(0.43)
uf45	2.6	(0.73)	22.1	(0.48)	98.6	(0.39)	357.0	(0.37)	1134.2	(0.34)	3210.9	(0.33)

Table 3.1: Statistics of the number of plateaus in UF-3-SAT instances, counting all plateaus. For every test set and level we show the mean number of plateaus as well as the coefficient of variation.

sections ran for over four CPU weeks), so the analysis is restricted to fairly small instances. Nevertheless, we hope that the intuitions and hypotheses developed from studying these smaller instances will also hold for larger instances — and we will show that this is the case in later chapters of the thesis.

3.3.2 Number of Plateaus

Figure 3.5 shows the distribution of the number of plateaus (of all types) for the *uf30* test set. The number of solution plateaus is quite low — in any given instance, there are less than 10 solution plateaus — and the CDF for level 0 looks quite jaggy simply because of the discretised scale on the x axis of the plot. As the level is increased, the typical number of plateaus also increases, though the relative rate of increase decreases with level. Interestingly, the coefficient of variation also decreases with level.

Table 3.1 shows summary statistics of the number of plateau regions found at each level for all of the UF3SAT test sets. We see that the observations we made for the *uf30* test set generalize with the number of

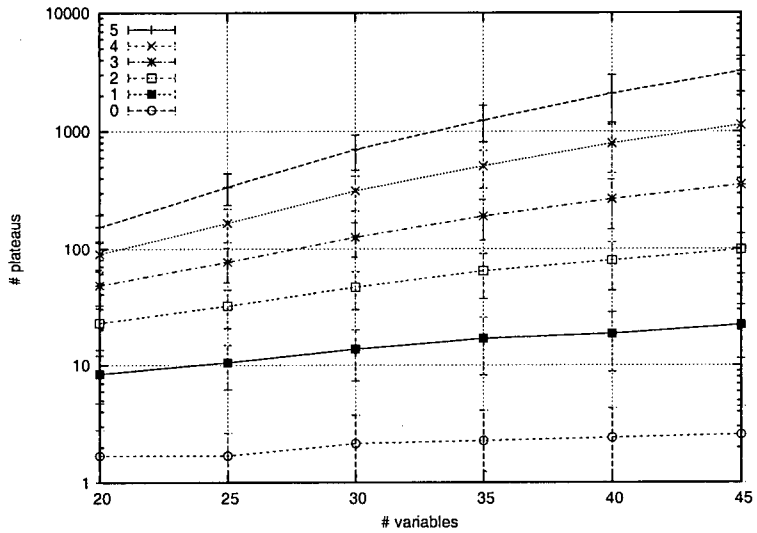

Figure 3.6: Scaling of the average number of plateaus with the number of variables for the UF3SAT test sets. Each point represents the average number of plateaus at the given level, for the distribution with the given number of variables. The error bars show the standard deviation of each distribution.

variables. In all cases, the average number of plateaus increases with level, while the coefficient of variation decreases. We also observe that the number of plateaus increases rapidly with the number of variables. Figure 3.6 shows how the number of plateaus scales with the number of variables. There are too few data points for an accurate assessment of the scaling behaviour, but the scaling seems to be sub-exponential (note the log scale on the y axis of the plot).

3.3.3 Fraction of Closed Plateaus

Tables 3.2 and 3.3 show the same data as Table 3.1, except that the plateaus have been separated into open and closed plateaus, respectively. We observe that the vast majority of plateaus are in fact open — especially at higher levels. Interestingly, while the number of open plateaus increases with level, the number of closed plateaus decreases. Thus, closed plateaus exist mainly at lower levels and are not only relatively rare (compared to open plateaus) at higher levels, they simply do not occur. Note that the number of closed plateaus does increase with the number of variables.

Figure 3.7 shows the average proportion of plateaus that are closed at each level for all of the UF3SAT test suites. This figure is surprisingly similar to Figure 3.1. The ratio decreases rapidly with the level; the function $f(x) = a^{-x^b}$ fits these curves well. The ratios tend to be higher for larger number of variables, though this is not true, for example, for the *uf25* test set, which has a higher than expected ratio for lower levels, and lower than expected for higher levels.

The coefficient of variation in the distribution of plateau counts is significantly higher for the closed

	level									
	1		2		3		4		5	
Test Set	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
uf20	6.0	(0.56)	20.7	(0.45)	46.7	(0.37)	89.8	(0.31)	155.6	(0.27)
uf25	7.0	(0.53)	28.5	(0.41)	73.8	(0.35)	167.4	(0.32)	339.1	(0.30)
uf30	9.4	(0.57)	40.9	(0.42)	123.7	(0.35)	314.2	(0.33)	702.6	(0.33)
uf35	11.3	(0.73)	56.5	(0.48)	187.3	(0.40)	507.5	(0.36)	1237.7	(0.34)
uf40	12.2	(0.71)	69.6	(0.52)	262.1	(0.47)	788.5	(0.44)	2085.1	(0.43)
uf45	14.3	(0.65)	86.3	(0.44)	343.8	(0.39)	1123.6	(0.35)	3205.2	(0.33)

Table 3.2: Statistics of the number of plateaus in UF-3-SAT instances, counting only open plateaus. For every test set and level we show the mean number of plateaus as well as the coefficient of variation.

	level											
	0		1		2		3		4		5	
Test Set	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
uf20	1.7	(0.65)	2.4	(0.69)	2.1	(1.00)	1.1	(1.03)	0.4	(1.92)	0.1	(3.55)
uf25	1.7	(0.56)	3.5	(0.59)	3.6	(0.67)	2.4	(0.83)	0.7	(1.47)	0.1	(3.03)
uf30	2.2	(0.73)	4.3	(0.61)	5.3	(0.59)	3.4	(0.77)	1.7	(1.05)	0.5	(1.73)
uf35	2.3	(0.80)	5.6	(0.67)	7.3	(0.58)	5.8	(0.68)	3.3	(0.83)	1.4	(1.07)
uf40	2.4	(0.78)	6.5	(0.62)	9.5	(0.54)	8.6	(0.62)	5.7	(0.78)	3.1	(0.91)
uf45	2.6	(0.73)	7.8	(0.61)	12.3	(0.48)	13.2	(0.49)	10.7	(0.72)	5.7	(0.93)

Table 3.3: Statistics of the number of plateaus in UF-3-SAT instances, counting only closed plateaus. For every test set and level we show the mean number of plateaus as well as the coefficient of variation.

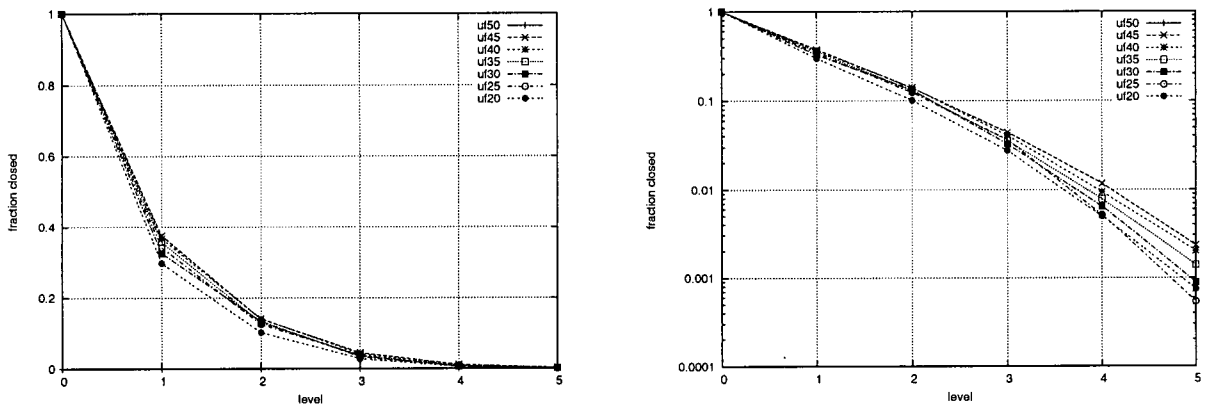

Figure 3.7: Average fraction of closed plateaus at each level for the UF3SAT test suites. Both plots show the same data, but the plot on the right is a semi-log plot which illustrates the (super-)exponential decrease of fraction closed with the level.

	level											
	0		1		2		3		4		5	
Test Set	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
uf20	6.7	(2.66)	5.3	(0.82)	4.9	(1.04)	2.5	(0.69)	2.1	(0.58)	2.4	(0.94)
uf25	8.0	(1.25)	7.1	(0.85)	5.3	(0.82)	3.6	(0.68)	2.3	(0.80)	2.0	(0.44)
uf30	13.8	(1.35)	17.4	(1.60)	9.2	(0.86)	5.2	(0.99)	4.3	(1.14)	3.2	(1.15)
uf35	35.0	(2.00)	24.7	(1.03)	13.3	(0.79)	8.4	(0.80)	5.6	(0.62)	4.4	(0.85)
uf40	76.3	(3.38)	37.4	(1.47)	23.4	(1.26)	11.6	(0.76)	6.8	(0.74)	4.8	(0.65)
uf45	68.4	(1.47)	48.9	(0.83)	25.1	(0.69)	19.7	(0.89)	11.5	(0.69)	7.3	(0.63)

Table 3.4: Statistics of the size of closed plateaus in UF-3-SAT instances. For every test set and level we show the mean of the average size of closed plateaus in each instance, as well as the coefficient of variation of the distribution of average sizes.

	level									
	1		2		3		4		5	
Test Set	f_{max}	$\sigma_x/\bar{x}$	f_{max}	$\sigma_x/\bar{x}$	f_{max}	$\sigma_x/\bar{x}$	f_{max}	$\sigma_x/\bar{x}$	f_{max}	$\sigma_x/\bar{x}$
uf20	0.64	(0.3)	0.74	(0.3)	0.91	(0.1)	0.96	(0.0)	0.98	(0.0)
uf25	0.64	(0.4)	0.78	(0.3)	0.95	(0.1)	0.98	(0.0)	0.99	(0.0)
uf30	0.70	(0.3)	0.87	(0.2)	0.96	(0.1)	0.99	(0.0)	>0.99	(0.0)
uf35	0.71	(0.3)	0.88	(0.2)	0.98	(0.0)	>0.99	(0.0)	>0.99	(0.0)
uf40	0.74	(0.3)	0.89	(0.2)	0.99	(0.0)	>0.99	(0.0)	>0.99	(0.0)
uf45	0.75	(0.3)	0.91	(0.1)	0.98	(0.0)	>0.99	(0.0)	>0.99	(0.0)

Table 3.5: Fraction of states contained in the largest plateau for the UF3SAT test sets. For every test set and level k we show the mean and coefficient of variation of the distribution of f^k_{max} values.

regions than the open, and this value increases with the level. This is very interesting, suggesting that while some search space features are fairly uniform between instances, the number of closed plateaus is not. Obviously, the number of closed plateaus is closely related to the number of LMIN states, which is known to be correlated with problem hardness in other combinatorial optimisation problems [Yok97]. Thus, it is features such as this that may help to explain the large variance in local search cost within a distribution of homogeneous instances.

3.3.4 Plateau Size

Table 3.4 shows statistics of the sizes of closed plateaus in the UF-3-SAT test sets. Not surprisingly, the size of the closed plateaus grows with the number of variables. What is more interesting, however, is that the closed plateaus are typically very small, and that the size actually decreases with the level. Thus, closed plateaus are both very rare and very small at higher levels. Since we believe that closed plateaus are one of the most prominent search space features that interfere with SLS algorithms, this is quite interesting — it is crucial for a high-performance SLS algorithm to effectively avoid closed plateaus at low levels of the search space.

The situation with open plateaus is markedly different. We find that the distribution of sizes of open plateaus is quite irregular. Recall that there are a large number of open plateaus in the search space, espe-

	level									
	1		2		3		4		5	
Test Set	$\overline{med}$	$\overline{max}$	$\overline{med}$	$\overline{max}$	$\overline{med}$	$\overline{max}$	$\overline{med}$	$\overline{max}$	$\overline{med}$	$\overline{max}$
uf20	2.5	63.8	1.9	428.6	1.7	1913.1	1.4	5891.4	1.1	14266.2
uf25	3.7	141.0	2.7	1389.0	2.1	7779.1	1.8	29658.1	1.6	88846.3
uf30	4.6	524.0	3.3	5753.5	2.5	34524.6	2.1	150860.1	2.0	523251.1
uf35	7.4	1409.8	4.4	15082.2	3.4	101773.3	2.6	512087.0	2.1	2.07×10^6
uf40	10.8	3675.0	6.2	46339.7	4.1	351692.3	3.3	1.98×10^6	2.6	8.99×10^6
uf45	16.9	5547.1	8.3	91009.9	5.8	851473.2	4.4	5.73×10^6	3.6	3.04×10^7

Table 3.6: Statistics of the size of open plateaus in UF-3-SAT instances. For every test set and level we show the mean of the distribution of median and maximum sizes of open plateau regions.

cially at higher levels. With this in mind, consider the data presented in Table 3.5. In this table, we show statistics of the value:

$$f^k_{max} = \frac{\#P^k_{max}}{\#Levelset^k}$$

where k is the level, and P^k_{max} is the open plateau at level k having maximal size. Intuitively, f^k_{max} gives an indication of how much of the level is contained in the largest plateau — it will soon become clear why this is an interesting value to study.

Table 3.5 illustrates a very interesting phonomenon: even though there are many open plateaus (upwards of 3000 for the largest test sets at level 5), *most of the states at higher levels belong to a single, large, open plateau*. This phenomenon is present for all of the test sets, and appears stronger for the larger sets. Starting at roughly level 3, every level has one large plateau region that contains the bulk of the states at that level. At higher levels, there are (relatively) very few states that do not belong to P_{max}. There is also virtually no variance in the distribution of f^k_{max} values for large k, indicating that the phenomenon is present in all of the instances.

Due to the irregular distribution of open plateau sizes, it is meaningless to consider, for example, the average plateau size at a given level. In order to illustrate the typical size of open plateau regions, we look at the distribution of median and max sizes, and present statistics of those distributions in Table 3.6. Not surprisingly (considering the results from the preceding paragraph), we find that the median open plateau size tends to be quite small. Surprisingly, the median size decreases with the level — typical open plateaus at level 5 contain less than 5 states. Closer examination reveals that at higher levels these very small open plateaus are typically degenerate in that they consist entirely of exit states — these plateaus would have very little effect on the behaviour of SLS algorithms. The median size of open plateaus increases with the number of variables.

The size of P^k_{max} is also shown in Table 3.6. In contrast to the median open plateau size, the maximum open plateau size increases rapidly with level. The maximum size also increase rapidly with the number of variables so that at level 5, the largest open plateau in the 45 variable test set has over 30 million states.

	level																	
	0			1			2			3			4			5		
Instance	#	○	●	#	○	●	#	○	●	#	○	●	#	○	●	#	○	●
anom	1	0	1	4	2	2	13	12	1	56	56	0	183	171	12	574	565	9
medium	2	0	2	5	2	3	37	33	4	221	188	33	986	871	115	4652	4431	221
flat20-easy	48	0	48	73	73	0	86	86	0	494	494	0	3023	3023	0	9014	9014	0
flat20-med	168	0	168	133	133	0	355	355	0	2371	2371	0	5662	5662	0	15841	15841	0
flat20-hard	6	0	6	43	19	24	38	38	0	320	320	0	1238	1238	0	3458	3458	0
par8-1-c	1	0	1	1	1	0	19	15	4	65	18	47	414	229	185	1794	1152	642

Table 3.7: Number of plateaus in structured instances. For each instance and level, we show the total number of plateaus, the total number of open plateaus, and the total number of closed plateaus (#, ○ and ●, respectively).

Happily, the proportion of exits from these immense plateaus is very high (for very high levels, these large plateaus actually consist only of exits), so they do not significantly impede the progress of SLS algorithms.

3.4 Plateau characteristics of structured instances

In this section, we turn our attention to another class of instances — structured instances. These instances (described in Appendix A) are not randomly generated, but rather are SAT-encoded instances of other problems, such as planning, graph colouring, and parity checking. Again, we examine the characteristics of plateaus in the search spaces of these instances. The algorithms we use and the data we measure are identical to those of the preceding section, allowing the results to be contrasted. We first study the number of plateaus distributed in the low levels of the search space, and measure the fraction of the plateaus that are closed. Following this, we present results of the empirically measured distributions of the sizes of the plateaus.

3.4.1 Number of Plateaus

We first turn our attention to Table 3.7, which shows the total number of plateaus at each level in the structured instances that we studied. The total number of plateaus is further broken down into the number of open and closed plateaus. Note that in this table the entries represent the actual plateau counts for the instances, unlike similar tables in previous sections which displayed statistics across a distribution of instances.

The results are very interesting. We first note that the total number of plateaus at each level increases with the level — this is consistent with our results for the Uniform Random 3-SAT test suites. There are too few data points to make a rigorous claim, but the scaling of the total number of plateaus appears to be exponential with the level, especially at higher levels. The total number of open and closed plateau regions also seem to increase with the level for all of the structured instances, with the exception of the graph

Instance	level 0	1	2	3	4	5
anom	1.00	0.50	0.08	0.00	0.07	0.02
medium	1.00	0.60	0.11	0.15	0.12	0.05
flat20-easy	1.00	0.00	0.00	0.00	0.00	0.00
flat20-med	1.00	0.00	0.00	0.00	0.00	0.00
flat20-hard	1.00	0.56	0.00	0.00	0.00	0.00
par8-1-c	1.00	0.00	0.21	0.72	0.45	0.36

Table 3.8: Fraction of plateaus that are closed at each level in structured instances.

colouring instances which have no closed plateau regions at higher levels. Additionally, there does not seem to be a direct relationship between the size of the instances and the number of plateaus. The blocks-world planning instance, `medium`, is the largest of the instances and while it does have more plateaus than the smaller planning instance `anom` and the smaller parity-checking instance `par-8-1-c`, the flat graph colouring instances have the most plateaus of all the structured instances.

Due to the differences in problem size, it is difficult to draw a direct comparison between these results and those for the Uniform Random 3-SAT test suites. However, the smallest structured instance, `anom`, is approximately the same size as the largest Uniform Random 3-SAT instances — though the instances are obviously syntactically quite different. When comparing the results of these instances, we observe that the `anom` instance has significantly fewer plateaus (and fewer open plateaus) than the average number of plateaus (and average number of open plateaus) in the *uf45* test set. At lower levels the `anom` instance has fewer closed plateau regions than instances in the *uf45*, but this relationship is reversed at higher levels.

Table 3.8 shows the fraction of plateaus at each level that are closed in the structured instances. For the planning instances we see that the fraction of plateaus that are closed tends to decrease rapidly with the level, indicating that the major obstacles to local search algorithms are encountered at low levels. The flat graph colouring instances are quite unusual in that, although they have a large number of plateau regions, all of the plateaus are open. The only exception to this is the hardest colouring instance, in which 56% of the level 1 plateaus were closed. Given this information about the search space of the graph colouring instances, it should not come as a surprise that these instances are indeed the easiest of the structured instances studied here. It is interesting that the hardest instance was the only one to have closed plateaus that were not solutions — this clearly suggests that these closed regions may be the reason why this instance, which is syntactically similar to the other colouring instances, is so much more difficult for SLS algorithms. This will be studied much more closely in the following chapter of this thesis. Lastly, we observe that the parity instance is quite different than all of the other instances studied in this chapter (including the Uniform Random 3-SAT instances) in that a large proportion of its plateaus are closed, even at higher levels in the search space. Thus, local search algorithms may have a difficult time even reaching the lower levels of the search space. This offers a partial explanation why parity instances (more specifically, larger versions of

	level											
	0		1		2		3		4		5	
Instance	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
anom	1.0	(0.00)	1.0	(0.00)	1.0	(0.00)	–	(–)	2.9	(0.89)	4.0	(1.04)
medium	1.0	(0.00)	1.0	(0.00)	1.0	(0.00)	2.3	(1.08)	4.1	(1.20)	5.2	(1.62)
flat20-easy	65530.5	(2.64)	–	(–)	–	(–)	–	(–)	–	(–)	–	(–)
flat20-med	58.4	(1.21)	–	(–)	–	(–)	–	(–)	–	(–)	–	(–)
flat20-hard	945.0	(0.00)	1701.0	(1.04)	–	(–)	–	(–)	–	(–)	–	(–)
par8-1-c	1.0	(0.00)	–	(–)	2.8	(0.47)	19.7	(1.26)	46.8	(2.17)	35.0	(1.67)

Table 3.9: Statistics of the size of closed plateaus in structured instances. For every instance and level we show the mean size of closed plateaus, as well as the coefficient of variation of the distribution. Many of the instances (most notably the graph colouring instances) do not have any closed plateaus at some levels; these entries are labelled with –.

syntactically similar instances) are among the hardest known instances for modern SLS algorithms.

3.4.2 Plateau Size

We now study the distribution of plateau sizes in structured instances. First, Table 3.9 presents statistics of the distributions of sizes of closed plateaus at each level for the structured instances. We note that the mean size of closed plateaus is quite small and has little variance for the planning instances, though the average size does increase slowly with the level. As was mentioned in the previous section, the easy and median difficulty graph colouring problems have no closed plateaus that are not solutions. The solution plateaus tend to be quite large (especially for the easiest instance), again helping to explain why those instances are so easy for local search algorithms. We also see that the average size of the closed regions at level 1 in the hardest graph colouring instance is by far the largest that we have observed in any of the instances (though still only an average of 1701 states). The average size of the closed plateaus in the parity instance increases with the level and is larger than those of the planning instances. We observe a fair amount of variation in this distribution, and closer inspection reveals that there is indeed a wide range of sizes. To illustrate, at level 5 there are 642 closed plateau regions. The smallest has a single state, and the largest has 630 states.

These results are quite different than those for the Uniform Random 3-SAT instances. While the closed plateaus tended to get smaller with the level in the Uniform Random 3-SAT instances, the size of closed plateaus increases with level in the structured instances. Thus, while closed plateaus were only a problem at very low levels for Uniform Random 3-SAT instances, they may significantly affect the behaviour of SLS algorithms even at high levels in the search spaces of structured instances. This is consistent with the relatively poor performance of current SLS algorithms on many structured instances.

Next, we consider the size of open plateau regions in the structured instances. An immediate first question that we would like to resolve is whether or not the structured instances exhibit the "one-big-plateau" phenomenon we observed in the Uniform Random 3-SAT test suites. Table 3.10 shows the f^k_{max}

Instance	level 1	2	3	4	5
anom	0.43	0.19	0.11	0.05	0.04
medium	0.36	0.12	0.06	0.02	0.01
flat20-easy	1.00	0.99	0.98	0.98	0.97
flat20-med	0.99	1.00	1.00	1.00	0.99
flat20-hard	0.96	1.00	0.99	0.99	0.98
par8-1-c	1.00	0.66	0.29	0.13	0.16

Table 3.10: Fraction of states contained in the largest plateau for the structured instances. For every instance and level k we show the (single) f^k_{max} value.

values, which can be contrasted with those in Table 3.5. We observe somewhat mixed results. The planning instances do not have large f^k_{max} values and, in fact, the value decreases with level. The parity instance shows a similar trend, though the values are significantly larger. These results are diametrically opposed to those that we obtained for the Uniform Random 3-SAT test suites — the search spaces of these instances do not seem to be dominated at higher levels by a single large plateau. The f^k_{max} values for the colouring instances, however, are exceptionally large, even at fairly low levels in the search space. Given that we know that there are a large number of plateaus at these levels, it must be the case that the vast majority of (open) plateaus in the search space of these instances are very small (relative to the largest plateau).

Table 3.11 shows statistics of the sizes of the open plateau regions for the structured instances. As expected, both the median and the maximum sizes of the open plateaus tend to increase with the level. The planning and parity instances seem to have fairly small open plateaus compared with the other instances (including the Uniform Random 3-SAT instances), even at fairly high levels of the search space. The graph colouring instances are again the exception. While the median open plateau sizes are all fairly small (less than 1260 states for all levels less than 5), the maximum sizes are extremely large. At the highest level measured, level 5, the largest open plateaus in all of the graph colouring instances consisted of *hundreds of billions* of search states.

In fact, these results are consistent with our observations for the random instances. In all of the cases that we studied (both random and structured), the median plateau size at each level were generally quite small, while the maximum sizes could be very large. This suggests that it may be possible to partially explore plateaus encountered during search, as long as the maximum number of states explored is limited to a fairly small value — for example, the median plateau size (an idea which has been proposed before, *e.g.* in [FCS97, HK95]). An approach such as this opens up many possibilities for augmenting existing SLS algorithms with plateau searching capabilities.

	level									
	1		2		3		4		5	
Instance	*med*	*max*	*med*	*max*	*med*	*max*	*med*	*max*	*med*	*max*
anom	3.0	3.0	3.0	7.0	2.0	19.0	3.0	44.0	3.0	164.0
medium	4.0	4.0	3.0	15.0	3.0	54.0	4.0	177.0	4.0	604.0
flat20-easy	21.0	9.04×10^7	35.0	1.27×10^9	189.0	1.23×10^{10}	273.0	9.45×10^{10}	266.0	5.87×10^{11}
flat20-med	21.0	9.78×10^5	10.0	3.93×10^7	45.0	8.7×10^8	49.0	1.25×10^{10}	42.0	1.26×10^{11}
flat20-hard	945.0	1.43×10^6	135.0	9.78×10^7	945.0	2.84×10^9	1260.0	4.7×10^{10}	1215.0	5.03×10^{11}
par8-1-c	4.0	4.0	6.0	162.0	24.0	498.0	18.0	3585.0	24.0	42886.0

Table 3.11: Statistics of the size of open plateaus in structured instances. For every instance and level we show both the median and max size of the open plateaus.

3.5 Internal Plateau Structure

Up to this point, the distribution of plateaus has been studied — the size, number, fraction of plateaus that are closed, etc. In this section, we study the actual structure of the plateau regions, which can obviously have a large impact on the behaviour of SLS algorithms. We will study properties such as the diameter of plateaus, the branching factor of the states that compose the plateaus, and well as a measure of how difficult it is for SLS algorithms to find an exit from the plateau. The following definitions formalise these concepts.

Definition 3.6. (Diameter)
Let $L = (\mathcal{S}, \mathcal{N}, g)$ be a search landscape and $G_N = (\mathcal{S}, \mathcal{N})$ be the corresponding search graph. Given a plateau P, the *diameter* of P is simply the diameter of the corresponding subgraph of G_N, *i.e.* the maximal graph geodesic of $G' = (P, N_P)$ where $N_P(s_1, s_2) \Leftrightarrow N(s_1, s_2) \wedge s_1, s_2 \in P$. □

Definition 3.7. (Branching Factor)
Let $L = (\mathcal{S}, \mathcal{N}, g)$ be a search landscape, and $G_N = (\mathcal{S}, \mathcal{N})$ be the corresponding search graph. Given a plateau P, the *plateau branching factor* of a state s in P is defined as the fraction of neighbours of s also in P:

$$BranchingFactor(s) := \frac{\#\{s' \in Plateau(s) \mid \mathcal{N}(s, s')\}}{\#\{s'' \in \mathcal{S} \mid \mathcal{N}(s, s'')\}}$$

Similarly, we define the *branching factor of plateau* P as the average of $BranchingFactor(s)$ over all states $s \in P$. □

The branching factor and diameter of a plateau give a sense of how the plateau is actually structured. Plateaus with a high branching factor and low diameter would be "squeezed" into a small region of the search space. Such a plateau would be highly connected (it would be fairly easy to maneuver around

the plateau), and we might expect that most of the propositional variables would have a constant value across the entire plateau. Conversely, plateaus with low branching factor and high diameter would be more "brittle" and would snake their way throughout a larger portion of the search space.

Definition 3.8. (Exit Distance)
Given an open plateau P, the *plateau exit distance* of a state $s \in P$ is simply the length of the shortest path in P from s to an exit state. We define the *exit distance of plateau* P as the average plateau exit distance of all states $s \in P$. □

Exit distance is a very interesting measure — it helps to quantify the intuition that even open plateaus may be difficult to escape from. If there are few exits distributed in a large plateau, they could be difficult for an SLS algorithm to find — especially considering that there is no heuristic guidance from the objective function on a plateau. Orthogonally, if there are many exits but they are all clustered together and not well connected with the rest of the plateau, it may still be difficult to reach an exit.

With the goal of aiding in the conceptualisation of internal plateau structure, Figures 3.8, 3.9 and 3.10 show sample plateaus from three very different instances. Each node in the graphs represents a single state. Neighbouring states are connected by edges. Exit states are rendered in green diamonds, while non-exit states are rendered in red octagons (note that all three of the plateaus are open). The graphs were generated by the excellent graph layout program *yEd*,[1] and laid out using the "smart organic" method, which is based on the force directed layout paradigm. The plateaus were chosen so that they each contain roughly 1000 states; small enough to study the details within the plateaus, but containing enough states to ensure that large scale structures are observable.

The plateaus are quite fascinating and beautiful. The differences between plateaus from different instances, though not surprising, are quite dramatic. After looking at many samples of plateaus from instances of various types, it becomes apparent that plateaus belonging to a given instance tend to share many features. To borrow a colourful term from Douglas Hofstadter, plateaus from the same family of instances seem to share the same "spirit" [Hof85].

To illustrate this point, first consider the plateau shown in Figure 3.8. This is P^3_{max} (*i.e.* the largest open plateau region at level 3) from the instance in the *uf20* test set with median *lsc*. The plateau has 1222 states, 1047 of which are exits. The spirit of this plateau could be described as disarray — which is not surprising considering that the plateau comes from a randomly generated instance. It is interesting to note that the branching factor of the plateau is fairly low (approximately 0.185). As we will discuss shortly, all of the plateaus we studied have low branching factors. Also interesting is the distribution of non-exit states in the plateau. These states are distributed throughout the plateau, but they also appear in large clusters.

[1] *yEd* is freely available for download at `www.yworks.com`.

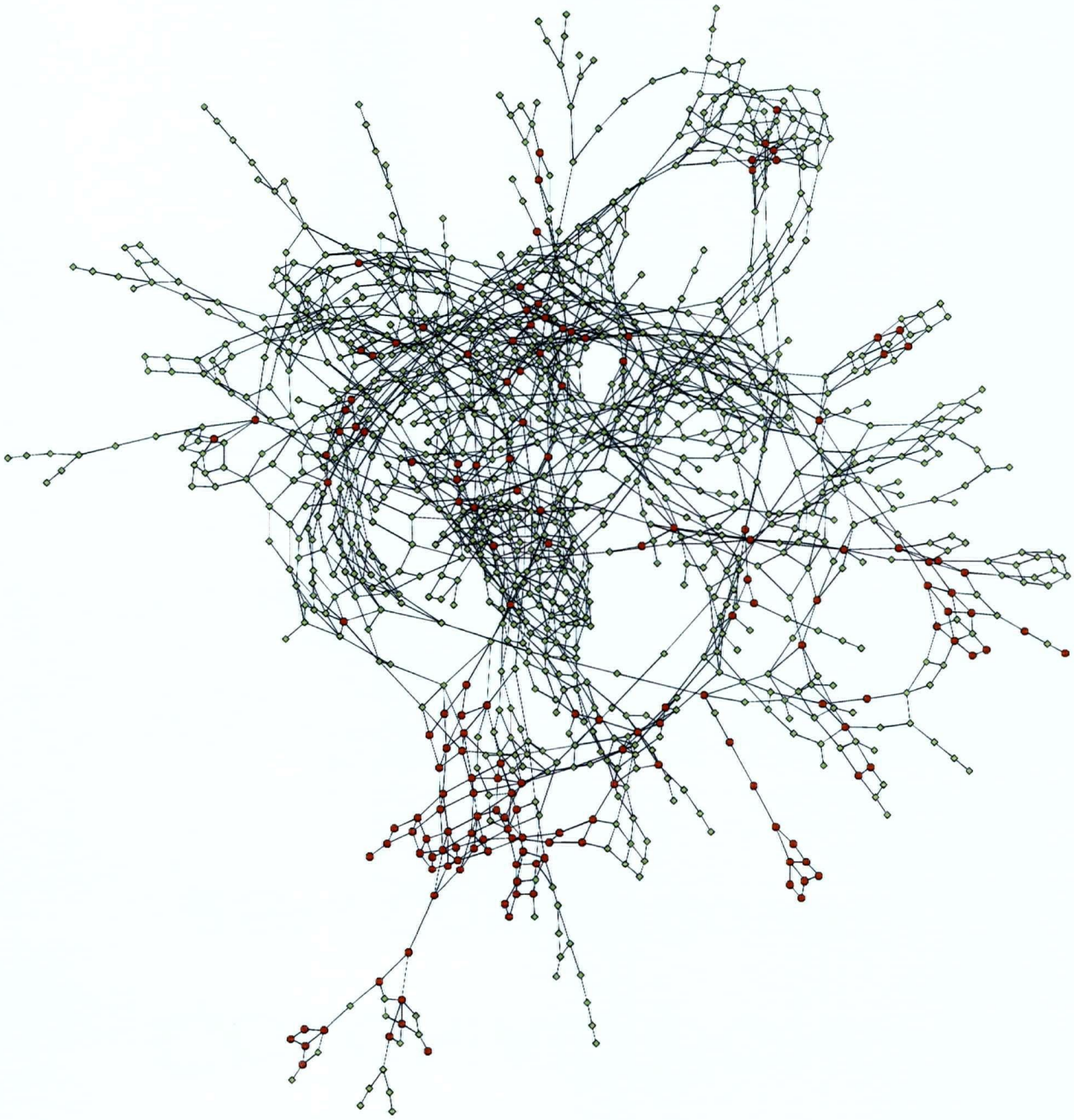

Figure 3.8: P^3_{max} from the instance in the *uf20* test set with median *lsc*.

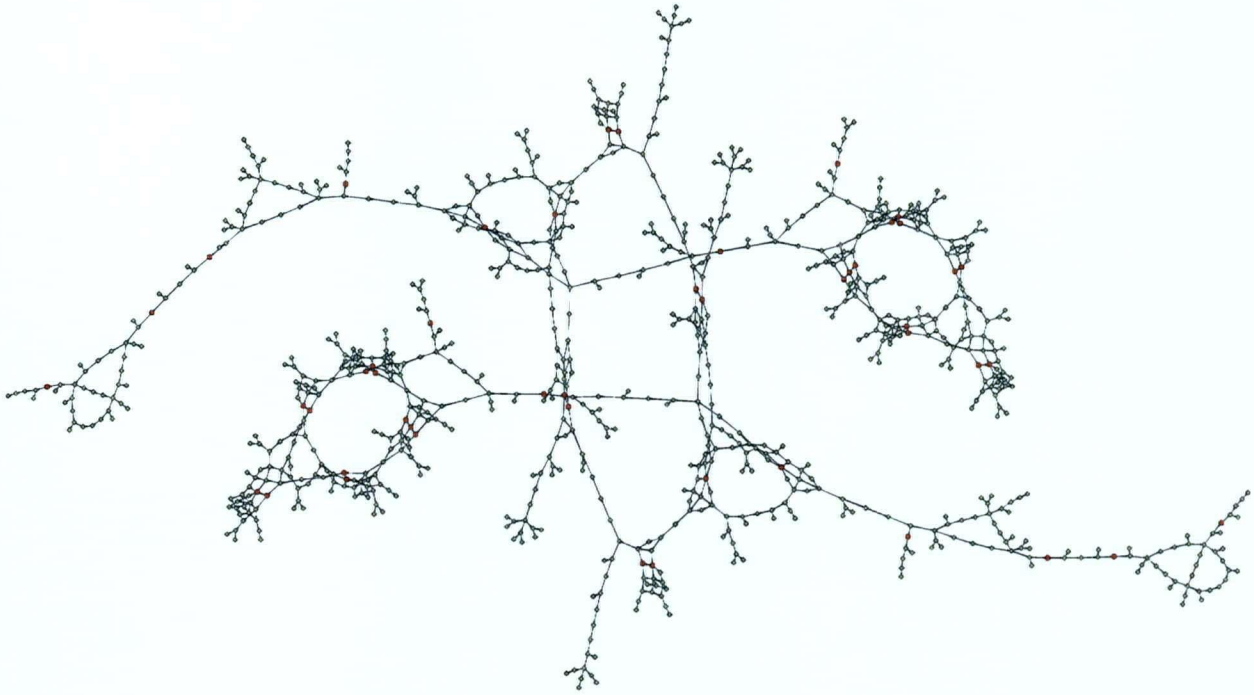

Figure 3.9: An example level 3 plateau from the `ais6` instance.

Because there are so many exits, the exit distance of this plateau is quite low — but there do exist states having plateau exit distance of 6. Finally, we note that the diameter of the plateau is actually larger than the number of variables.

We now turn our attention to the plateau shown in Figure 3.9, which has a very different spirit. This plateau is from an All-Interval-Series instance, and has level 3. It was chosen arbitrarily from all plateaus having approximately 1000 states — it contains 848 states, 800 of which are exits. This plateau is very beautiful — in fact, we found all of the large plateaus from the *ais* instance to be quite beautiful. The branching factor of the plateau is very low (approximately 0.04), and the plateau is perfectly symmetric.[2] Again, there are a large number of exits from this plateau resulting in a small exit distance, and the diameter is larger than the number of variables.

The last example plateau is shown in Figure 3.10. This level 5 plateau is from a highly structured parity learning instance — and this inherent structure is dramatically manifested in the plateau's spirit. Plateaus from the *parity* test set are characterised by large, very regular grids. The branching factor of this plateau is again quite low (approximately 0.09), and the plateau is comprised of multiple symmetric, well connected regions which are loosely coupled by a small number of edges. The plateau is open, but unlike the other sample plateaus, the fraction of exits is quite low (235 of the 969 states are exits), making this plateau a

[2]Note that it is not true that all plateaus in *ais* instances share this symmetry — many do, and all low level plateaus seem to have at least some symmetry, but they are not all as perfect as the example shown in Figure 3.9.

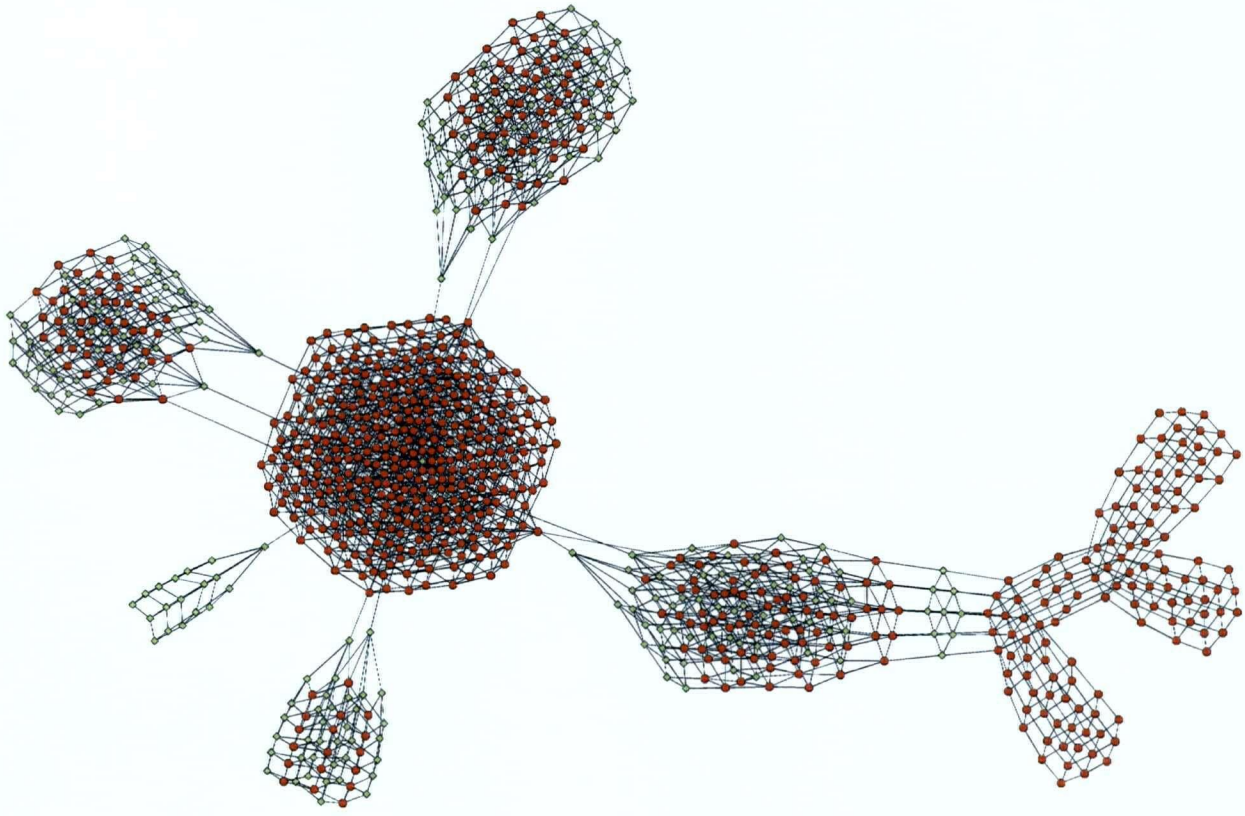

Figure 3.10: An example level 5 plateau from the `parity8` instance.

potential obstacle to local search algorithms. In fact, many of the low level plateaus in this instance look similar to this one, which helps to explain why these types of instances are so difficult for SLS algorithms. In contrast with the other example plateaus, the diameter of this plateau is less than the number of variables. Also note that in this instance, the exits appear in tightly connected clusters rather than uniformly distributed throughout the plateau — this makes it significantly more difficult to find an exit from these plateaus.

We should note now that these plateaus are somewhat atypical. As we have seen in the previous sections, the majority of plateaus are smaller than these. Nonetheless, there are many large plateaus – and as we will see in the next chapter, the larger plateaus often dominate the space actually encountered by SLS algorithms – and studying these larger plateaus has illustrated how dramatically different the search spaces of different instance types can be.

The remainder of this section studies the distributions of these properties over multiple plateaus from the same instances as well as over distributions of instances. In the following, the data for the UF-3-SAT test sets was collected in the same manner as in Section 3.2. We similarly sample plateaus for the structured instances, but since the structured test suites are composed of single instances, we sample 1000 plateaus at

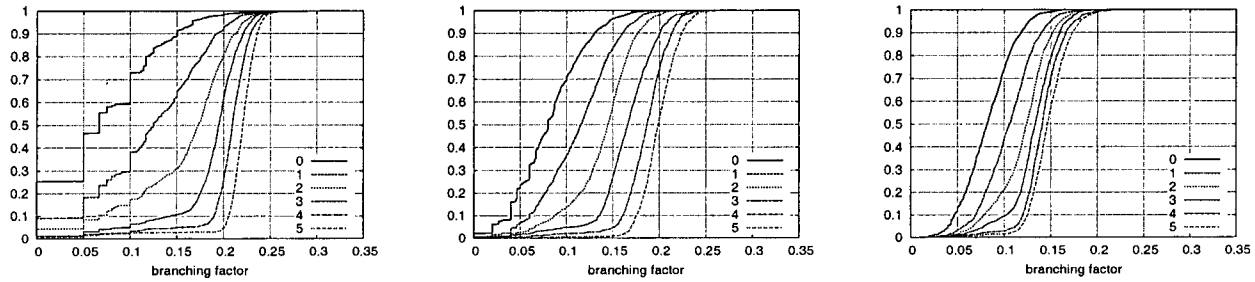

Figure 3.11: Distribution of branching factors for the UF-3-SAT test sets. Left to Right: uf20, uf50, uf100.

	level											
	0		1		2		3		4		5	
Instance	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
anom	0	(0)	0.007	(1.65)	0.019	(0.80)	0.025	(0.81)	0.036	(0.58)	0.053	(0.47)
medium	0	(0)	0.002	(2.30)	0.009	(0.87)	0.012	(0.72)	0.019	(0.48)	0.025	(0.44)
ais6	0	(0)	0.006	(1.70)	0.031	(0.45)	0.045	(0.41)	0.058	(0.32)	0.064	(0.36)
flat20-easy	0.209	(0.23)	0.245	(0.07)	0.257	(0.07)	0.276	(0.08)	0.298	(0.09)	0.321	(0.09)
flat20-med	0.069	(0.22)	0.120	(0.04)	0.167	(0.04)	0.205	(0.05)	0.240	(0.06)	0.272	(0.06)
flat20-hard	0.136	(0)	0.178	(0.06)	0.232	(0.04)	0.272	(0.06)	0.310	(0.06)	0.345	(0.07)
par8-1-c	0	(0)	0.023	(0)	0.043	(0.30)	0.056	(0.26)	0.072	(0.25)	0.086	(0.25)

Table 3.12: Summary statistics of the distribution of branching factors for the structured instances. For each instance and level, we should the mean and variation coefficient of the distribution.

each level for each structured instance.

3.5.1 Branching Factor

Figure 3.11 shows the distribution of branching factors for plateaus sampled from the *uf20*, *uf50*, and *uf100* test sets. The plateaus were sampled by using the same sampled states from Section 3.2. Starting from each sampled state, the plateau was expanded and the plateau branching factor was measured for each state, and then averaged to give the branching factor of each plateau. The entire distributions of branching factors are shown for each level in the figure.

The branching factors tend to be lower for lower level plateau regions; this is likely a consequence of the smaller plateau size at lower levels (it should be obvious that the branching factor will increase monotonically with plateau size, all else being equal). There is also more variance in the distribution of branching factors at lower levels; again, this is likely a consequence of the increased variance in the distribution of plateau sizes at lower levels. The branching factor also tends to decrease with the number of variables (an exception is the distribution for level 0; the average of the level 0 distribution increases slightly with the number of variables). Considering that the branching factors are normalised by the number of neighbouring states (*i.e.* the number of variables for SAT), this indicates that the rate of growth of the average branching factor is slower than linear in the number of variables.

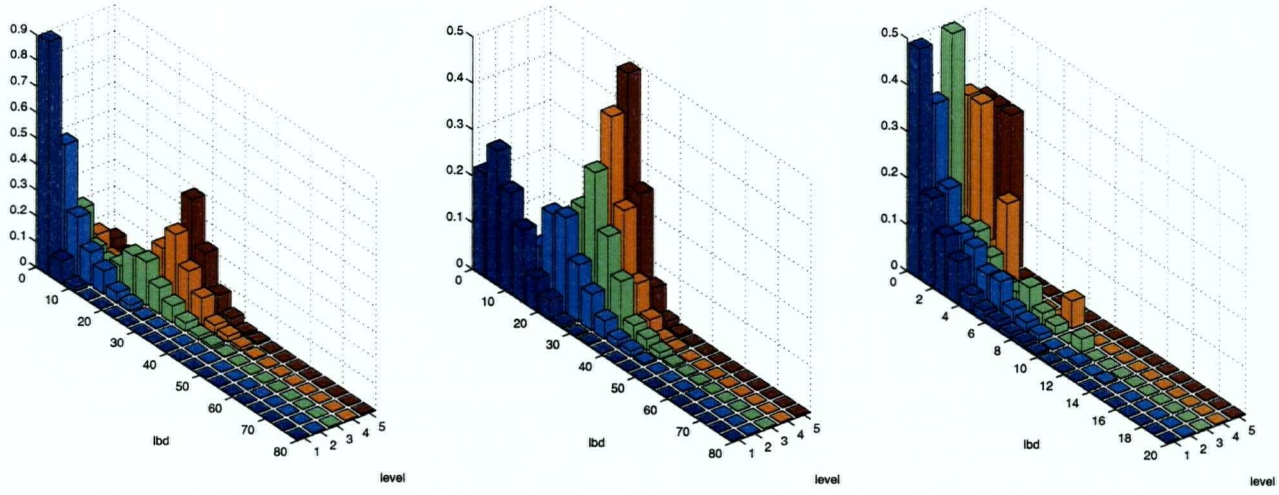

Figure 3.12: Distribution of lower bound on plateau diameter for the *uf20* test set. Left: All plateaus, Centre: open plateaus, Right: closed plateaus.

	level																	
	0			1			2			3			4			5		
Instance	$\bar{x}$	$\sigma_x/\bar{x}$	max	$\bar{x}$	$\sigma_x/\bar{x}$	max	$\bar{x}$	$\sigma_x/\bar{x}$	max	$\bar{x}$	$\sigma_x/\bar{x}$	max	$\bar{x}$	$\sigma_x/\bar{x}$	max	$\bar{x}$	$\sigma_x/\bar{x}$	max
anom	0	(0)	0	0.009	(1.75)	0.04	0.022	(0.88)	0.06	0.033	(1.04)	0.12	0.050	(0.75)	0.19	0.089	(0.64)	0.27
medium	0	(0)	0	0.002	(2.36)	0.02	0.011	(0.95)	0.04	0.016	(0.85)	0.06	0.027	(0.61)	0.09	0.038	(0.56)	0.11
ais6	0	(0)	0	0.008	(1.75)	0.03	0.264	(0.98)	1.30	0.764	(1.05)	3.20	0.521	(0.49)	1.97	0.403	(0.68)	2.95
flat20-easy	0.619	(0.35)	1.02	0.264	(0.18)	0.40	0.207	(0.13)	0.33	0.178	(0.10)	0.25	0.160	(0.10)	0.23	0.148	(0.10)	0.27
flat20-med	0.161	(0.44)	0.33	1.315	(0.08)	1.55	0.354	(0.09)	0.53	0.236	(0.20)	1.55	0.187	(0.16)	0.97	0.164	(0.10)	0.33
flat20-hard	0.189	(0.10)	0.22	0.930	(0.28)	1.23	0.263	(0.12)	0.38	0.183	(0.09)	0.25	0.153	(0.09)	0.27	0.139	(0.08)	0.22
par8-1-c	0	(0)	0	0.039	(0.20)	0.05	0.202	(0.60)	0.36	0.129	(0.95)	0.66	0.198	(0.69)	0.66	0.448	(0.91)	1.45

Table 3.13: Summary statistics of the distribution of lower bounds on plateau diameter for the structured instances. For each instance and level, we show the mean, variation coefficient, and max of the distribution.

Table 3.12 shows summary statistics of the distribution of branching factors for the structured test suites. Because the instances are so different, there is no clear picture emerging from the data, though we can make some observations. As we observed for the UF-3-SAT test sets, the branching factor increases with the level for the structured instances (again, likely a consequence of the increasing plateau size). We also note that the branching factors are much lower than observed for the random instances, with the exception of the graph colouring instances (recall that the search spaces of these instances consists of a single very large plateau). The branching factor of zero at level zero for all of the instances except the graph colouring instances is due to the fact that these instances have a single solution — and thus no neighbouring states.

3.5.2 Diameter

Figure 3.12 shows the distribution of a lower bound on the plateau diameter for the 20 variable UF-3-SAT test sets. This data is measured using the same sampled states from Section 3.2. From each sampled state s, $Plateau(s)$ is searched in a breadth-first manner, and the maximal length of the shortest path from s to any other state in $Plateau(s)$ is recorded. Because this value is only calculated for one starting state, this value may not equal the true plateau diameter, but it is clearly a lower bound of the true diameter. Additionally, because a maximum of 10^6 states are explored, this bound may not be tight for the larger test sets (this is because the plateaus encountered are very large, and the breadth-first search restricts the explored states to those with short path lengths from the starting state s), so we present data only for the *uf20* test set.

As we can see from the figure, there are a large number of cases in which the diameter is larger than the number of variables. This is interesting because it means that many of the plateaus are not contained in small sub-regions of the search space in which many of the variables have constant value for all states in the plateau. The larger plateaus at higher levels really are "interlaced" throughout the search space. Interestingly, the modes of the distributions at higher levels (level 3 and above) are exactly equal to the number of variables. Not surprisingly, the smaller closed plateaus have much smaller diameters as well.

We also study the lower bound on plateau diameter for the structured instances. Table 3.13 shows the mean, variation coefficient, and maximum of the sampled distributions. Because the number of variables is different between each of the structured instances, we show the lower bound on plateau diameter normalised by the number of variables. Thus, values greater than one indicate that the diameter is at least as large as the number of variables. Not surprisingly, the diameter data essentially reflects the plateau sizes (*c.f.* Tables 3.9, 3.10, and 3.11). The instances that had large plateaus, namely the all-interval series and flat graph colouring instances, did have plateaus whose diameter was larger than the number of variables. Recall that we expand a maximum of one million search states per plateau. This bound was exceeded for *ais6* at level 4, `flat20-easy` at level 1, `flat20-med` at level 2, and `flat20-hard` at level 1. Because the plateaus are so much larger than the cutoff, and we perform a breadth-first search, the measured lower bounds on the plateaus cannot be expected to be tight for the previously mentioned instances at higher levels, and thus we cannot extract any sense of the scaling behaviour of the diameter with level for these instances.

3.5.3 LMIN State Fraction

We now turn to a very interesting measure — the fraction of LMIN states in a given plateau. This value is interesting because even though a plateau may be open (*i.e.* it may have an exit), if most of the states are non-exit states then it may be very difficult for an SLS algorithm to find the exit state. In particular,

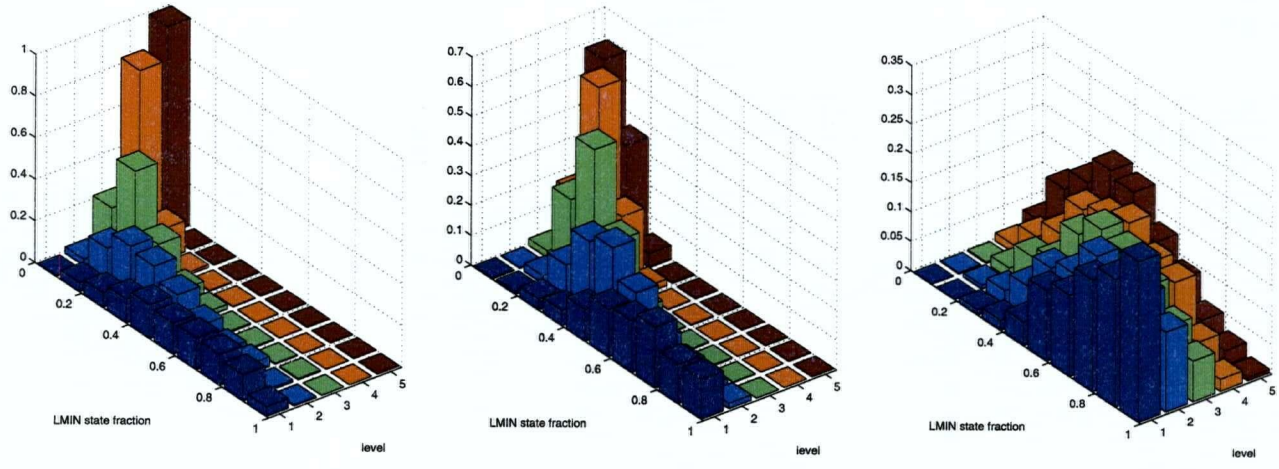

Figure 3.13: Distribution of the fraction of LMIN states for the UF-3-SAT test sets. Left to Right: uf20, uf50, uf100.

greedy algorithms that do not allow non-improving moves could become temporarily trapped in these open plateaus. Note that we only measure these values for open plateaus, since the fraction is one for closed plateaus, by definition. Furthermore, since every open plateau consists of LMIN, possibly IPLAT, and exit states — and since IPLAT states are exceedingly rare [Hoo98] — there is a (near) perfect inverse relationship between the fraction of LMIN states and the exit density of a plateau.

Figure 3.13 shows the distribution of LMIN state fractions for the UF-3-SAT test sets. For the smallest set, *uf20*, we see that at higher levels there are very few LMIN states in the plateaus meaning that most (or in some cases all) of the states are exit states — at high levels, it is easy to improve on the objective function. As the level decreases, we observe a monotonic increase in the fraction of LMIN states until at level one the range of the distribution essentially covers the interval $(0, 1)$. At level one, the mean LMIN state fraction is 0.58 — many of the plateaus have a high proportion of non-exit states.

The situation becomes even more interesting for the *uf50* and *uf100* test sets. As the number of variables increases, the LMIN state distributions shift toward one. Thus, as the problem size grows, even the open plateaus at low levels are made up of mostly of LMIN states. For the largest test set, *uf100*, the mean fraction of LMIN states at level five is 0.41 and this increases to 0.77 at level one, with the mode of the distribution approaching one. It should not come as a surprise that the fraction is so high, or that it increases with problem size given the (relative) difficulty of these instances.

Table 3.14 shows summary statistics from the corresponding distribution for the structured instances. We observe the same general trends in the structured instances as we do for the UF-3-SAT test sets. The fraction of LMIN states is higher at low levels, and tends to decrease with the level. One exception is the parity learning instance, for which the ratio stays relatively high even at higher levels. Interestingly, the rate of decrease for the planning (and to some extent the parity learning) instances is lower than for the all-

	level									
	1		2		3		4		5	
Instance	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
anom	0.613	(0.13)	0.587	(0.20)	0.458	(0.24)	0.407	(0.41)	0.472	(0.47)
medium	0.750	(0)	0.667	(0.25)	0.573	(0.30)	0.616	(0.31)	0.580	(0.37)
ais6	–	(–)	0.781	(0.20)	0.267	(0.78)	0.021	(0.64)	0.001	(0.35)
flat20-easy	0.403	(0.43)	0.235	(0.63)	0.151	(0.81)	0.077	(1.09)	0.035	(1.34)
flat20-med	0.681	(0.02)	0.464	(0.11)	0.266	(0.26)	0.141	(0.48)	0.056	(0.74)
flat20-hard	0.896	(<0.01)	0.695	(0.13)	0.442	(0.28)	0.243	(0.50)	0.099	(0.83)
par8-1-c	0.750	(0)	0.908	(0.17)	0.666	(0.26)	0.610	(0.28)	0.723	(0.25)

Table 3.14: Summary statistics of the distribution of the fraction of LMIN states in open plateaus for the structured instances. For each instance and level, we show the mean and variation coefficient of the distribution. The entries are marked '–' for level one of the `ais6` instance because there are no open plateaus at level one for this instance.

interval-series and flat graph colouring instances (and also seems to be slower than for the UF-3-SAT test sets). At higher levels, the all-interval-series and flat graph colouring instances have relatively few LMIN states, indicating that it is easy to get to lower levels in these instances.

In addition to these observations, we note also that all-interval-series and graph colouring instances seem to be much easier for SLS algorithms than the planning or parity instances (this is a very general statement, but it can be backed up with solid empirical evidence given for example in [Hoo98, HS04]). Thus, it seems reasonable to conjecture that there is a direct correlation between high LMIN state fractions at low levels and the intrinsic hardness of structured instances. Furthermore, if we consider only the flat graph colouring instances, we make an interesting observation that seems to corroborate our previous conjecture. At each and every level, the LMIN state fraction increases monotonically as we go from the easiest to the hardest instance. Intuitively, this must be the case because, as we discussed in Section 3.4.1, the search spaces of the flat graph colouring instances are dominated by a single large open plateau. If it were possible to easily find an exit from each of these large open plateaus, then the graph colouring instances would be trivial to solve. Since they are not trivial, there must be other sources of hardness such as the one we just discussed.

3.5.4 Exit Distance

Figure 3.14 shows the distribution of exit distances for the UF-3-SAT test sets. This data is very interesting, because it gives an indication of the typical number of steps an SLS algorithm must take on an open plateau before reaching an exit. Obviously, the higher the exit distance, the more difficulties a plateau will pose for a local search algorithm. This data was also measured using the same sampled states from Section 3.2. Starting from a sampled state s, $Plateau(s)$ was explored using breadth-first search, and the distance to the closest exit was recorded. For obvious reasons, the distribution is taken only over open plateaus.

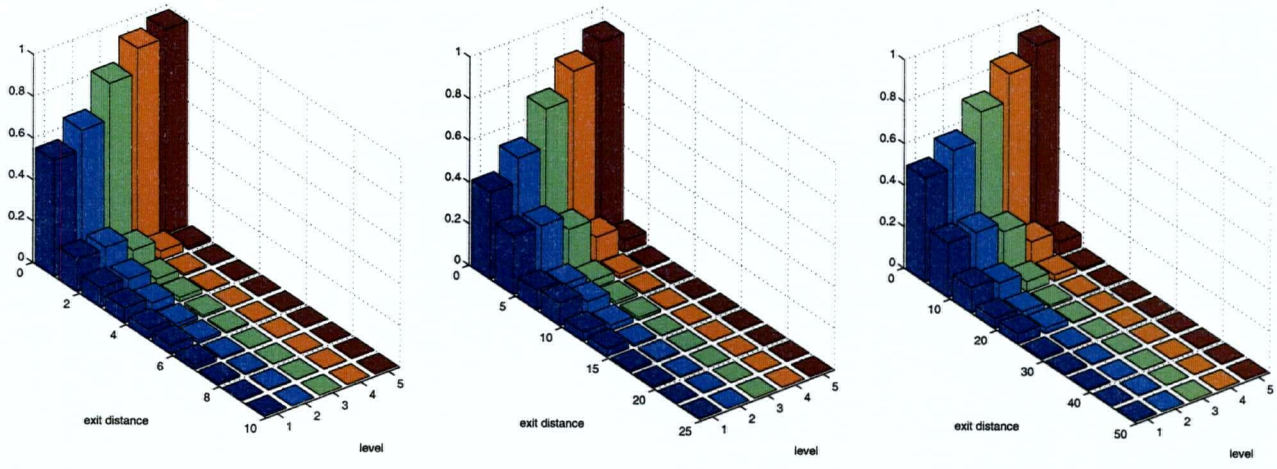

Figure 3.14: Distribution of exit distances for the UF-3-SAT test sets. Left to Right: uf20, uf50, uf100.

	level									
	1		2		3		4		5	
Instance	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
anom	0.777	(0.54)	0.910	(0.70)	0.771	(0.81)	0.964	(0.82)	1.385	(0.96)
medium	0.811	(0.48)	0.986	(0.72)	1.036	(0.76)	1.383	(0.86)	1.392	(0.90)
ais6	–	(–)	2.917	(1.01)	0.430	(1.44)	0.045	(4.61)	0.002	(25.10)
flat20-easy	1.551	(0.90)	0.709	(1.11)	0.392	(1.42)	0.187	(2.11)	0.086	(3.29)
flat20-med	3.168	(0.68)	1.250	(0.75)	0.661	(0.98)	0.369	(1.37)	0.148	(2.40)
flat20-hard	11.476	(0.64)	2.087	(0.69)	0.879	(0.87)	0.468	(1.20)	0.179	(2.16)
par8-1-c	2.127	(0.42)	9.003	(0.58)	2.071	(0.96)	2.123	(1.03)	4.411	(1.13)

Table 3.15: Summary statistics of the distribution of exit distances in open plateaus for the structured instances. For each instance and level, we show the mean and variation coefficient of the distribution. The entries are marked '–' for level one of the `ais6` instance because there are no open plateaus at level one for this instance.

The results are very interesting. At higher levels, even though the size of the sampled plateau is typically very large, the exit distances are very small – meaning that they should be easy to find. As the level decreases, the expected exit distance increases, indicating that even open plateau regions at lower levels may pose a problem for SLS algorithms. We do note, however, that the exit distance distributions at all levels are skewed toward shorter distances and the expected exit distances are small, even at lower levels. Thus, even though the plateaus encountered by GSAT are larger at higher levels, they are typically open with very small exit distances. The troublesome plateaus are those smaller plateaus at lower levels that are either closed, or have high exit distances.

Table 3.15 shows summary statistics for the distribution of exit distances measured for the structured instances. There are some very interesting differences between these results and those for the distributions of LMIN state fractions in the previous section. The expected exit distance increases with the level for the planning instances, even though the LMIN state ratio decreases. The mean exit distances for the planning

instances were very low for all levels, indicating that, for these instances, it must be closed plateau regions that are the major source of hardness since open plateaus seem quite easy to escape from.

The all-interval-series instance is quite interesting. There are no open plateaus at level one, and the exit distance at level two moderately high (2.917). Note however, that the exits from level two must lead to solutions. At higher levels, the exit distance is negligible. When we combine this information with the fact that the LMIN state ratio is very low at higher levels, increasing to 0.781 at level two, it should not be surprising that this instance is quite easy for SLS algorithms.

The data from the parity learning instance is also quite interesting. Along with the high LMIN state ratio mentioned in the previous section, the `par8-1-c` instance also has very high exit distances, even at higher levels. Thus, even though the parity learning instance has a large fraction of open plateaus at each level, it is very difficult to find an exit from many of the open plateaus. Figure 3.10 gives a good visual indication of the internal plateau structure underlying these observations.

Finally we consider the flat graph colouring instances. The mean exit distance decreases with level for these instances, as the LMIN state fraction did. Also, corresponding to our observations for the LMIN state fraction, the mean exit distance monotonically increases from the easiest to the hardest instance at each level, demonstrating nicely which search space features make instances from the same distribution more difficult to solve than syntactically similar instances.

3.6 Summary

In this chapter, we have introduced the basic search space features that we study in this thesis, in particular the plateau. We studied the plateau characteristics of the UF-3-SAT instance distribution, attempting to replicate and extend the results of Frank *et al.* [FCS97] by studying the plateaus encountered by the GSAT algorithm. We studied plateau properties such as size of plateaus, and the fraction of closed plateaus. We observed that the average fraction of closed plateaus decreases rapidly with the level, and that open plateaus tend to be much larger than closed plateaus, especially at higher levels. Our results differ from those of [FCS97] in that we report much larger plateau sizes. We have ensured that our results are correct, and provide some possible explanations for the discrepancies.

Next, we studied the plateau characteristics of individual UF-3-SAT instances by exhaustively exploring the low level regions of the search space. This allows us to study the number of plateaus at each level, and we note that the number of plateaus increases rapidly with the level and that the majority of high level plateaus are open. We also studied the size of plateaus, observing that the size of closed plateaus decreases with level — thus, closed plateaus are only prominent at very low levels in the search space. Interestingly, we observe that while there are many open plateaus at high levels, the majority of these plateaus are very

small (and consist entirely of exits), while one single large plateau contains most of the states at higher levels. This "one-big-plateau" phenomenon is very interesting because it indicates that the entire low level search space is connected, and thus an algorithm never has to go to very high levels to reach any point in the low level search space.

We performed a similar analysis on individual structured instances, with similar results. We observe that the number of plateaus increases (exponentially) with the level, and that the number of open as well as closed plateaus increases with the level (recall that for the random test sets, the number of closed plateaus decreases with level). When studying the fraction of closed plateaus at each level, we observed that, while the fraction of closed plateaus tended to decrease with level, the harder structured instances tended to have a higher proportion of closed plateaus at each level. We also noted that closed plateaus were typically quite small, and that they were largest for the hardest instance. We observed that the size of open plateaus increases with level, and that the median sizes were fairly small. The only structured instances to exhibit the "one-big-plateau" phenomenon were the graph colouring instances.

Finally, we studied the internal structure of plateau regions. We observed very low branching factors for all of the instances studied, indicating that plateaus are rather "brittle" (this has been previously reported in [Hoo98]), though the branching factors increase with level. Next, we study the diameter of plateaus and observe that there are many cases, especially at higher levels, where the diameter is greater than the number of variables. In general, the diameter seemed to be proportional to the plateau size. We next studied the fraction of LMIN states from a plateau, and found that at high levels there were very few LMIN states (indicating that it is very easy to find an exit to a lower level state from high level plateaus), and that the LMIN state fraction decreases with level, though there were typically a substantial number of exits from plateaus at all levels. Lastly, we studied the exit distances of plateaus, and observed that at high levels the exit distance was typically very low, while at low levels the exit distances were occasionally very high.

Chapter 4

Plateau Connectivity

The previous chapter studies the distribution of plateaus in the search space and properties of the plateaus, which is crucial to gain an understanding of what these components of the search landscape look like. This chapter takes an alternative direction, and focuses rather on the connections between plateau regions, and how the model consisting of these interconnected plateaus accurately characterises the search space of the instances that we study. We will demonstrate how understanding properties of this model of the search space can aid in the understanding of algorithm behaviour. This understanding provides insight into how existing local search algorithms can be improved and new algorithms can be developed.

This chapter is structured as follows. We first formalise the definitions of concepts that are central to this chapter, and motivate the research. Then we begin our study of Plateau Connectivity Graphs by undertaking an empirical study of their basic properties. Next, we demonstrate how Plateau Connectivity Graphs can be used to approximate the likelihood that an iterated improvement algorithm will reach a solution (rather than get caught in a local minimum). Following this, we introduce a novel method for classifying search space traps, and empirically demonstrate how these traps adversely affect local search algorithms. Finally, we summarise and discuss the main results from the chapter.

4.1 Definitions and Motivation

We begin with the central definition of the chapter; that of a Plateau Connectivity Graph. A Plateau Connectivity Graph (PCG) is a directed graph in which each node represents a plateau, and there are edges between nodes if and only if the corresponding plateaus are connected. Two plateaus are considered to be connected if they each contain at least one state that neighbours a state in the other plateau. The following definition formalises this concept:

Definition 4.1. (Plateau Connectivity Graph)
A *Plateau Connectivity Graph* (PCG) is a directed graph $G = (V, E)$. Each node $v \in V$ represents an entire plateau in the search landscape $L = (\mathcal{S}, \mathcal{N}, g)$. There is an edge $(u, v) \in E$ from u to v if and only if there exists at least one pair of states $s_u \in u$ and $s_v \in v$ such that $level(s_u) > level(s_v)$

and s_u and s_v are neighbours under the given neighbourhood relation $\mathcal{N}$. Such a pair of states is called an *exit-target pair* from u to v. A *Partial PCG* is a PCG in which not all of the plateaus (or corresponding edges) in the search space are represented. Unless otherwise noted, we will assume throughout this thesis that we are referring to partial PCGs. □

Intuitively, PCGs form a fairly representative, but still small and simple, model of the features of the search space that have a large impact on SLS algorithm behaviour. For example, we can consider a simple local search algorithm which always always makes either sideways or downwards moves, and chooses randomly between all exit targets. Such an algorithm would always either move within a plateau, or follow one of the PCG edges — thus, the algorithm would walk strictly along the PCG edges. The intuition is similar for more complicated algorithms — because most SLS algorithms for SAT are guided by the same objective function, we conjecture that PCGs will capture the important aspects of the search space, and thus will adequately model the behaviour of even complex algorithms.

In order to better understand how PCGs accurately model the search space, we make distinctions between different types of PCG nodes, depending on the number of incoming and outgoing edges. The following definition formalises our classification.

Definition 4.2. (PCG Node Types)
Given a (partial) PCG $G = (V, E)$, we can define the following sets:

$$R = \{v \in V | \neg \exists u : (u, v) \in E\} \tag{4.1}$$

$$L = \{u \in V | \neg \exists v : (u, v) \in E\} \tag{4.2}$$

$$M = \{u \in L | level(u) > 0\} \subseteq L \tag{4.3}$$

$$S = \{u \in L | level(u) = 0\} \subseteq L \tag{4.4}$$

R is the set of all roots of the graph; that is, the set of nodes with no incoming edges. L is the set of all leaves of the graph; the set of nodes with no outgoing edges. Obviously, elements of L correspond to closed plateaus. M is the set of closed plateaus that are not solution plateaus (clearly $M \subset L$, and $M = L$ if and only if there are solutions). S is the set of solution plateaus (clearly $S \subset L$, and $S = L$ if and only if there are no closed plateaus that are not solutions). □

Note that the current definition of PCGs can only portray whether two plateaus are connected (meaning that it is possible for a local search algorithm to move directly between the two plateaus). Intuitively, however, we would expect that certain connections may be more important than others. For example, consider a plateau P_1 having a large number of exits leading to another plateau P_2, but only a few leading to a third plateau P_3. The connection between P_1 and P_2 seems somehow more important, since in practice we would expect an SLS algorithm to follow that edge more than the other. To capture this notion of relative importance, we introduce edge weights for PCGs. The following definition formalises this concept.

Definition 4.3. (Weighted PCG)
Given a search landscape $L = (S, \mathcal{N}, g)$, a *weighted PCG* is a PCG augmented with a function $w : E \mapsto [0, 1]$ having the following property:

$$\forall u \in V : \sum_{\{v \in V | (u,v) \in E\}} w(u, v) = 1 \tag{4.5}$$

That is, the sum of the weights of all outgoing edges must sum to 1 for every PCG node. Thus, a weighted PCG G is represented by a triple: $G = (V, E, w)$, where V and E are the vertex and edge sets of the corresponding unweighted PCG, and w is the weight function. □

There are indefinitely many ways to construct the function w, but there are two major factors that w should attempt to capture. Firstly, as hinted at above, $w(u, v)$ should be proportional to the number of neighbouring states in u and v. Secondly, w should reflect the behaviour of the algorithm it is attempting to model. For example, if we would like to model a best improvement algorithm with $w(u, v)$, then for every exit state in u, only the target state having the best score should contribute to the weight (reflecting the fact that the other targets would never be selected by the algorithm). Note that w only approximates the expected algorithm behaviour on the plateau — in particular, we are making the simplifying assumption that the SLS algorithm being modelled is able to choose uniformly and randomly from all exit states on the plateau, which is obviously not the case in existing algorithms.[1]

In this chapter, we opt for a fairly general weight function to ensure that we are not modelling features that are overly algorithm specific. Our weight function basically gives equal precedence to every exit-target pair from every source plateau. Formally, we first define the set of all exit-target pairs between two plateaus u and v:

$$Ext_{uv} = \{(s_u, s_v) \mid s_u \in u \wedge s_v \in v \wedge \mathcal{N}(s_u, s_v) \wedge g(s_u) > g(s_v)\} \tag{4.6}$$

[1]The exit taken by an SLS algorithm from a given plateau depends on many factors, including the state at which the algorithm entered the plateau, possible clustering effects which could make some exits inaccessible from neighbours in the plateau, and even higher-order effects such as search history (which could affect the exits chosen by, for example, a tabu search algorithm).

Next, we define the set of all exit-target pairs having u as the target:

$$Ext_u = \bigcup_{v \in V} Ext_{uv} \tag{4.7}$$

Finally, using these sets we are able to define our weight function:

$$\overline{w}(u,v) = \frac{\#Ext_{uv}}{\#Ext_u} \tag{4.8}$$

Note that $\overline{w}$ gives equal precedence to every exit-target pair leaving a plateau. This ensures that we are not making the function algorithm-specific by only including certain exit-target pairs, but it also ensures that stronger connections are reflected by higher weights. In practice, we have found that this weight function achieves a good compromise between simplicity, generality, and predictive power.

Now that we have fully defined PCGs, we will aid the intuitive understanding of these structures through examples. Figures 4.1 and 4.2 show partial PCGs for an easy instance and the hardest instance from the *uf20* test set, respectively. All nodes are labelled $x.y$, where x is the level of the plateau and y is simply an identifier that is unique to the level. Open plateaus are rendered in white ovals, closed plateaus in red squares, and solution plateaus in green triangles. The edge labels are the $\overline{w}$ values, and dashed edges represent those edges with weight less than 0.05. Edges with weight greater than 0.3 are rendered with bold lines.

In order to clearly illustrate these graphs we had to omit some details. First, plateaus are only shown if they were encountered at least one time over the course of 1000 runs of GWSAT (the graphs remain unchanged if other algorithms, such as WalkSAT or Novelty$^+$ are used). This results in only P_{max} being shown at higher levels (*c.f.* Section 3.3.4). Clearly this will not affect the conclusions that we can draw from studying the PCGs, since if the small degenerate plateaus at higher levels are very rarely encountered then they cannot have a significant impact on the behaviour of the SLS algorithms. The second detail omitted is that edges with weights lower than 0.01 are not displayed.[2] This is simply because again we assume that very weak edges will have little impact on SLS algorithm behaviour but they clutter up the graphs and make it difficult to see the more important structures.

We now turn our attention to Figure 4.1, which represents an easy instance from the *uf20* test set. The main feature of note is that the search space appears to be structured essentially as a large funnel directing all search trajectories toward the solution. Given the structure of the PCG, it is not at all surprising that the instance is easy. There are a fair number of closed plateaus, but, with the exception of the plateau labelled `1.1`, none of them are strongly connected to higher-level plateaus — meaning that we would not expect an

[2] An exception to this rule is made if removing these edges would leave a node with no incoming edges; in this case the incoming edge with the highest weight (still less than 0.01) is left in the graph. These weak edges are rendered in dashed grey lines.

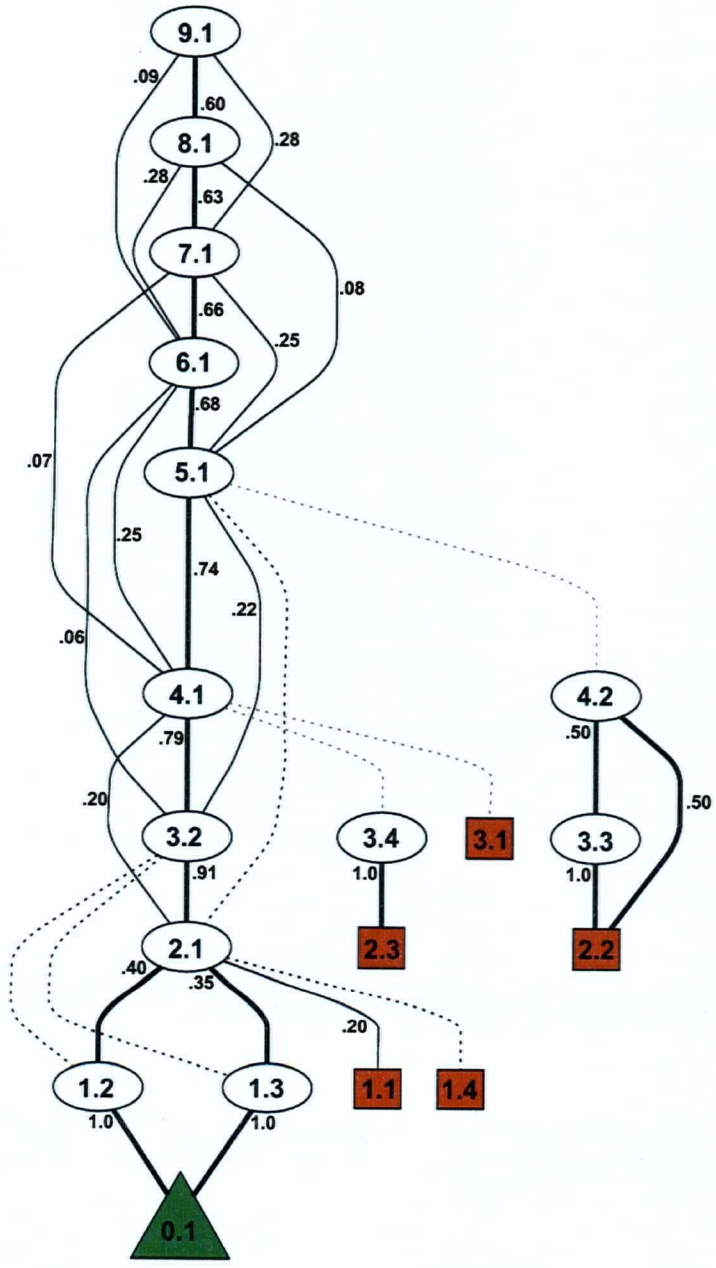

Figure 4.1: Weighted, partial PCG for an easy *uf20* instance.

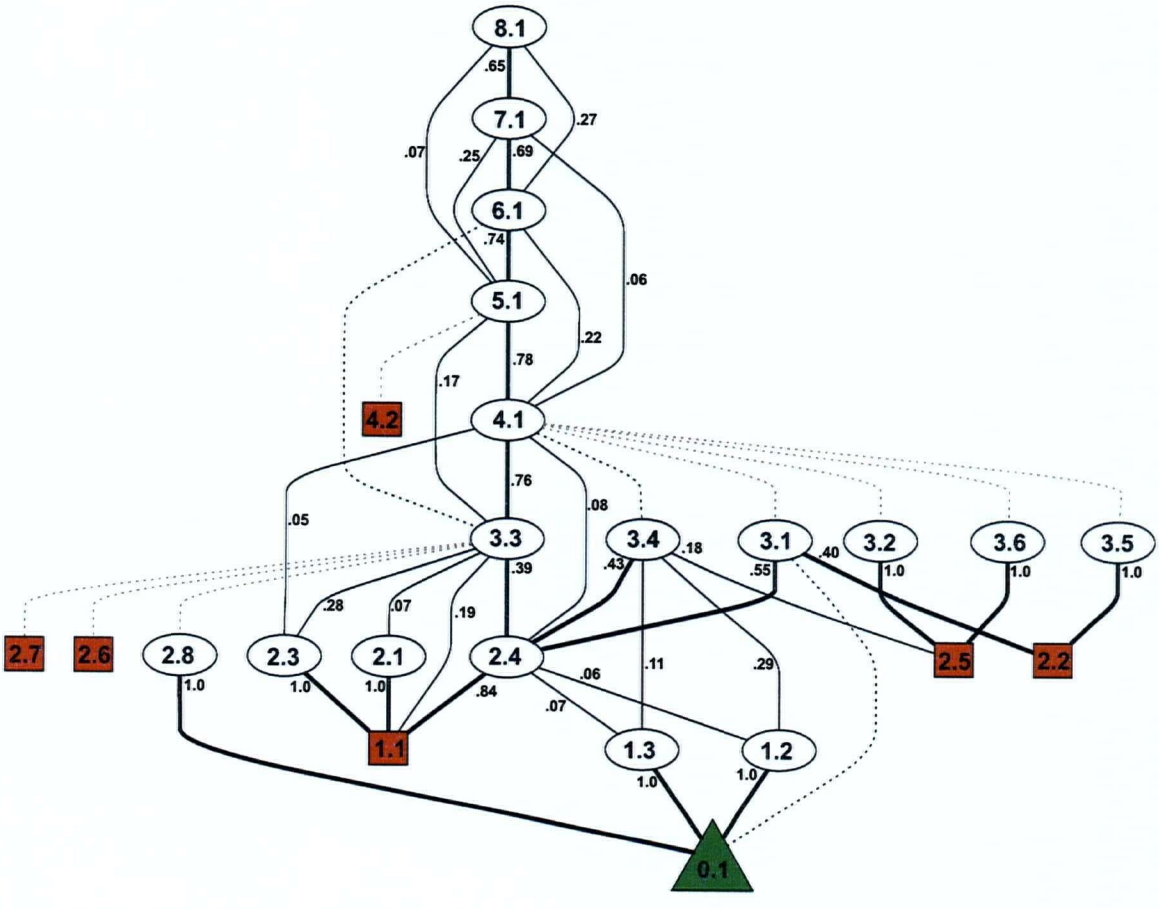

Figure 4.2: Weighted, partial PCG for the hardest *uf20* instance.

SLS algorithm to be attracted to those closed plateaus.

Figure 4.2, representing the hardest of the *uf20* instances, is notably different. While the PCG seems similar at higher levels (only P_{max} was encountered, and each high level plateau is strongly connected to the large plateau directly below), there are substantial differences at lower levels. Firstly, there are considerably more low-level plateaus that were encountered during the 1000 runs, and many of these are either closed, or are well-connected to closed plateaus. Secondly, and more importantly, if we follow the highest-weighted edges down from the high level plateaus, the majority of the edges lead to the closed plateau `1.1`, not to a solution. Clearly this is where the difficulty of this instance comes from. Assuming that the edge weights are representative of how a real SLS algorithm would behave on this instance, we can see that the vast majority of trajectories lead to plateaus `2.1`, `2.3`, and `2.4`. Both `2.1` and `2.3` are only connected to the closed plateau `1.1`, and plateau `2.4` has an edge with weight 0.84 leading to `1.1`.

It is interesting to note that these two instances have a similar number of closed plateaus, and they both only have one well-connected closed plateau, yet the local search cost of the instances is drastically different. As we have anecdotally explained, the key difference lies in the connections between the various plateaus. The remainder of this chapter will further develop these ideas and empirically demonstrate that hard instances consistently have such search space structures.

Finally, Figure 4.3 shows the PCG for the smallest structured instance, `anom`. As we can see, this PCG is substantially more complex than those of the random instances. Because of this complexity, we were unable to add edge weights without cluttering up the figure. Instead, the colour of every edge is directly proportional to the weight; darker edges indicate higher edge weights. The PCG is broken into 13 disjoint connected components,[3] some containing many plateaus some only two. This instance has a single model, and thus only one solution plateau, labelled `0.1`. The largest connected component actually contains the solution. All paths in all of the other components lead to closed plateau regions, indicating that there are large portions of the search space from which it is impossible to reach a solution except by first going to higher levels. Because they appear to be so attractive and well-connected, we would expect that the closed plateaus labelled `1.1` and `1.3` would be major factors contributing to the difficulty of this instance. This will be investigated further in later sections of this chapter.

4.2 PCG Properties

In this section, we study some basic graph-theoretic properties of PCGs in order to paint an empirical picture of what typical PCGs look like. We use the same test suites as in the previous chapter, allowing us to complement the existing results w.r.t. plateau properties with these new higher-level PCG results. In this

[3]Note that this disjointness is an artifact resulting from the omission of plateaus from the partial PCG — if all encountered plateaus up to level 9 are included in the partial PCG then the resulting partial PCG is connected.

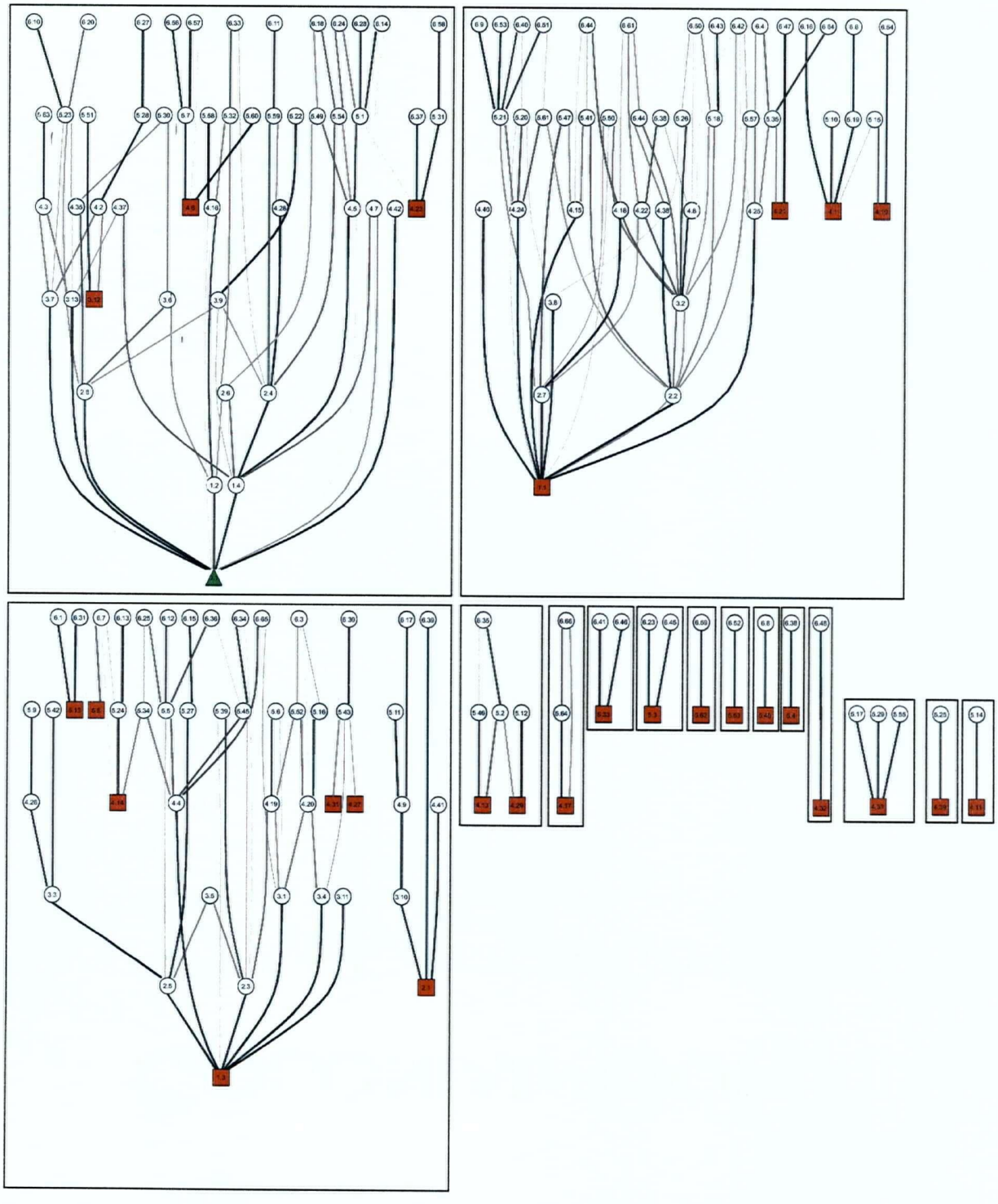

Figure 4.3: Weighted, partial PCG for the structured `anom` instance. This figure shows a single PCG, but the PCG consists of multiple disjoint components. The vertical ordering of the nodes within each component represents the relative levels the corresponding plateaus. Nodes representing plateaus at the same level are aligned vertically where possible.

	level									
	1		2		3		4		5	
Test Set	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
uf20	1.10	(0.16)	1.94	(0.17)	2.72	(0.13)	3.32	(0.11)	3.84	(0.10)
uf25	1.08	(0.12)	1.89	(0.16)	2.57	(0.11)	3.07	(0.09)	3.50	(0.08)
uf30	1.09	(0.17)	1.90	(0.13)	2.49	(0.10)	2.97	(0.09)	3.40	(0.07)
uf35	1.09	(0.14)	1.81	(0.15)	2.41	(0.11)	2.90	(0.09)	3.28	(0.08)
uf40	1.09	(0.11)	1.77	(0.14)	2.30	(0.10)	2.78	(0.08)	3.16	(0.07)
uf45	1.08	(0.08)	1.77	(0.14)	2.32	(0.10)	2.77	(0.08)	3.15	(0.07)

Table 4.1: Summary statistics of the distribution of the out-degree of PCG nodes from the UF-3-SAT instances. For every test set and level we show the mean and coefficient of variation of the distribution (over all instances) of the average out-degree of all open plateau nodes (over all open plateau nodes in each instance).

	level									
	1		2		3		4		5	
Instance	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
anom	1.000	(0)	1.250	(0.35)	1.607	(0.37)	2.216	(0.41)	2.696	(0.45)
medium	1.000	(0)	1.061	(0.22)	1.537	(0.39)	1.886	(0.44)	2.307	(0.49)
flat20-easy	1.644	(3.32)	2.744	(3.35)	2.656	(1.81)	2.970	(0.90)	4.104	(0.80)
flat20-med	2.256	(6.40)	1.913	(8.31)	2.301	(3.32)	3.471	(1.96)	4.194	(1.69)
flat20-hard	1.263	(0.88)	2.579	(2.61)	1.794	(1.89)	2.942	(0.84)	3.770	(0.86)
par8-1-c	1.000	(0)	1.067	(0.23)	2.889	(1.19)	2.022	(0.64)	2.168	(1.28)

Table 4.2: Summary statistics of the distribution of the out-degree of PCG nodes from the structured instances. For every test set and level we show the mean and coefficient of variation of the distribution (over all open plateau nodes in the instance) of the out-degree of the open plateau nodes.

section, we measure the distribution of out-degrees of PCG nodes corresponding to open plateaus. We also measure the distribution of average and maximum target depths for the same PCG nodes (these terms will be defined shortly). The PCGs were constructed from the plateau data presented in Section 3.3. To generate the PCGs, we started with the highest level plateaus and iteratively calculated Ext_u for the source plateau, and Ext_{uv} between u and all low level plateaus. Finally, $\overline{w}(u,v)$ was calculated according to Equation 4.8.

4.2.1 Out-Degree

Given an PCG $G = (V, E)$ and an open plateau u at level k, the out-degree is simply:

$$\#\{v \in V \mid \exists s_u \in u, s_v \in v : \mathcal{N}(s_u, s_v) \wedge level(u) > level(v)\} \tag{4.9}$$

For each instance and level, we calculated the out-degree of each node and then averaged these out-degrees to give an average out-degree for the given instance and level.

Table 4.1 shows summary statistics of the distribution of out-degrees of PCG nodes for the UF-3-SAT instances. There are some interesting trends present in the table. The out-degree tends to increase with level, which is not surprising considering that there are fewer plateaus at low levels. More interesting is the fact

that the average out-degrees tend to decrease with problem size. This trend is present at all levels (including level 1), and the relationship is monotonic with the single exception of level 3 of uf40 being slightly lower than level 3 of uf45. The trend is particularly puzzling considering that the number of plateaus increases with problem size. Also notable from the table is the very low coefficient of variation in the distributions — because this is the coefficient of variation of the average out degrees, it indicates that the average values are similar across all instances in each test set. Finally, we observe that the coefficient of variation decreases both with level and problem size.

We also consider similar data for the structured instances, and show this in Table 4.2. Note that the data in this table have not been double-averaged, and so the variation reported is the variation of the distribution over all open plateau nodes at each level. We see that the data is quantitatively similar to that for the structured instances. The out-degrees of the nodes tend to increase with the level. Interestingly, the magnitude of the out-degrees for the `anom`, `medium`, and `par8-1-c` instances are lower than the corresponding values from the random test sets (with the exception of level 3 of the `par8-1-c` instance, which has a higher than expected average out-degree). Close examination of these instances reveals that when there are a large number of plateaus present at a given level, many of these are very small and only connected to a single lower level plateau — resulting in an out-degree of one for many of the plateaus.

The data presented in this section has interesting implications for algorithm mobility. Because the out-degrees of many of the plateaus are quite low, especially at lower levels, it becomes difficult for local search algorithms to directly move around the low level search space. Especially considering that many plateaus are connected to only one lower level plateau, it may be necessary for SLS algorithms to occasionally "retreat" to significantly higher levels of the search space in order to avoid being restricted to certain regions of the search space. This is illustrated quite dramatically in Figure 4.3 (the PCG for the `anom` instance), where we see large disjoint subgraphs in the PCG between which it is impossible for an SLS algorithm to move without moving to higher search space levels.

4.2.2 Target Depth

Tables 4.3 and 4.4 shows statistics of the average and maximum target depth, respectively, for the UF-3-SAT test sets. Given a PCG edge (u, v), the target depth is simply the difference in level between the two plateaus. Target depth gives a sense of the expected improvement we expect to observe when exiting a plateau — high target depth values indicate that it is possible to descend multiple levels by following a single edge.

Interestingly, there is quite a range in the target depth values. Both the average and maximum target depths are (trivially) 1 at level 1, and increase with the level. The rate of increase appears to be linear with the level, but the maximum values are only 1.51 (*uf20*, level 5), and 2.12 (*uf20*, level 5) for the average and

	level									
	1		2		3		4		5	
Test Set	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
uf20	1.00	(0)	1.17	(0.10)	1.29	(0.06)	1.40	(0.04)	1.51	(0.03)
uf25	1.00	(0)	1.12	(0.06)	1.23	(0.04)	1.33	(0.03)	1.43	(0.03)
uf30	1.00	(0)	1.11	(0.05)	1.21	(0.04)	1.30	(0.03)	1.39	(0.03)
uf35	1.00	(0)	1.10	(0.05)	1.19	(0.04)	1.27	(0.03)	1.36	(0.03)
uf40	1.00	(0)	1.09	(0.05)	1.16	(0.03)	1.24	(0.03)	1.32	(0.03)
uf45	1.00	(0)	1.09	(0.05)	1.16	(0.03)	1.23	(0.02)	1.30	(0.02)

Table 4.3: Summary statistics of the distribution of the average target depth of PCG nodes from the UF-3-SAT instances. For every test set and level we show the mean and coefficient of variation of the distribution (over all instances) of the average target depth of the open plateau nodes (over all open plateaus at the given level).

	level									
	1		2		3		4		5	
Test Set	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
uf20	1.00	(0)	1.33	(0.12)	1.61	(0.09)	1.87	(0.07)	2.12	(0.06)
uf25	1.00	(0)	1.26	(0.10)	1.50	(0.07)	1.74	(0.06)	1.98	(0.05)
uf30	1.00	(0)	1.24	(0.08)	1.46	(0.07)	1.68	(0.06)	1.91	(0.05)
uf35	1.00	(0)	1.21	(0.08)	1.41	(0.07)	1.63	(0.06)	1.84	(0.05)
uf40	1.00	(0)	1.19	(0.08)	1.37	(0.06)	1.57	(0.06)	1.77	(0.06)
uf45	1.00	(0)	1.18	(0.07)	1.36	(0.07)	1.54	(0.05)	1.73	(0.05)

Table 4.4: Summary statistics of the distribution of the average target depth of PCG nodes from the UF-3-SAT instances. For every test set and level we show the mean and coefficient of variation of the distribution (over all instances) of the maximum target depth of the open plateau nodes (over all open plateaus at the given level).

maximum, respectively. Because the values are so low, we can expect that SLS algorithms typically progress by going down in one or two levels only, rather than jumping down multiple levels. Both the average and maximum target depths decrease with the problem size, which was unexpected but is likely related to the similar trend in out-degree discussed in the previous section. Finally, we note that the variance in the distributions is very low and tends to decrease with both the level (with the exception of level 1, for which the coefficient of variation is 0) and the problem size.

The corresponding data for the structured instances is shown in Tables 4.5 and 4.6. Surprisingly, the data is quite quantitatively similar to the data for the random test sets. The target depths increase with level, and are fairly similar between the structured instances. One notable difference occurs at level 2. The `anom`, `med`, and `par8-1-c` instances have a large number of level 2 plateaus with a target depth of 2 — indicating that it is fairly easy to reach solutions from level 2 in these instances. In contrast, the graph colouring and random test sets have relatively few connections between solutions and any states not at level 1.

	level									
	1		2		3		4		5	
Instance	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
anom	1.000	(0)	1.417	(0.32)	1.705	(0.32)	1.657	(0.32)	1.635	(0.35)
medium	1.000	(0)	1.758	(0.23)	1.644	(0.33)	1.530	(0.35)	1.557	(0.34)
flat20-easy	1.000	(0)	1.008	(0.07)	1.335	(0.16)	1.453	(0.19)	1.452	(0.19)
flat20-med	1.000	(0)	1.002	(0.03)	1.296	(0.18)	1.327	(0.17)	1.371	(0.17)
flat20-hard	1.000	(0)	1.004	(0.02)	1.266	(0.20)	1.335	(0.17)	1.379	(0.19)
par8-1-c	1.000	(0)	1.967	(0.06)	1.399	(0.14)	1.652	(0.27)	1.489	(0.29)

Table 4.5: Summary statistics of the distribution of the average target depth of PCG nodes from the structured instances. For every instance and level we show the mean and coefficient of variation of the distribution (over all open plateaus at the given level) of average target depths from the open plateau nodes.

	level									
	1		2		3		4		5	
Instance	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$	$\bar{x}$	$\sigma_x/\bar{x}$
anom	1.000	(0)	1.500	(0.33)	2.036	(0.35)	2.199	(0.39)	2.258	(0.43)
medium	1.000	(0)	1.788	(0.23)	1.989	(0.40)	1.848	(0.47)	2.002	(0.44)
flat20-easy	1.000	(0)	1.012	(0.11)	1.765	(0.24)	2.114	(0.27)	2.313	(0.28)
flat20-med	1.000	(0)	1.003	(0.05)	1.639	(0.29)	1.896	(0.28)	2.021	(0.28)
flat20-hard	1.000	(0)	1.026	(0.16)	1.534	(0.33)	1.886	(0.27)	2.082	(0.31)
par8-1-c	1.000	(0)	2.000	(0)	1.889	(0.24)	1.729	(0.28)	1.659	(0.32)

Table 4.6: Summary statistics of the distribution of the maximum target depth of PCG nodes from the structured instances. For every instance and level we show the mean and coefficient of variation of the distribution (over all open plateaus at the given level) of maximum target depths from the open plateau nodes.

4.3 Solution Reachability

Now that we have defined PCGs and studied some of their basic graph-theoretic properties, we begin to treat PCGs as actual models of the search space and demonstrate PCGs can be used to model and understand SLS algorithm behaviour. The intuitive idea is that actual SLS algorithm trajectories will correspond to paths in the PCGs, under various simplifying assumptions. First, we are unable to model non-improving moves using PCGs. However, it is well-known that the performance of many SLS algorithms (including algorithms that do and do not make non-improving moves) is highly correlated on the test sets studied here (see, for example, [Hoo98]). Thus, the intrinsic hardness of an instance is relatively algorithm-independent, so accurately modelling the behaviour of the simpler algorithms is desirable because it abstracts away the subtleties introduced by more complex algorithms. Second, the time spent searching plateaus is not captured by PCGs, and we instead assume that an algorithm is able to immediately choose randomly and independently from the set of exit-target pairs from a given plateau. While this assumption may be violated by phenomenon such as the clustering of exits within plateaus (which has been observed and discussed briefly in Chapter 3), and by the fact that, given an exit, most SLS algorithms are biased towards choosing

higher-quality targets, we believe that it is a reasonable first-order approximation of the behaviour of actual SLS algorithms.

Formally, given a weighted PCG $G = (V, E, w)$ we model algorithm behaviour by defining a Markov chain $\mathcal{M} = \{X_0, X_1, X_2, \cdots\}$, where each X_i is a random variable with domain V. The transition matrix, P, is derived from the PCG edge weights: $P_{uv} = \overline{w}(u, v)$ wherever $\overline{w}(u, v)$ is defined, $P_{uu} = 1$ for all $u \in L$, and all other entries are zero. The self loops for all of the closed plateaus make those Markov chain states into "sinks", ensuring that all of the probability mass is eventually accumulated into those states. This transition matrix is of a special form: because it contains no cycles, the maximum number of transitions that can be made before encountering one of the "sink" states is bounded above by the maximum path length in the PCG, l (note that l is usually defined to be a constant for the partial PCGs that we study, but in all cases is bounded above by the number of propositional clauses). This immediately implies that, for $t > l$, we have:

$$p^{(t)} = p \cdot p^{(t-1)} = p^{(t-1)} = p^{(l)} \tag{4.10}$$

Now, given an initial distribution Δ over the Markov chain states, we can calculate the distribution of X_t, the state of the Markov chain at time t. By definition, this is just $p^{(t)} = \Delta \cdot P^{(t)}$. For all $t > l$, this becomes $\pi = p^{(t)} = \Delta \cdot P^{(l)}$. Thus, the Markov chain converges to steady state π in time polynomial in the size of the *PCG*. Obviously, the only states which will be assigned any probability mass in π will the those in L — *i.e.* the closed plateaus (including solutions).

$\mathcal{M}$ gives us a model with which we can estimate the probability that a greedy SLS algorithm we will end up in each of the closed plateau regions, given an arbitrary initial distribution over the starting states. In order to model the behaviour of actual SLS algorithms as accurately as possible, we let $\Delta(u)$ be directly proportional to the number of states in u — corresponding to randomly and uniformly choosing a starting state.

Given our model, we would like to evaluate how accurately it captures the interesting features of the search space, and how accurately it models the behaviour of actual local search algorithms. In order to quantify this, we define the "solution reachability" of an instance as the probability mass assigned by $\mathcal{M}$ to all solution plateaus in the steady state. Intuitively, we expect that the hardness of an instance should be (negatively) correlated with the probability that the Markov chain simulation ends up in a solution. The next definition demonstrates how we (efficiently) calculate the desired probability, and introduces our notation of "solution reachability".

Definition 4.4. (Solution Reachability)

Let $G = (V, E, w)$ be a weighted PCG, and Δ be a probability distribution over V. We define the

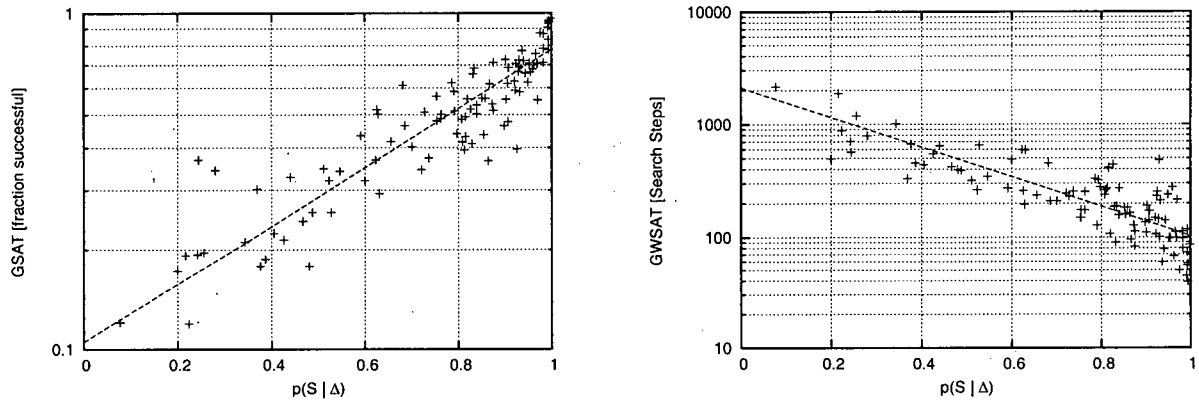

Figure 4.4: Correlation between $p(S \mid \Delta)$ and two measures of *lsc* for the *uf30* test set. Left: fraction of successful GSAT runs. Right: GWSAT *lsc* (measured in search steps).

conditional reachability $p(v \mid \Delta)$ of each node $v \in V$ recursively:

$$p(v \mid \Delta) := \begin{cases} \Delta(v) & \text{if } v \in R \\ \sum_u p(u \mid \Delta) \cdot w(u,v) + \Delta(v) & \text{otherwise} \end{cases} \tag{4.11}$$

Given the conditional reachability of each node, the *solution reachability* $p(S \mid \Delta)$ of the PCG is then simply defined as:

$$p(S \mid \Delta) := \sum_{s \in S} p(s \mid \Delta) \tag{4.12}$$

Note that the set S in the equation represents the set of solution plateaus in the PCG. □

Figure 4.4 demonstrates how effectively our model captures the search space features underlying instance hardness. In the left pane of the figure, we show the correlation between the solution reachability $p(S \mid \Delta)$ and the fraction of successful runs of the GSAT algorithm. The plot also shows the function $y = 0.105102 * 10^{0.870166 \cdot x}$, which is the best least squares fit through the points of an equation of the form $y = c \cdot 10^{b \cdot x}$ (*i.e.* a straight line in the semi-log plot). To measure the fraction of successful GSAT runs, we ran GSAT 1000 times (without restart) with a cutoff high enough to ensure that all runs that did not encounter a closed plateau successfully completed (a cutoff of 1000000 was used for all experiments, whereas all successful runs took less than 10000 steps). There is a surprisingly strong correlation between the logarithm of the success fraction and the solution reachability (correlation coefficient $r = 0.91$). While we were expecting the correlation, the exponential relationship between $p(S \mid \Delta)$ and the fraction of successful runs is surprising. Note, however, that the exponential function has a sum of squared residuals $ssr = 0.864$ (rms of residuals = 0.094, variance of residuals = 0.009), while the best fit linear function has $ssr = 1.138$ (rms of residuals = 0.108, variance of residuals = 0.012), so the exponential function more accurately models the relationship.

Test Set	$\overline{p(S \mid \Delta)}$	$\sigma_x/\bar{x}$	$\overline{lsc}$	$\sigma_x/\bar{x}$	r	t_s
uf20	0.73	(0.31)	124.60	(1.50)	-0.79	12.8
uf25	0.73	(0.33)	200.07	(0.75)	-0.85	16.2
uf30	0.75	(0.31)	309.92	(1.05)	-0.86	16.5
uf35	0.74	(0.34)	429.50	(0.95)	-0.85	16.0
uf40	0.75	(0.35)	628.30	(0.99)	-0.84	15.4
uf45	0.80	(0.28)	713.63	(1.09)	-0.65	8.5

Table 4.7: Correlation between $p(S \mid \Delta)$ and GWSAT *lsc* for the UF-3-SAT test sets. $\overline{p(S \mid \Delta)}$ and $\overline{lsc}$ are the mean solution reachability and local search cost for the entire test sets, respectively. The columns labelled $\sigma_x/\bar{x}$ are the coefficient of variation of the associated distributions. Finally, the columns labelled r and t_s show the correlation coefficient and test statistic of each regression analysis, respectively. Note that a test statistic $t_s > 2.262$ indicates a statistically significant correlation with confidence level $\alpha = 0.05$.

The right pane of Figure 4.4 shows the correlation between $p(S \mid \Delta)$ and the local search cost of a more robust variant of GSAT, GWSAT [SK93]. This plot also shows the best least squares fit equation of the form $y = 10^{a \cdot x + b}$, which in this case is the function $y = 10^{-1.3104 \cdot x + 3.32362}$. GWSAT is able to make upwards moves (which are not captured by our simple model), but this does not seem to have a significant negative effect on the predictive power of our model. We observe a strong negative correlation ($r = -0.86$) between the logarithm of GWSAT *lsc* and $p(S \mid \Delta)$. In this case, the exponential relationship is less surprising since given a fixed success probability in our model, it may still require an SLS algorithm an exponential number of steps to actually find a solution.

Table 4.7 shows the correlation coefficient of the regression analysis performed on each of the test sets, in order to study how the predictions of the model scale with problem size. The results demonstrate clearly that $p(S \mid \Delta)$ is highly (inversely) correlated with the local search cost of GWSAT for all of the UF-3-SAT test sets studied here, and the results seem to scale well. It is slightly troubling that the *uf45* test set had a relatively low correlation coefficient. Closer examination reveals that this is caused by a few outliers having high $p(S \mid \Delta)$ and *lsc* values. Further investigation indicates that in each of these outlying instances there was at least one open plateau region that had an uncharacteristically high exit distance — making the instance relatively hard, while not being factored into the calculation of $p(S \mid \Delta)$. Given our simplifying assumptions, it is not surprising that there exist cases for which the model gives poor predictions — overall, however the results are very positive. In order to get even more accurate predictions, the simplifying assumptions could be relaxed by including information in the model about, for example, the fraction of exits in plateaus and/or the exit distance, and incorporating this into the model by adding suitably defined self-loops for open plateaus representing the expected number of steps required to find an exit from the plateau. We also note that this correlation analysis was repeated with other local search algorithms such as WalkSAT, Novelty$^+$, and SAPS, with similar results.

Finally, Table 4.8 shows the solution reachability values for each of our structured instances. Although

Instance	$p(S \mid \Delta)$
anom	0.566
medium	0.552
flat20-easy	1.000
flat20-med	1.000
flat20-hard	0.969
par8-1-c	0.298

Table 4.8: $p(S \mid \Delta)$ values for the structured instances.

we do not have a distribution of instances available to study the correlation between $p(S \mid \Delta)$ and *lsc* for the structured instances, we do note that the values that we observe are negatively correlated with *lsc* between all of the instances. The graph colouring instances, with the very high solution reachability values, are also the easiest instances (GWSAT *lsc* of 50, 96, and 710 for the easy, median, and hard instance, respectively), and the solution reachability of the hardest graph colouring instance is lower than the others. The planning instances, which have mid-range solution reachability values, are also only moderately difficult instances (GWSAT *lsc* of 1096 and 2573 for `anom` and `medium`, respectively). Finally, the parity instance, which has the lowest solution reachability value is also by far the most difficult instance (GWSAT *lsc* 14 248). This indicates that our model is informative for the structured instances as well as the random ones.

4.4 Properties of Traps in the Search Space

In the previous section, we studied a novel method of modelling the search space of SAT instances and the connection between a feature of this model — namely the probability of reaching a solution plateau — and the hardness of an instance. In this section, we study the extent to which a single closed plateau can affect the behaviour of local search algorithms. Intuitively, if a closed plateau is both attractive and difficult for a local search algorithm to escape from, then we would expect that plateau to have a significant negative impact on the local search algorithm. Informally, we refer to such plateaus as traps. The remainder of this section is broken into two major subsections, one dealing with the attractivity of traps, and the second with their escape difficulty. We will show that there do exist closed plateaus that are exceptionally attractive for SLS algorithms, and that this attractivity is closely related to the difficulty of the UF-3-SAT instances. In fact, we will show that in the hardest instances, an accurate measure of the attractivity of the single most attractive trap is enough to explain the difficulty of the instance — indicating that these traps are the single most influential search space feature that makes SAT instances hard.

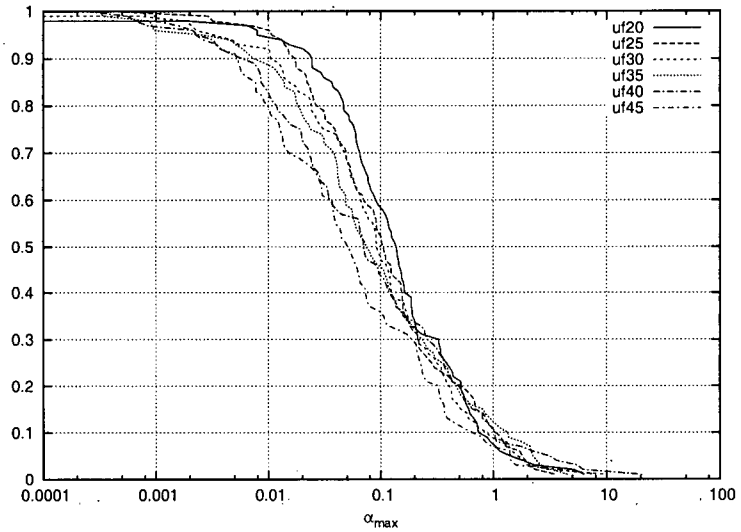

Figure 4.5: Distribution of α_{max} values for the UF-3-SAT test sets. The y value at every α value along the x axis of the graph gives the fraction of instances having $\alpha_{max} > \alpha$.

4.4.1 Attractivity

The previous section gave us the foundations for a definition of the attractivity of a trap. Intuitively, we define the attractivity of a trap as the reachability of the trap versus the reachability of all solutions. The following definition formalises this notion.

Definition 4.5. (Attractivity Factor)
Given a PCG $G = (V, E)$ and a closed plateau $u \in V$, the attractivity factor α of u is defined as:

$$\alpha(u) := \frac{p(u \mid \Delta)}{p(S \mid \Delta)} \tag{4.13}$$

This is simply the conditional reachability of the plateau normalised by the solution reachability, and gives an indication of how likely we expect the trap would be encountered compared to solutions. For brevity, we let P^{α}_{max} denote the plateau with the largest attractivity factor $\alpha_{max} := \max_{u \in M} \alpha(u)$. □

Figure 4.5 shows the distribution of α_{max} values for all of the UF-3-SAT test sets, and Table 4.9 shows the same data numerically for specific α thresholds. It is somewhat surprising how low the fraction of instances having $\alpha_{max} > 1$ is, ranging between 0.06 and 0.12. This means that less than 12% of instances had closed plateaus that were more attractive than solutions, in all of the test sets. Also interesting is that the fractions tend to decrease with problem size for low thresholds, but for α larger than 0.5 there does not seem to be a well-defined relationship between the various problem sizes. This indicates that as the problems get larger,

Test Set	$\alpha \geq 0.05$	$\alpha \geq 0.1$	$\alpha \geq 0.25$	$\alpha \geq 0.5$	$\alpha \geq 0.75$	$\alpha \geq 1$	$\alpha \geq 2$
uf20	0.78	0.58	0.30	0.21	0.09	0.07	0.03
uf25	0.68	0.52	0.26	0.20	0.14	0.10	0.02
uf30	0.69	0.47	0.28	0.15	0.10	0.08	0.03
uf35	0.58	0.45	0.28	0.18	0.15	0.12	0.07
uf40	0.56	0.43	0.30	0.19	0.14	0.10	0.06
uf45	0.50	0.35	0.21	0.11	0.09	0.06	0.02

Table 4.9: Statistics from the distribution of α_{max} values for the UF-3-SAT test sets. Every column gives the proportion of instances having α_{max} greater than the given value.

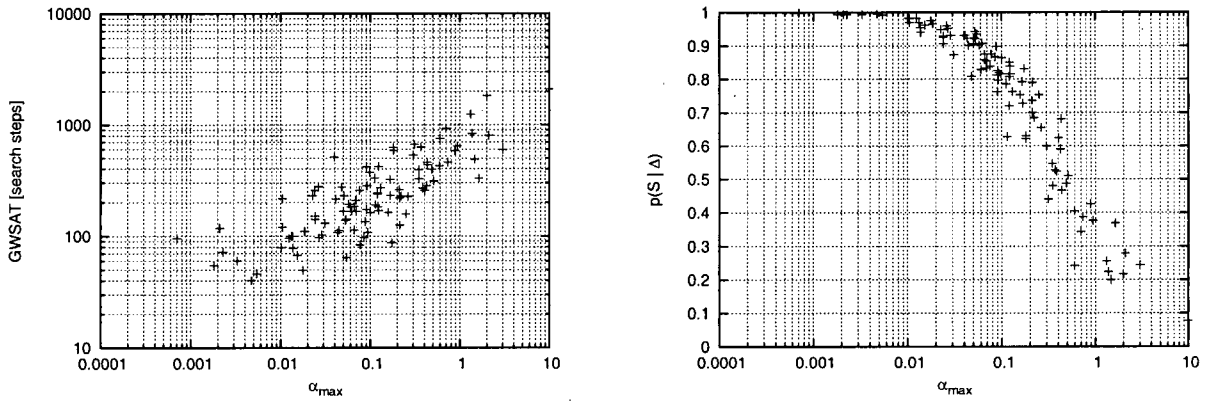

Figure 4.6: Correlation between α_{max} and *lsc* (left) as well as $p(S \mid \Delta)$ (right) for the *uf-30* test set.

fewer instances tend to have very attractive single plateau regions, suggesting that the hardness of larger instances might come either from multiple attractive closed plateaus, or from plateaus that are open, but still pose an obstacle for local search by having, for example, high exit-distances and/or few exits.

In order to understand the extent to which P^{α}_{max} is responsible for the hardness of UF-3-SAT instances, we examine the correlation between α_{max} and *lsc*. Figure 4.6 shows this correlation for the *uf30* test set (*c.f.* Figure 4.4, showing the correlation between $p(S \mid \Delta)$ and *lsc* for the same test set). The results are quite surprising — the correlation is very strong (correlation coefficient $r = 0.81$). This indicates that simply measuring the attractivity of the single most attractive trap gives a very good prediction of the hardness of the instance. Also note that the relationship between α_{max} and *lsc* is polynomial, as opposed to the exponential relationship between $p(S \mid \Delta)$ and *lsc*. Table 4.10 summarises the correlation data for all of the UF-3-SAT test sets. Very interestingly, the correlation coefficients that we observe are only slightly worse that those in Table 4.7, and the values scale well with the problem size.

Because α_{max} and $p(S \mid \Delta)$ are so closely related, and both are highly correlated with *lsc*, we must determine if the effects of one subsumes the other. Figure 4.6 (right) shows the correlation between α_{max} and $p(S \mid \Delta)$ on the *uf30* test set (similar correlations were observed for the other UF-3-SAT test sets). There is an obvious (inverse) correlation between the two measures, which should not be surprising considering their

Test Set	$\overline{\alpha_{max}}$	$\sigma_x/\bar{x}$	$\overline{lsc}$	$\sigma_x/\bar{x}$	r	t_s
uf20	0.39	(2.23)	128.91	(1.49)	0.81	13.7
uf25	0.33	(1.73)	201.42	(0.75)	0.81	13.6
uf30	0.38	(2.85)	310.39	(1.04)	0.81	13.4
uf35	0.39	(1.97)	431.26	(0.96)	0.83	14.9
uf40	0.60	(3.70)	628.29	(0.98)	0.89	19.1
uf45	0.30	(3.12)	712.61	(1.07)	0.80	13.1

Table 4.10: Correlation between α_{max} and GWSAT lsc for the UF-3-SAT test sets. $\overline{\alpha_{max}}$ and $\overline{lsc}$ are the α_{max} and local search cost for the entire test sets, respectively. The columns labelled $\sigma_x/\bar{x}$ are the coefficient of variation of the associated distributions. Finally, the columns labelled r and t_s show the correlation coefficient and test statistic of each regression analysis, respectively.

related definitions. What is surprising, however, is the tightness of the correlation. It seems that, for the UF-3-SAT test sets, the effects of P^{α}_{max} alone are enough to explain the variation is hardness between instances from the same test set. Thus, rather than there being many smaller closed plateaus causing the difficulty, there seems to be one major trap in each instance. This suggests that search strategies that explicitly detect and avoid such traps may be very effective.

These results are particularly interesting considering the results from Chapter 3. In particular, while we observe that the UF-3-SAT instances contain a significant number of closed plateaus at low levels, many of these plateaus do not contribute to the hardness of the instance. Furthermore, since the number of closed plateaus becomes vanishingly small as the level increases, it appears that the task of discovering P^{α}_{max} should be feasible.

4.4.2 Escape Difficulty

As we alluded to earlier in this section, we would like to characterise a "trap" as a closed plateau that is both attractive and difficult to escape from. To some extent, the attractivity factor of a closed plateau captures both attractivity and escape difficulty. However, we now attempt to explicitly characterise the escape difficult of closed plateaus. We characterise the escape difficulty of a closed plateau by measuring the shift in the RLD that occurs when a local search algorithm is initialised uniformly and randomly from states within the closed plateau.

Definition 4.6. (Escape Difficulty)
Given a PCG $G = (V, E)$ and a closed plateau $u \in V$, and any local search algorithm $\mathcal{A}$, the escape difficulty ϵ of u is defined as:

$$\epsilon(u) := \frac{lsc(u)}{lsc} \tag{4.14}$$

Where lsc is simply the mean of the RLD obtained by running $\mathcal{A}$ on the instance, and $lsc(u)$ is

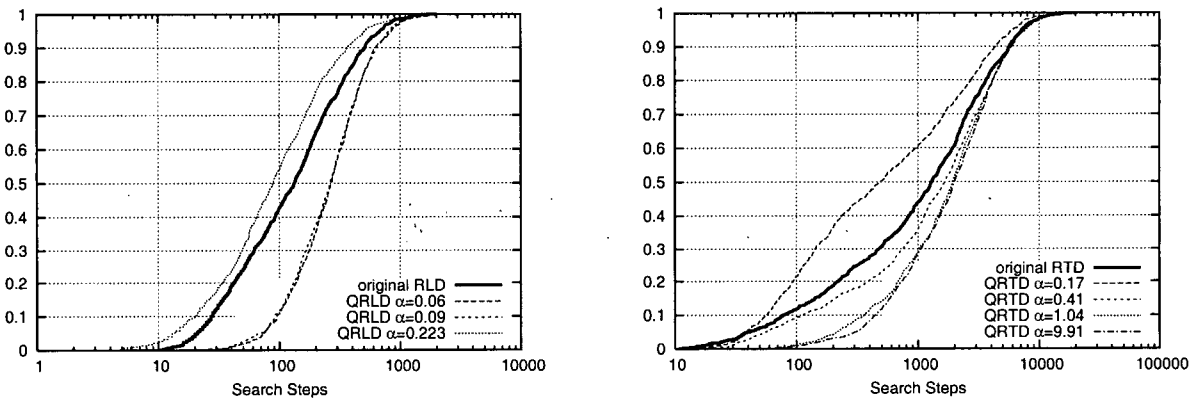

Figure 4.7: Shifted RLDs obtained by initialising GWSAT from various closed plateau regions. Shown are the instance with median and maximum local search cost (left and right, respectively) from the *uf30* test set. The RLDs are labelled by the α values of the closed plateaus from which GWSAT is initialised. See text for further details.

the mean of the the qualified RLD obtained by first randomly and uniformly initialising $\mathcal{A}$ from states in u, and then running $\mathcal{A}$ as usual (without restart). For brevity, we define $\epsilon_{max} := \epsilon(P^{\alpha}_{max})$, *i.e.* the escape difficulty of the most attractive plateau. □

Figure 4.7 shows example shifted RLDs for the instances with median and maximum local search cost for the GWSAT algorithm from the *uf30* test set. In this figure, we initialised GWSAT from each of the closed plateaus with highest α values (*i.e.* the most attractive plateaus). We first turn our attention to the left pane of the figure, which shows the shifted RLDs for the instance with median search cost. We show shifted RLDs for the three most attractive closed plateaus. Most surprising in this figure is that initialising from the most attractive plateau ($\alpha = 0.223$) actually makes the instance slightly easier for GWSAT. While surprising, this is likely an indication of why this instance is not more difficult; though P^{α}_{max} is quite attractive, being situated in the plateau is not detrimental to the search trajectory. The two other plateaus ($\alpha = 0.06$ and 0.09) result in a shift to the right in the RLDs, indicating that starting from these plateaus is detrimental to the search. However, in these cases, the plateaus are not very attractive so the instance is still easy to solve.

The right pane of Figure 4.7 shows shifted RLDs for the hardest instance from the *uf30* test set. This instance has fewer surprises. Initialising from the most attractive plateaus ($\alpha = 9.91$, 1.04, and 0.41) results in a shift to the right, and in fact ϵ monotonically increases with α for each of these plateaus. We also observe one plateau for which the initialisation makes the instance easier, but this plateau is not very attractive. Given these observations, it should be evident why this instance is difficult — the search space contains closed plateaus which are both very attractive and difficult to escape from.

Figure 4.8 shows the distribution of ϵ_{max} values for all of the UF-3-SAT test sets, and Table 4.11 shows summary statistics from the same distributions.

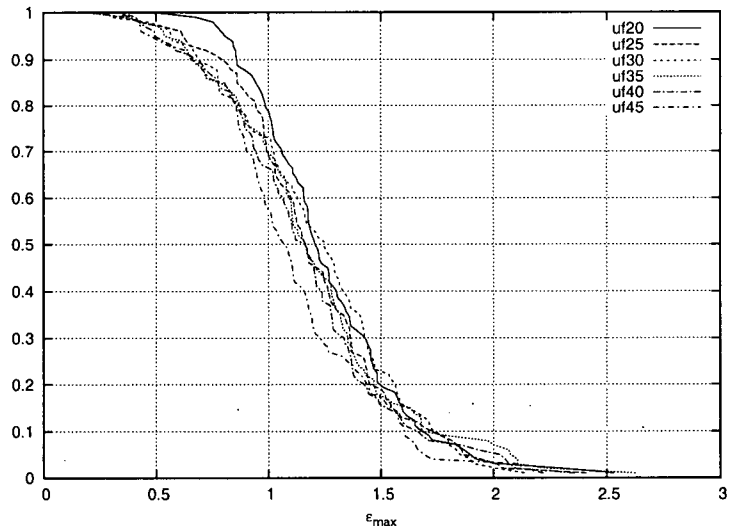

Figure 4.8: Distribution of ϵ_{max} values for the UF-3-SAT test sets. The y value at every ϵ value along the x axis of the graph gives the fraction of instances having $\epsilon_{max} > \epsilon$.

Test Set	$\bar{x}$	$\sigma_x/\bar{x}$	min	med	max	$\epsilon_{max} \geq 1$
uf20	1.259	(0.27)	0.489	1.201	2.529	0.786
uf25	1.199	(0.31)	0.343	1.155	2.410	0.690
uf30	1.212	(0.32)	0.273	1.221	2.136	0.727
uf35	1.193	(0.36)	0.187	1.154	2.625	0.690
uf40	1.154	(0.34)	0.328	1.120	2.357	0.660
uf45	1.101	(0.34)	0.195	1.054	2.220	0.570

Table 4.11: Summary statistics for the distribution of ϵ_{max} values for the UF-3-SAT test sets. $\bar{x}$, $\sigma_x/\bar{x}$, min, med, and max are the mean, coefficient of variation, min, median, and maximum of the distribution, respectively. The column labelled $\epsilon_{max} > 1$ shows the fraction of instances from each test set for which initialising from P^{α}_{max} increases the search cost.

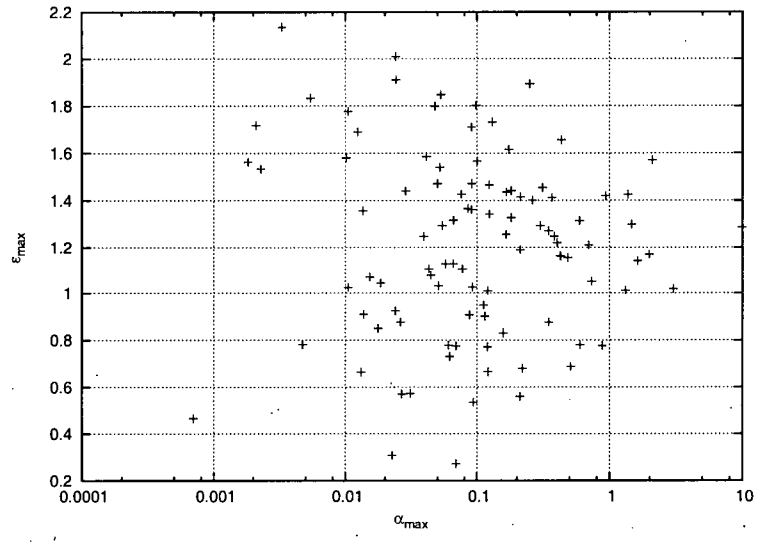

Figure 4.9: Correlation between α_{max} and ϵ_{max} for the *uf30* test set. Each point represents a single instance.

In general, we observe that α_{max} and ϵ_{max} are not correlated, as illustrated for the *uf30* test set in Figure 4.9. The correlation coefficient in this case is $r = 0.06$. This is not terribly surprising, and illustrates that, while ϵ_{max} is an interesting and useful search space feature, it is not directly related to *lsc*. For example, consider an instance that has a high *lsc* caused by a very attractive trap. In this case, it is likely that ϵ_{max} will actually be small because the SLS algorithm is drawn rapidly into the trap — thus, the difference between the original RLD and the RLD when initialised from in the trap is small. This intuition is confirmed by Figure 4.9, where all instances with $\alpha_{max} > 1$ have only moderate ϵ_{max} (though interestingly, ϵ_{max} is greater than 1 in all of these cases). In contrast, we have seen cases in which closed plateaus had a low α value, but if the search was initialised in the plateau, the local search cost increased dramatically (*c.f.* Figure 4.7). Again, this intuition is supported by Figure 4.9, which illustrates many cases in which an instance has a low α_{max} value, but high ϵ_{max}.

4.4.3 Connection With Mixture Models

In [Hoo02], Hoos introduces a "mixture model" for the behaviour of SLS algorithms for SAT. He observes that, while the RLDs of SLS algorithms on SAT instances are typically exponentially distributed, the RLDs of hard instances may exhibit substantial deviations from exponential distributions. Hoos demonstrates that the RLDs of these irregular instances can typically be fit by two component mixed exponential distributions, *i.e.* functions of the form

$$med[p, m_1, m_2] = p \cdot ed[m_1] + (1 - p) \cdot ed[m_2] \tag{4.15}$$

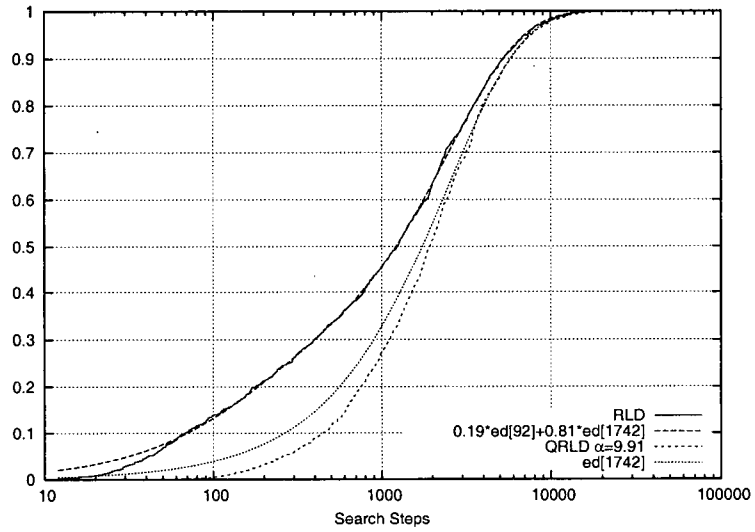

Figure 4.10: RLD for hard instance from the *uf30* test set, and approximation by mixed exponential distribution

where $ed[m](x) = 1 - 2^{-x/m}$ is the cdf of an exponential distribution with median m. The intuitive explanation is that there are two competing factors contributing to the irregular RLD: the solution(s) and a trap.

Our search space models and the mixture models agree very well with this earlier result. As an example, we consider Figure 4.10 which shows the RLD of GWSAT on the hardest instance from the *uf30* test set. Also shown in this figure is the best fit *med* function $med[0.19, 92, 1742]$, which approximates the empirical RLD almost perfectly.[4]

We also show the qualified RLD when initialising GWSAT from states in P^{α}_{max}. The RLD is fairly significantly shifted ($\epsilon = 1.27$) and, not surprisingly, differs substantially from an exponential distribution. However, $ed[1742]$ matches the right tail of the shifted RLD well. Interestingly, $ed[1742]$ also matches the right tail of the original RLD well (the tails of the original RLD and the shifted RLD coincide). Given our hypotheses about the effect of traps (such as P^{α}_{max} in this instance) on run-time behaviour, it should not be surprising that the right tails of these distributions coincide; it the trap is very attractive and difficult to escape from, then the algorithm will likely fall into the trap quickly and the long term behaviour of the trajectory will not be substantially different than if the algorithm had been initialised into the trap. However, because the original RLD is located substantially to the left of the shifted RLD, this indicates that there is at least some chance of reaching a solution without encountering the trap. This observation is supported by the solution reachability of this instance, which is 0.08.

[4]Note that there is a slight deviation in the left tail. This is caused by the initial search phase after initialisation, where the algorithm is able to improve the evaluation function rapidly, but has little chance of finding a solution. For an in-depth explanation of this deviation, refer to [Hoo98].

This interpretation agrees well with the best-fit mixed exponential $med[0.19, 92, 1742]$.[5] The first component of this distribution, weighted 0.19, is an exponential distribution with a low median of only 92 search steps. This component represents the trajectories which are "lucky" and do not encounter any major traps, and thus find a solution quite quickly. The second component, with the higher weight of 0.81, is an exponential with a higher median of 1742 search steps. Most interestingly, this exponential distribution matches the right tails of both the original RLD and the RLD after initialising from the P^{α}_{max}. This component represents trajectories which encounter the trap.

4.5 Summary

In this chapter, we introduced a novel *a posteriori* method of modelling and studying the search space features underlying the behaviour of SLS algorithms. This model gives insights into the search space features responsible for the hardness of instances, and also gives new insights into existing results from the literature.

The central definition of this chapter extends the notion of plateaus, defined in the previous chapter, with information about the *connectivity* of the plateaus. By modelling the search space with a graph in which plateaus are nodes, and nodes are connected by an edge if there exists at least one exit-target pair between the nodes — called a plateau connectivity graph — we are able to construct a compact model of the search space, while retaining many interesting properties.

We then improve this model by introducing weighted PCGs, in which the edge weights correspond to the degree to which the corresponding plateaus are connected. With this additional information, weighted PCGs capture the relevant properties of the corresponding search spaces in a compact way that facilitates our later analyses.

Our first empirical analysis of the chapter is of basic graph theoretic properties of the PCGs. We measure the average out-degree of PCG nodes, and the average and maximum target depth of PCG nodes. This analysis gives insight into high-level search space features that affect, for example, the mobility of SLS algorithms.

We next introduced one of the major contributions of this thesis — the notion of solution reachability. We demonstrated how it is possible to construct a Markov chain over the weighted PCG which effectively simulates SLS algorithm behaviour on the corresponding instance. Next, we proved that the Markov chain reaches the steady state in time polynomial in the size of the instance, and that in the steady state all of the probability mass is assigned to closed plateaus (including solutions). Solution reachability was then defined as the probability mass over all solution plateaus in the steady state. We demonstrated that solu-

[5] This function was computed using the function fitting procedure in `gnuplot` which, given the function $med[p, m_1, m_2] = p \cdot ed[m_1] + (1-p) \cdot ed[m_2]$ and the empirical RLD, finds the parameters p, m_1, and m_2 that minimise the sum of squared errors.

tion reachability correlates very well with *lsc*, and thus our model effectively characterises SLS algorithm behaviour.

Following this, we introduced and studied the properties of traps in the search space. Intuitively, a trap is a closed plateau that is both attractive and difficult to escape from. We defined the attractivity of a closed plateau as the amount of probability mass assigned to the closed plateau relative to the probability mass assigned to all solution plateaus. Interestingly, we demonstrate that the attractivity of the most attractive trap correlates well with *lsc* as well as solution reachability. Given the results from the previous chapter (*i.e.* the small size of closed plateaus), this suggests that it may be possible and effective in practice to dynamically detect and avoid these attractive traps, potentially improving the performance of SLS algorithms.

Next, we defined the notion of the escape difficulty of a trap, which is essentially a measure of how much worse off an SLS algorithm is when initialised into the given trap. We find that a large proportion of the closed plateaus have an escape difficulty greater than one (indicating that it is detrimental to be initialised into these closed plateaus), and that the average escape difficulty is greater than one for all of the test sets considered. Finally, we note that the attractivity factor of the most attractive trap and the escape difficulty of the same trap are uncorrelated, and we give an explanation of why this is not surprising, and how it illustrates a potential weakness of our measure of escape difficulty.

Our final analysis of the chapter connects plateau connectivity with the "mixture models" defined by Hoos [Hoo02]. We provide a limited amount of empirical evidence demonstrating that mixture models are the result of highly attractive closed plateaus, that have a high escape difficulty. We demonstrate that P^{α}_{max} (*i.e.* the most attractive closed plateau) is responsible for the large component of the mixed exponential distribution, which is consistent with the motivation given by Hoos when he introduced these models.

Chapter 5

Performance Critical Variables and the Value of Information in SAT Solving

In this chapter, we introduce the notion of performance criticality of variables in solving satisfiable SAT instances. Intuitively, a variable is performance critical if and only if it has a truth assignment under which the run-time of a given SAT algorithm is substantially reduced. Understanding the properties of PCVs (and the connection between PCVs and the search space features studied in the previous chapters) is useful for understanding what makes some SAT instances harder than others. PCVs are tightly related to many of the concepts presented in earlier chapters; in particular, PCVs reflect asymmetry in the search space.

We present a bound on the amortisable cost of realising an oracle for performance critical variables (PCVs) and show that PCVs exist for a number of high-performance SAT algorithms and a range of SAT instances. PCVs are algorithm dependent, yet, particularly for structured instances, the PCV sets of different SAT solvers often overlap. We also provide results on the additivity of the effects of PCV assignments and on the correlation between the number of PCVs and the hardness of SAT instances. We also discuss the relationship between PCVs and the established concepts of backdoor and backbone variables. Most interestingly, we discuss the relationship between PCVs, plateau connectivity, and traps in the search space.

This chapter is structured as follows. First, we motivate the study and introduce the important concepts used throughout the chapter. Next, we give formal definitions for performance critical variables and variable assignments and discuss how these notions of performance criticality are related to the theoretical idea of oracles for variable and value selection in SAT and their practical approximation using heuristics. Following this, we present the results of our empirical studies of performance critical variables and variable assignments for high-performance stochastic local search and systematic search algorithms for SAT. Finally, we discus the relationship between PCVs and related concepts from previous chapters of this thesis as well as some existing concepts from the literature, and conclude by summarising our main results and potential extensions to this work.

5.1 Motivation

The work presented in this chapter is based on the intuitive premise that in typical satisfiable instances of SAT, not all variables are equal with respect to their potential for reducing the run-time of a given SAT algorithm when assigned a particular truth value. This gives rise to the notion of *performance criticality;* intuitively, *performance critical variable assignments (PCVAs)* are truth assignments of individual variables that lead to substantial reductions of the run-time of a given SAT algorithm; and *performance critical variables (PCVs)* are the variables for which such assignments exist. The magnitude of the run-time reductions caused by a PCVA or PCV is measured by a suitably defined *criticality factor*.

Note that the notion of a performance critical variable differs substantially from the well-known concept of a backbone variable [Par97]; in particular, as we will see later, PCVs exist for instances that have no backbones and backbone variables are not necessarily performance-critical. The ideas behind PCVs are closely related to that underlying the recently introduced notion of a minimal backdoor set but differ in several important aspects. In particular, while backdoor sets capture a potentially drastic reduction in the asymptotic time complexity for solving a given family of SAT instances based on the knowledge of a set of variables, PCVs offer a more fine-grained measure of run-time reductions for individual instances that are achievable based on knowledge about individual variables.

PCVs are closely related to the search space features that we have studied in previous chapters. Understanding how search space features are related to PCVs facilitates a deeper understanding of the reasons underlying the performance criticality of variables. In particular, as we will demonstrate later in the chapter, there is a strong correlation between the average criticality factor of variables and both solution reachability and α_{max} (*c.f.* Sections 4.3 and 4.4). Furthermore, we demonstrate the effect of critical variables on plateau connectivity, which illustrates how the search space changes when PCVs are instantiated.

PCVs also have interesting practical applications: The existence of PCVs indicates a potential for reducing the run-time of a given SAT algorithm. This raises two fundamental questions. Do PCVs exist in commonly studied types of SAT instances? And how much is it worth to identify them, in terms of the amortisable cost of a procedure that determines the performance criticality of a given variable? In this chapter, we provide answers to both of these questions.

Regarding the first question, we provide a number of positive results. We found that for six high-performance SAT algorithms and a broad range of SAT instances we studied, there exist sets of PCVs, which in many cases contain substantially more than 5% of the variables appearing in the respective instance. Furthermore, for random as well as for structured instances we found a positive correlation between the algorithm-specific hardness of the instance and the size of its PCV set — indicating that there is the most potential for exploiting PCVs in the hardest instances. We also found evidence that the run-time reduction

potential of individual PCVs is additive, *i.e.*, by considering sets of PCVs, often substantially larger run-time reductions can be achieved than by the respective individual PCVs.

Regarding the second question, we present a simple bound on the amortisable computational cost for obtaining PCV information, giving a sense of how much value we can assign to the information that a given variable is a PCV. Furthermore, we present preliminary empirical results suggesting that at least in certain cases, information on PCVs for SLS algorithms can be obtained in a computationally inexpensive way, by collecting simple statistics on the individual variable assignments encountered in local minima of the respective search spaces.

It may be noted that our notion of performance criticality is algorithm dependent. However, unlike the somewhat related notion of backdoor variable sets, the concept of performance criticality applies directly to any type of SAT algorithm. Furthermore, we have found that the PCV sets for the different algorithms applied to the same SAT instance typically overlap to some extent; this is particularly the case for structured instances and suggests that the notion of performance criticality captures (at least to some extent) algorithm independent properties of these instances.

5.2 Oracles and Performance Criticality

The idea underlying performance criticality is rather straightforward. Given a SAT instance in the form of a CNF formula $\mathcal{F}$, fixing the truth value assigned to one of the variables in $\mathcal{F}$ such that satisfiability is preserved should intuitively lead to a reduction in the run-time required by a SAT algorithm for solving $\mathcal{F}$. When measuring the size of the search space in terms of the complete set of truth assignments to unassigned variables in $\mathcal{F}$, assigning one variable leads to a reduction in search space size by a factor of two. Clearly, the same factor would be obtained for the reduction in run-time of an extremely simple (and inefficient) search algorithm, such as uniform random sampling.

However, the effect of such an atomic assignment on more powerful algorithms can be very different, for example, because the solution density of a satisfiable instance (or the potential for propagation effects) increases (or decreases) sharply as a result of the atomic assignment under consideration. Intuitively, we call an atomic assignment performance critical w.r.t. the given algorithm $\mathcal{A}$ if it reduces the run-time of $\mathcal{A}$ by more than a factor of two, *i.e.*, the reduction effect for $\mathcal{A}$ exceeds what could be expected due to the corresponding reduction in search space size. This is captured by the following definition:

Definition 5.1. (Performance-critical variable assignment (PCVA))
Given a propositional formula $\mathcal{F}$, let $Var(\mathcal{F})$ denote the set of variables occurring in $\mathcal{F}$. Let $\mathcal{A}$ be a SAT algorithm, and let $sc(\mathcal{F}, \mathcal{A})$ denote the search cost of $\mathcal{A}$ on $\mathcal{F}$, *i.e.*, the (expected)

run-time required by $\mathcal{A}$ for solving $\mathcal{F}$. Finally, let $\mathcal{F}[x := v]$ denote the formula obtained from $\mathcal{F}$ by setting x to truth value v. An atomic truth assignment $x := v$, where $x \in Var(\mathcal{F})$ and $v \in \{\bot, \top\}$, is called a *performance critical variable assignment (PVCA)* of $\mathcal{F}$ with respect to $\mathcal{A}$ if and only if:

$$sc(\mathcal{F}[x := v], \mathcal{A}) < sc(\mathcal{F}, \mathcal{A})/2 \tag{5.1}$$

In this context, the *criticality factor of* $x := v$ is defined as:

$$cf(x := v, \mathcal{F}, \mathcal{A}) := sc(\mathcal{F}, \mathcal{A})/(2 \cdot sc(\mathcal{F}[x := v], \mathcal{A})) \tag{5.2}$$

and PCVAs are characterised by a criticality factor > 1. □

Note that the definition of PCVAs generalises easily to multiple variable assignments, by ensuring that the search cost is decreased by a factor of 2^k, when there are k variable assignments being made. We investigate the behaviour of PCVAs in this general sense later in the chapter when we study the additivity of PCVAs. The notion of performance criticality can also be easily generalised to variables rather than variable assignments. The intuition here is to run $\mathcal{A}$ on $\mathcal{F}[x := \bot]$ and $\mathcal{F}[x := \top]$ in parallel until a solution is found. A variable is called performance critical if and only if the combined run-time of these two parallel runs is smaller than the run-time of $\mathcal{A}$ on $\mathcal{F}$, as formalised by the following definition.

Definition 5.2. (Performance-critical variable (PCV))
A variable $x \in Var(\mathcal{F})$ is a *performance critical variable (PCV)* of $\mathcal{F}$ with respect to $\mathcal{A}$ if and only if:

$$\min\{sc(\mathcal{F}[x := \top], \mathcal{A}), sc(\mathcal{F}[x := \bot], \mathcal{A})\} < sc(\mathcal{F}, \mathcal{A})/2 \tag{5.3}$$

In this context, the *criticality factor of* x is defined as:

$$cf(x, \mathcal{F}, \mathcal{A}) := sc(\mathcal{F}, \mathcal{A})/(2 \cdot \min\{sc(\mathcal{F}[x := \top], \mathcal{A}), sc(\mathcal{F}[x := \bot], \mathcal{A})\}) \tag{5.4}$$

and PCVs are characterised by a criticality factor > 1. □

From these definitions it follows immediately that if $x := v$ is a PCVA of $\mathcal{F}$ with respect to $\mathcal{A}$, then x is a PCV of $\mathcal{F}$ w.r.t. $\mathcal{A}$, and conversely, if x is a PCV of $\mathcal{F}$ w.r.t. $\mathcal{A}$, then either $x := \top$ or $x := \bot$ is a PCVA of $\mathcal{F}$ w.r.t. $\mathcal{A}$. Although in the latter case, it is possible that both $x := \top$ and $x := \bot$ are PCVAs of $\mathcal{F}$ w.r.t. $\mathcal{A}$, our empirical results (reported in the next two sections) indicate that this happens extremely rarely. Consequently, for a given SAT instance and algorithm there is a near one-to-one relationship between PCVAs and PCVs.

Both notions of performance criticality are conceptually motivated by the idea of perfect oracles (*i.e.*, oracles without error). In the context of this work, a perfect oracle correctly decides the truth of a predicate without incurring any cost in terms of resource consumption. A *perfect PCVA oracle* would decide for a given algorithm $\mathcal{A}$, formula $\mathcal{F}$, variable $x \in Var(\mathcal{F})$, and truth value $v \in \{\top, \bot\}$, whether $x := v$ is a PCVA of $\mathcal{F}$ w.r.t. $\mathcal{A}$. Likewise, a *perfect PCV oracle* would decide for a given algorithm $\mathcal{A}$, formula $\mathcal{F}$, and variable $x \in Var(\mathcal{F})$, whether x is a PCV of $\mathcal{F}$ w.r.t. $\mathcal{A}$. Considering the close relationship between PCVs and PCVAs, in the following we only consider PCV oracles.

In practice, the results obtained from an oracle may be approximated using heuristics. Typically, this has two consequences: the answers provided by the heuristic may be erroneous, and computing these answers consumes computational resources. This raises the following central question: Up to which point can the cost of a heuristic approximating a perfect PCV oracle be amortised through the reduction in solution cost that is achieved by using such an oracle, or in other words: what is the value of the information obtained by the oracle?

The answer to this question depends on two factors: (i) the average reduction in search cost that can be achieved by using a perfect PCV oracle, and (ii) the assumed quality of the heuristic approximation, *i.e.*, the error rate of the heuristic. The average search cost reduction can be easily determined from the criticality factors introduced above. In particular, for the error-free case, we obtain the following result:

Theorem 5.1. *Given a propositional formula $\mathcal{F}$ and a SAT algorithm $\mathcal{A}$, let n denote the number of variables in $\mathcal{F}$, and n_c the number of PCVs of $\mathcal{F}$ w.r.t. A; furthermore, let $\overline{cf}$ denote the average criticality factor of the PCVs of $\mathcal{F}$ with respect to $\mathcal{A}$. Then the amortisable expected run-time of an error-free heuristic for deciding the criticality of variables in $\mathcal{F}$ is bounded from above by:*

$$sc(\mathcal{F}, \mathcal{A}) \cdot n_c/n \cdot (1 - 1/\overline{cf}).$$

Proof: Let $\mathcal{H}$ be the heuristic procedure, $hc(x)$ be the cost of evaluating the heuristic for variable x, and $\overline{hc}$ be the average heuristic cost over all of the variables in $\mathcal{F}$. Assume that we use $\mathcal{H}$ to construct a new algorithm $\mathcal{A}'$ as follows: we consider the variables in random order, and for each variable x we execute $\mathcal{H}$ to determine if x is a PCV. If x is a PCV we run $\mathcal{A}$ on $\mathcal{F}[x := \top]$ and $\mathcal{F}[x := \bot]$ in parallel, otherwise we continue trying the variables in order. The search cost of this algorithm, $sc(F, \mathcal{A}')$ is just the expected heuristic cost ($\overline{hc}$) times the expected number of trials before choosing a PCV (n/n_c) plus the expected cost of solving the two simplified formulae in

parallel $(1/\overline{cf} \cdot sc(\mathcal{F}, \mathcal{A}))$. Obviously, the cost will be amortised if and only if:

$$\begin{aligned} & sc(\mathcal{F}, \mathcal{A}') && \leq && sc(\mathcal{F}, \mathcal{A}) \\ \Leftrightarrow \quad & (n/n_c) \cdot \overline{hc} + (1/\overline{cf}) \cdot sc(\mathcal{F}, \mathcal{A}) && \leq && sc(\mathcal{F}, \mathcal{A}) \\ \Leftrightarrow \quad & (n/n_c) \cdot \overline{hc} && \leq && sc(\mathcal{F}, \mathcal{A}) \cdot (1 - 1/\overline{cf}) \\ \Leftrightarrow \quad & \overline{hc} && \leq && sc(\mathcal{F}, \mathcal{A}) \cdot n_c/n \cdot (1 - 1/\overline{cf}) \end{aligned}$$

□

For example, if we assume that $n_c/n = 0.08$, and $\overline{cf} = 1.76$ then the heuristic cost $\overline{hc}$ is amortised if and only $\overline{hc} \leq 0.035 \cdot sc(\mathcal{F}, \mathcal{A})$. These values are actual data obtained from Novelty$^+$ on the bwa instance — thus, in practice, a perfect heuristic for deciding if a variable is a PCV could be allocated a substantial amount (3.5%) of the computational resources allocated to solving the problem and the cost would be amortised. In general, PCV oracles will likely be imperfect, *i.e.* have errors, but the cost can still be amortised in this case.

5.3 Performance Critical Variables for SLS Algorithms

In this section, we study PCVs with respect to various high-performance SLS algorithms for SAT, and discuss the relationship between PCVs and algorithm behaviour. We show that a nontrivial number of PCVs exist for many different types of SAT instances, and that the corresponding criticality factors are reasonably high, which demonstrates the potential for practically exploiting PCVs. The algorithms we study are WalkSAT [SKC93], Novelty$^+$ [Hoo99], and SAPS [HTH02]. We use the implementation of these algorithms from the SLS algorithm package UBCSAT [TH04]. WalkSAT and Novelty$^+$ have been widely studied in the literature, and are often used as benchmarks to compare against, while SAPS is relatively new but has been shown to be the best performing SLS algorithm on various types of SAT instances. As opposed to previous chapters of this thesis, where we focused on simpler SLS algorithms for the sake of abstracting away algorithm details, in this chapter we use state-of-the art algorithms.

The instances used in this chapter are described in Appendix A, and include larger instances than we were able to study in previous chapters. Again, we chose a variety of instance types and sizes, in order to study the effects of PCVs on many different instances. However, the complexity of the analysis, while less restrictive than in previous chapters, still limited the size of the instances we could study.

In order to measure the distribution of criticality factors over all PCVAs in a given SAT formula $\mathcal{F}$, we do the following. First, a complete SAT algorithm (*i.e.* an algorithm that is guaranteed to decide whether an instance is satisfiable in a finite amount of time) is used to test whether the (restricted) formula $\mathcal{F}[x := v]$ is

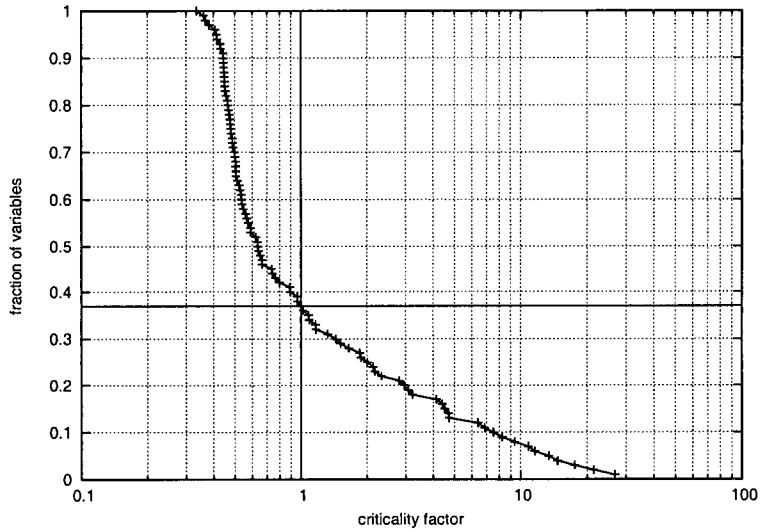

Figure 5.1: Distribution of criticality factors with respect to Novelty$^+$ for variables in the hardest *uf100* instance. Each point represents one variable, and criticality factors are measured as in Definition 5.2. Variables with a criticality factor greater than one (in this case 37% of all variables) are PCVs.

satisfiable for every atomic variable assignment $x := v$ ($2 \cdot n$ tests must be completed, where n is the number of variables in $\mathcal{F}$). Then, $sc(\mathcal{F}[x := v], \mathcal{A})$ is measured and recorded for every *satisfiable* formula $\mathcal{F}[x := v]$ and SLS algorithm $\mathcal{A}$. If $\mathcal{F}[x := v]$ is unsatisfiable, then its search cost is set to infinity. The criticality factor of each atomic assignment and variable are calculated from the search costs according to Definitions 5.1 and 5.2, respectively.

We evaluate two methods of computing $sc(\mathcal{F}[x := v], \mathcal{A})$. Both of these methods involve substituting the truth value v for x in the formula, then simplifying the formula before solving it using $\mathcal{A}$. The first method, *partial*, substitutes variable x with truth value v and simplifies the formula by removing all literals that become false and by removing all clauses that become true as a consequence of the substitution (this process is known as unit propagation). The second, *full*, does a *complete* unit propagation of the atomic assignment $x := v$ — that is, the assignment $x := v$ is unit propagated, but if any new unit clauses are generated as a result of a propagation, the corresponding atomic assignment is also propagated. Finally, as a baseline measurement, we also consider a third method, *init*, which simply initializes x to v at the start of the search, but does not further restrict the truth value of x. Since *init* did not result in any major reductions in search cost, we do not discuss results for this variant in the following. In all cases at least 100 runs were performed on each (simplified) formula, and the mean number of search steps required to solve the instance is reported as the search cost. Note that technically, the first two methods can be seen as preprocessing stages that are integrated into the respective SLS algorithms.

Figure 5.1 shows the distribution of criticality factors with respect to Novelty$^+$ for the hardest random-

Benchmark	Walksat-SKC sc	Walksat-SKC PCV frac / #	Walksat-SKC $\overline{cf}$ (c.v.)	Novelty$^+$ sc	Novelty$^+$ PCV frac / #	Novelty$^+$ $\overline{cf}$ (c.v.)	SAPS sc	SAPS PCV frac / #	SAPS $\overline{cf}$ (c.v.)
uf50	682	0.06 / 2.87	1.24 (0.17)	416	0.09 / 4.30	1.37 (0.27)	291	0.06 / 2.89	1.21 (0.11)
uf100	3643	0.04 / 4.20	1.26 (0.14)	3369	0.08 / 8.06	1.44 (0.47)	1358	0.03 / 2.71	1.16 (0.10)
uf200	27249	0.03 / 6.20	1.29 (0.16)	14529	0.04 / 8.38	1.28 (0.16)	9423	0.01 / 2.02	1.16 (0.08)
anomaly	587	0.29 / 14	42.91 (0)	177	0.42 / 20	10.04 (0)	137	0.40 / 19	8.29 (0)
medium	1182	0.17 / 20	149 (0)	446	0.16 / 18	62.98 (0)	325	0.19 / 22	37.97 (0)
huge	21786	0.07 / 31	1.66 (0)	12498	0.08 / 37	1.86 (0)	3427	0.05 / 25	1.51 (0)
bwa	18073	0.06 / 27	1.53 (0)	10362	0.08 / 36	1.76 (0)	3203	0.05 / 24	1.44 (0)
loga	139465	0.04 / 29	1.48 (0)	63276	0.05 / 43	1.38 (0)	11606	0.01 / 5	1.11 (0)
ii8	542	<0.01 / 1.5	2.24 (0.32)	8317	0.03 / 17.14	3.80 (0.25)	368	0.01 / 4.17	1.19 (0.07)
par8-1-c	9251	0.25 / 16	3.46 (0)	2421	0.25 / 16	2.36 (0)	1577	0.17 / 11	1.73 (0)
ssa7552-038	85319	0.06 / 83	4.90 (0)	–	– / –	– (–)	4539	0 / 0	n/a

Table 5.1: PCV results for thee prominent SLS algorithms. *sc* is the expected search cost for each algorithm on each benchmark set. PCV frac and # are the mean fraction of variables that are PCVs and number of variables that are PCVs, respectively. $\overline{cf}$ and *c.v.* are the mean and coefficient of variation, respectively, of the criticality factors of variables that are PCVs. Note that the test sets containing more than one instance are presented in italics. Entries marked '–' indicate that the PCV measurement had not completed after 150 000 CPU seconds.

3-SAT instance from the *uf100* test set (the distribution is over all variables in the instance). Note the large fraction (0.37) of variables that are PCVs. This shows that solving $\mathcal{F}[x := \top]$ and $\mathcal{F}[x := \bot]$ in parallel for any of those variables would result in a speedup. Even more interesting is the range of criticality factors. The smallest criticality factor is 0.33, meaning that solving $\mathcal{F}[x := \top]$ and $\mathcal{F}[x := \bot]$ in parallel would result in an approximately 3-fold slowdown, while some variables have criticality factors greater than 25, *i.e.*, lead to a 25-fold speedup. This result is very encouraging — because there are so many PCVs, it may be possible in practice to find such variables and realise the speedup. Also note that to take advantage of a PCV it is only necessary that the identity of the variable be known, not the correct truth value of the variable.

Table 5.1 shows the mean number and fraction of PCVs, as well as the mean criticality factors for all of the benchmark sets used in this study. The results show clearly that for all three algorithms, almost all of the instances studied here have a nontrivial number of PCVs. Generally, there seem to be a higher fraction of PCVs with respect to Novelty$^+$ than the other two algorithms, and the fraction for WalkSAT seems higher than SAPS in most cases. The average criticality factors are also reasonably high, indicating that a substantial speedup is realisable for most of the PCVs, particularly for the structured instances. The fraction of variables that are performance critical decreases with increasing problem size for both random and structured instances, though the average criticality factors stay relatively constant as the problem size is increased for the UF-3-SAT test sets. We also note that for all of the instances studied there exist PCVs with criticality factors much higher than the average — in many cases, finding the "maximally critical" PCV renders the instance trivial to solve.

In fact, the relationship between PCVs and algorithm behaviour is even more interesting than Table 5.1 alone would suggest. Figure 5.2 illustrates the correlation between search cost and the fraction of variables

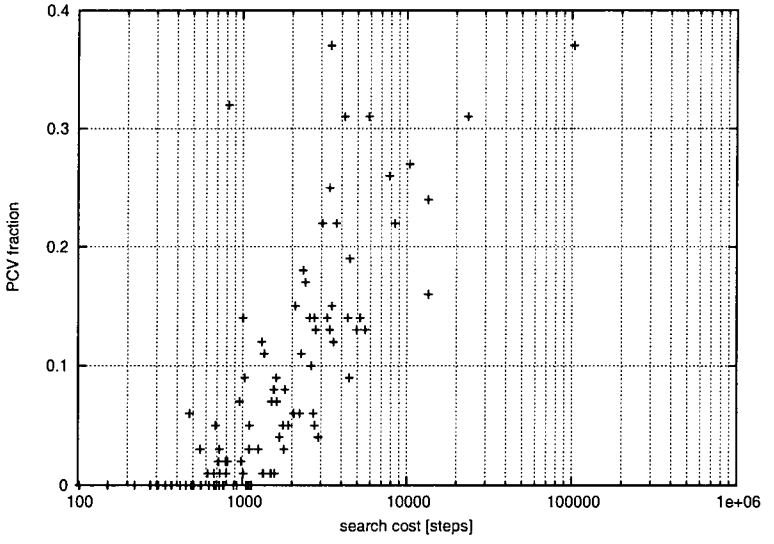

Figure 5.2: Correlation between search cost and PCV fraction for the *uf100* benchmark set using the Novelty$^+$ algorithm.

that are PCVs for all of the instances in the *uf100* benchmark set. Interestingly, we see that harder instances have more critical variables, which indicates that harder instances offer more potential for performance improvements than easier instances. We have confirmed that similar (though in some cases weaker) correlations exist for all of the algorithms studied in this paper (including the DPLL algorithms of the next section), on all of the random benchmark sets.

Interestingly, the criticality factors with respect to different SLS algorithms are quite highly correlated. Figure 5.3 shows the correlation of criticality factors with respect to Novelty$^+$ and SAPS for the hardest instance from the *uf100* test set, as well as for the bwa instance. Furthermore, the areas of the plot containing variables that are critical for both algorithms consistently contains a large fraction of the critical variables for each algorithm. In particular, if a variable is performance critical for Novelty$^+$, then it is almost always performance critical for SAPS. The correlations and overlap of critical variables is even stronger when comparing the two algorithms to WalkSAT. Thus, it appears that for at least some instances, some variables are performance critical independent of the algorithm. We give evidence in the next section that this is also true between different DPLL-type algorithms, as well as between SLS and DPLL-type algorithms.

Next, we study to which extent the run-time reductions achieved by correctly assigning PCVs are additive in the sense that combining multiple PCVAs leads to run-time reductions greater than those achieved by individual PCVAs. For this purpose, for a given formula $\mathcal{F}$, we generate a sequence of formulae $\mathcal{F} = \mathcal{F}_0, \cdots, \mathcal{F}_n$ such that for $1 \leq i \leq n$, $\mathcal{F}_i := \mathcal{F}_{i-1}[x := v]$, where the atomic assignment $x := v$ is chosen so that $sc(\mathcal{F}_{i-1}[x := v], \mathcal{A})$ is minimized. Intuitively, this gives us an optimistic estimate on the performance improvement afforded by combinations of PCVAs.

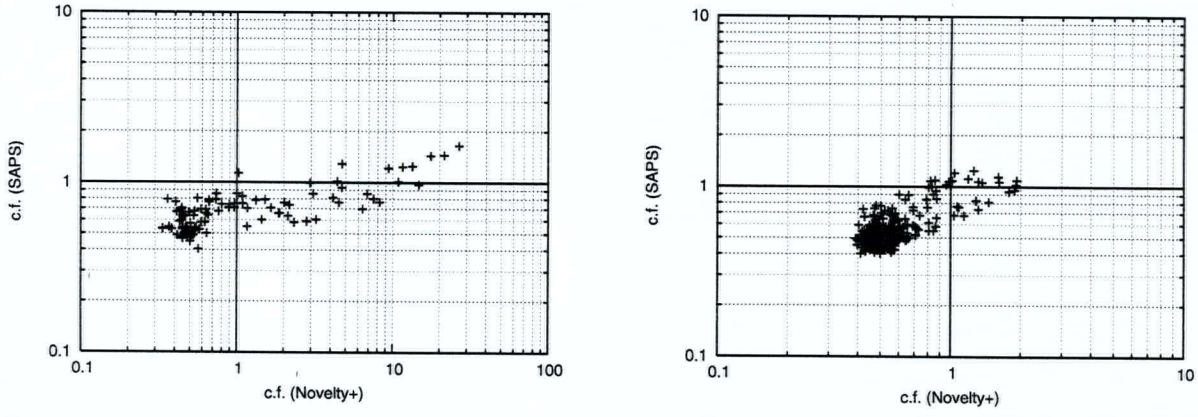

Figure 5.3: Correlation of criticality factors with respect to Novelty$^+$ and SAPS. Left: correlation on the hardest instance from the *uf100* test set. Right: correlation on the bwa instance. Each point represents one variable, and criticality factors are measured as in Definition 5.2.

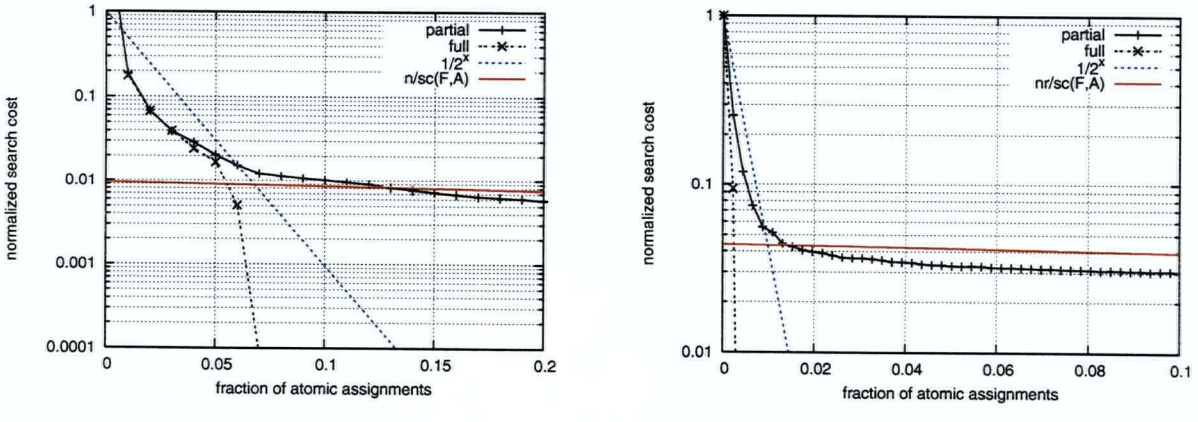

Figure 5.4: Additivity of PCVAs using dynamic ranking for Novelty$^+$. $nr/sc(\mathcal{F},\mathcal{A})$ is a normalized line on the plot indicating when the instance becomes trivial; see text for details. Left: Hard *uf100* instance; right: bwa instance from the *blocks* benchmark set.

Figure 5.4 shows the normalized search cost $sc(\mathcal{F}_i, \mathcal{A})/sc(\mathcal{F}, \mathcal{A})$ for each of the formulae $\mathcal{F}_0, \ldots, \mathcal{F}_n$. The results show the dramatic decreases in search cost obtained when multiple PCVAs with the largest criticality factors are used to simplify the formula. The line labelled $nr/sc(\mathcal{F}, \mathcal{A})$ in the plots indicates the point at which the search cost of the simplified instance becomes less than the number of variables in $\mathcal{F}_i$. In both of the examples shown, the instances become trivially solvable after only a relatively small fraction of the atomic assignments have been made. In fact, in both cases when using the *full* simplification technique, the unit propagation alone solves the instances after a typically relatively small number of PCVAs have been made. It may be noted that as a side-effect, this method for measuring the effects of PCVA additivity provides an effective way for finding small backdoor sets for the given instances (this will be discussed further in the 'Discussion' section of this chapter).

5.4 Performance Critical Variables for DPLL Algorithms

We now present results for PCVs for DPLL algorithms. These algorithms are complete (guaranteed to find a solution or determine that none exists in finite time), and have been shown to outperform SLS algorithms on many problem classes, particularly on SAT-encoded instances from domains such as planning and hardware verification. We test three DPLL-type algorithms: SATZ-rand [GSK98], KCNFS [DD01], and zchaff [MMZ+01]. These three algorithms have been well-studied in the literature, and are among the state-of-the-art complete algorithms on the benchmark sets we study in this paper. The heuristics employed in each algorithm are quite different; nevertheless, we demonstrate that there are nontrivial sets of PCVs with respect to each of the DPLL algorithms.

Our experimental protocol for finding PCVs with respect to these systematic SAT algorithms is similar to that for the previously discussed SLS algorithms. We iterate through every atomic assignment $x := v$, and measure $sc(\mathcal{F}[x := v])$ by first doing a full unit propagation of the assignment to obtain the simplified formula $\mathcal{F}[x := v]$ in which variable x is forced to take the value v, and then running each algorithm on $\mathcal{F}[x := v]$. Because SATZ-rand is a randomized algorithm, we run it 100 times on each formula, and report the mean search cost. The two other algorithms are deterministic, and consequently need to be run only once per instance. For all three algorithms we used measures of search cost that correlate linearly with run time (*e.g.* number of nodes in the search tree, or total number of implications). Hence, the search cost values reported in Table 5.2 can be compared for the same algorithm between different instances, but not between different algorithms.

DPLL-type algorithms typically run a full unit propagation as a preprocessing step; even if this were not the case, unit clauses are generally discovered and propagated very quickly during the search. For this reason, our results for *partial* (*i.e.* doing a partial rather than full unit propagation to obtain $\mathcal{F}[x := v]$) are

Benchmark	KCNFS sc	KCNFS PCV frac / #	KCNFS $\overline{cf}$ (c.v.)	SATZ-rand sc	SATZ-rand PCV frac / #	SATZ-rand $\overline{cf}$ (c.v.)	zchaff sc	zchaff PCV frac / #	zchaff $\overline{cf}$ (c.v.)
uf50	67	0.03 / 1.38	1.24 (0.12)	302	0.23 / 11.61	1.86 (0.46)	279	0.22 / 11.06	1.24 (0.12)
uf100	587	0.07 / 6.80	1.17 (0.07)	2090	0.06 / 6.26	1.21 (0.23)	3528	0.25 / 25.13	1.83 (0.36)
uf200	5186	0.22 / 44.8	1.76 (0.34)	42172	0.12 / 24.51	1.27 (0.08)	376187	0.24 / 47.18	2.66 (1.26)
anomaly	3	0.04 / 2	1.5 (0)	14	0.46 / 22	2.41 (0)	82	0.27 / 13	7.58 (0)
medium	12	0.21 / 24	2.88 (0)	142	0.26 / 30	18.77 (0)	318	0.22 / 26	32.08 (0)
huge	10	<0.01 / 2	1.25 (0)	1041	0.11 / 49	2.50 (0)	1805	0.05 / 21	1.60 (0)
bwa	10	0.01 / 6	1.32 (0)	958	0.10 / 45	2.46 (0)	2163	0.06 / 26	1.70 (0)
loga	–	– / –	– (–)	–	– / –	– (–)	45325	0.01 / 11	1.06 (0)
ii8	881419	0.07 / 4.78	1.51 (0)	4695	0.11 / 79.43	1.36 (0.24)	1944	0.11 / 62.29	1.92 (0.25)
par8-1-c	20	0.23 / 15	2.95 (0)	111	0.42 / 27	4.40 (0)	581	0.36 / 23	2.12 (0)
ssa7552-038	1628	0 / 0	n/a	440084	0.36 / 540	4.90 (0)	2677	<0.01 / 6	1.04 (0)

Table 5.2: PCV results for DPLL-type algorithms.*sc* is the expected search cost for each algorithm on each benchmark set. PCV frac and # are the mean fraction of variables that are PCVs and number of variables that are PCVs, respectively. $\overline{cf}$ and *c.v.* are the mean and coefficient of variation, respectively, of the criticality factors of variables that are PCVs. Entries marked '–' indicate that the PCV measurement had not completed after 150 000 CPU seconds.

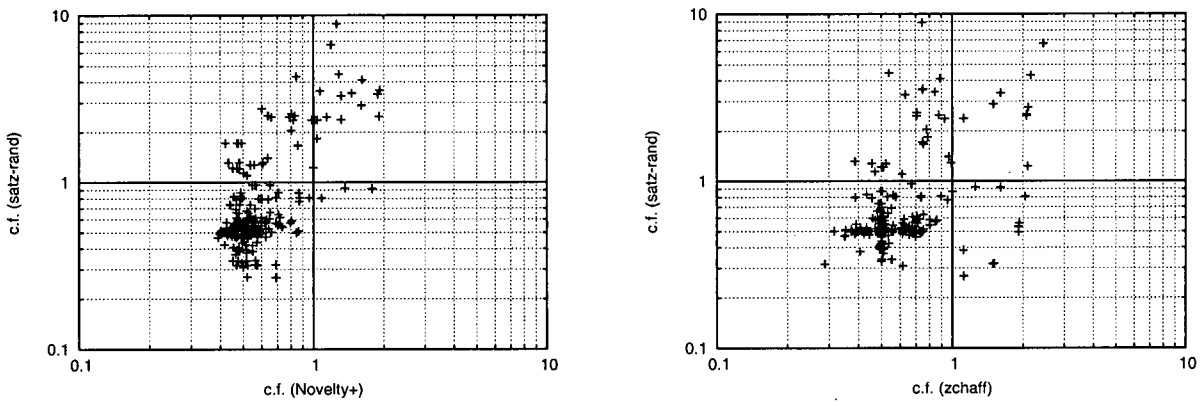

Figure 5.5: Correlation of criticality factors with respect to different algorithms on the bwa instance. Left: Novelty$^+$ vs. SATZ-rand, Right: zchaff vs. SATZ-rand. Each point represents one variable, and criticality factors are measured as in Definition 5.2.

virtually identical to those for *full*, and we report only the results for the latter.

Table 5.2 shows our PCV results for the DPLL-type algorithms. As for the previously considered SLS algorithms, that there are a nontrivial number of PCVs for virtually all of the benchmark instances considered. Interestingly, there often seem to be more PCVs for the DPLL-type algorithms than for the SLS algorithms when compared on a given test-set, most notably for the SATZ-rand and zchaff algorithms. The average criticality factors are also reasonably high for all of the test sets. As with the SLS algorithm results, most instances have PCVs with very high criticality factors with respect to the DPLL algorithms.

Additional results indicate that there is a significant amount of overlap between the sets of PCVs for SLS algorithms and DPLL-type algorithms, as well as between PCV sets for different DPLL-type algorithms. For example, consider Figure 5.5, which shows the correlation between the criticality factor of each variable in the bwa instance for Novelty$^+$ and SATZ-rand (left) and zchaff and SATZ-rand (right). We see that

the region for which the criticality factor is greater than one for both algorithms consistently contains a large portion of the points that are greater than one for each algorithm. This is quite surprising, given the different nature of these algorithms and their underlying heuristics, and provides evidence that algorithm-independent performance critical variables exist for some instances.

We also observe additivity effects for PCVAs with respect to the DPLL-type algorithms on many of the test sets that are similar to those for SLS algorithms. For example, zchaff on the `ssa7552-038` instance has only 6 PCVs (out of 1501 variables). However, when the single PCVA with the highest criticality factor is used to simplify the formula, the entire formula becomes trivial. As another example, KCNFS shows excellent additivity on the hardest *uf100* instance — the dynamic ranking found a small backdoor of only 6 PCVAs, which is surprisingly small considering the results from [Int03]. Furthermore, the search cost decreased by more than a factor of two with the addition of each PCVA, indicating that a speedup is possible even if all 2^6 assignments to the corresponding variables were run in parallel.

5.5 Discussion

As previously mentioned, the notion of PCVs is conceptually related to that of backdoor variables [WGS03]. Both concepts are algorithm-dependent; but while the notion of performance criticality applies to arbitrary SAT algorithms, backdoor sets are relative to polynomial-time SAT algorithms, such as unit propagation. Furthermore, it should be noted that, unlike our concept of performance criticality, the notion of a backdoor inherently refers to a property of a set of variables. It is also more coarse-grained, since it only captures potential reductions from exponential to polynomial-time asymptotic complexity for certain classes. It should be noted that variables from (minimal) backdoor sets are not necessarily performance critical, since a reduction may only be obtained when several or all of the elements of a given backdoor set are assigned the correct truth value. Additionally, it is currently unclear whether small backdoors can be determined and exploited in an amortised way.

Interestingly, our additivity for PCVAs resulted in new, substantially smaller backdoor sets than previously known for the random SAT instances considered here. For example, at clauses-to-variables ratio 4.3, Interian [Int03] reports an expected backdoor size of approximately $0.5 \cdot n$ for $n = 100$ variables. Using our dynamic ranking scheme and Novelty$^+$ on the *uf100* set, we find expected backdoor sets of size $0.078 \cdot n$, with minimum and maximum values of $0.05 \cdot n$ and $0.16 \cdot n$, respectively. We find similar results on this test-set when using the dynamic ranking scheme with all of the other algorithms.

Another concept that is intuitively related to PCVs is that of the backbone; the backbone of a satisfiable SAT instance is the set of variables that have the same truth values in all satisfying assignments. Obviously, any PCVA $x := v$ where x is a backbone variable x must set v to the correct value; hence, knowledge of

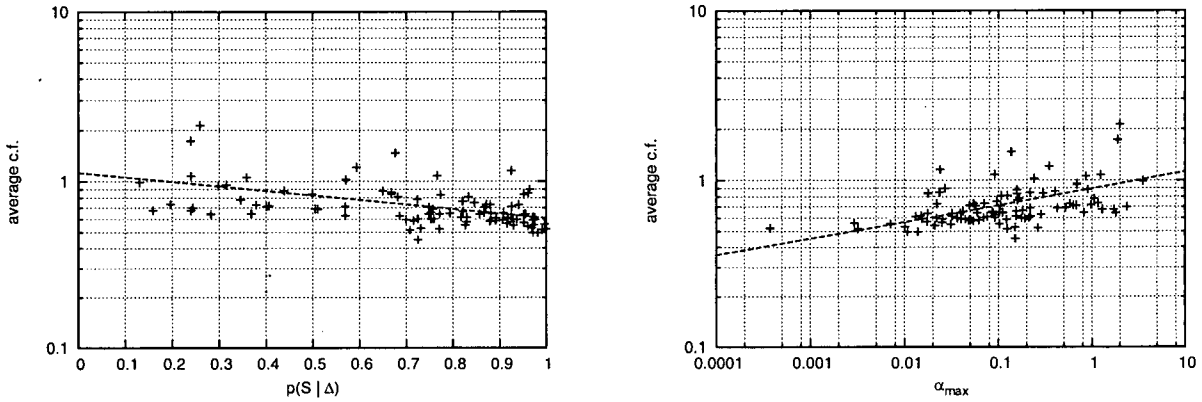

Figure 5.6: Correlation between $p(S \mid \Delta)$ and average criticality factor (left), and α_{max} and average criticality factor (right) with respect to Novelty$^+$ on the *uf25* test set. Each point represents one instance from the set, and the criticality factor is averaged over all variables in the instance.

backbone is potentially useful in narrowing down the set of candidates for PCVAs. However, backbone variables are not necessarily performance critical, and PCVs may not necessarily appear in the backbone of a given instance. In this context, it may be noted that some of our test-instances (*ii8**) have no backbones, but do have PCVs (*c.f.* Tables 5.1 and 5.2). Still, there may be a noisy correlation between the two concepts; considering recent approaches for the efficient heuristic determination of backbone variables [ZRL03, DD01], this issue should be further investigated.

5.5.1 Connection with Search Space Features

While we have demonstrated that performance critical variables exist, and that they have a substantial impact on the performance of SAT algorithms, so far we have not discussed the underlying causes of performance criticality. We now investigate the nature of the connection between the search space features studied in previous chapters and performance critical variables.

The left pane of Figure 5.6 shows the correlation between solution reachability $p(S \mid \Delta)$ (defined in Section 4.3), and the average criticality factor of each instance in the *uf25* test set. Very interestingly, instances with a high average criticality factor tend to have low solution reachability (the correlation coefficient is $r = -0.51$). This suggests that fixing critical variables to their correct values may increase the solution reachability, thus rendering the instances easier.

The right pane of Figure 5.6 shows a similar correlation between the maximum attractivity factor of closed plateaus α_{max}, and the average criticality factor of each instance. These two measures are positively correlated (correlation coefficient $r = 0.49$), suggesting further that fixing critical variables to their correct values may reduce the attractivity factor of the most attractive trap, thus rendering the instances easier.

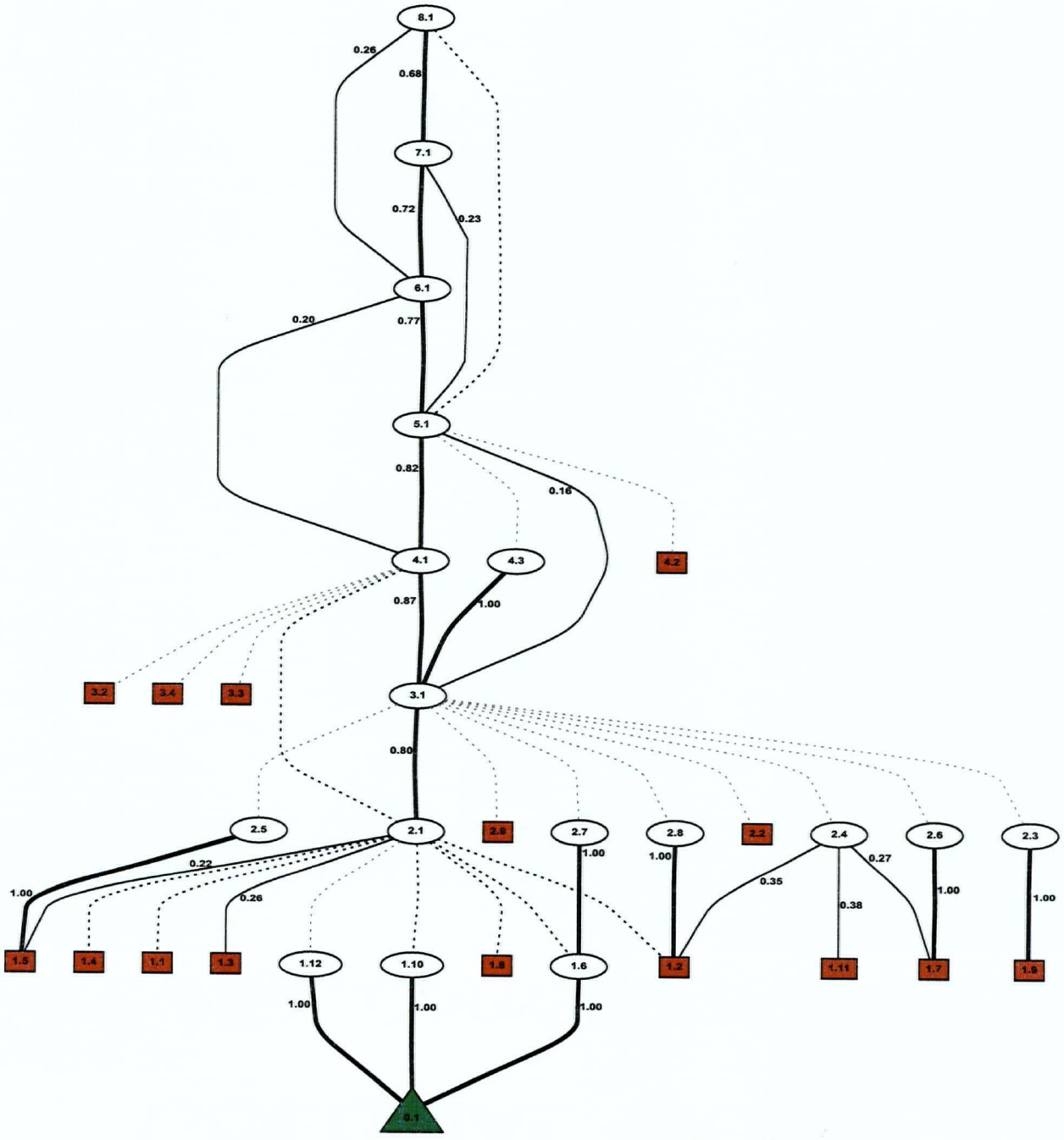

Figure 5.7: Partial, weighted PCG for the hardest *uf25* instance.

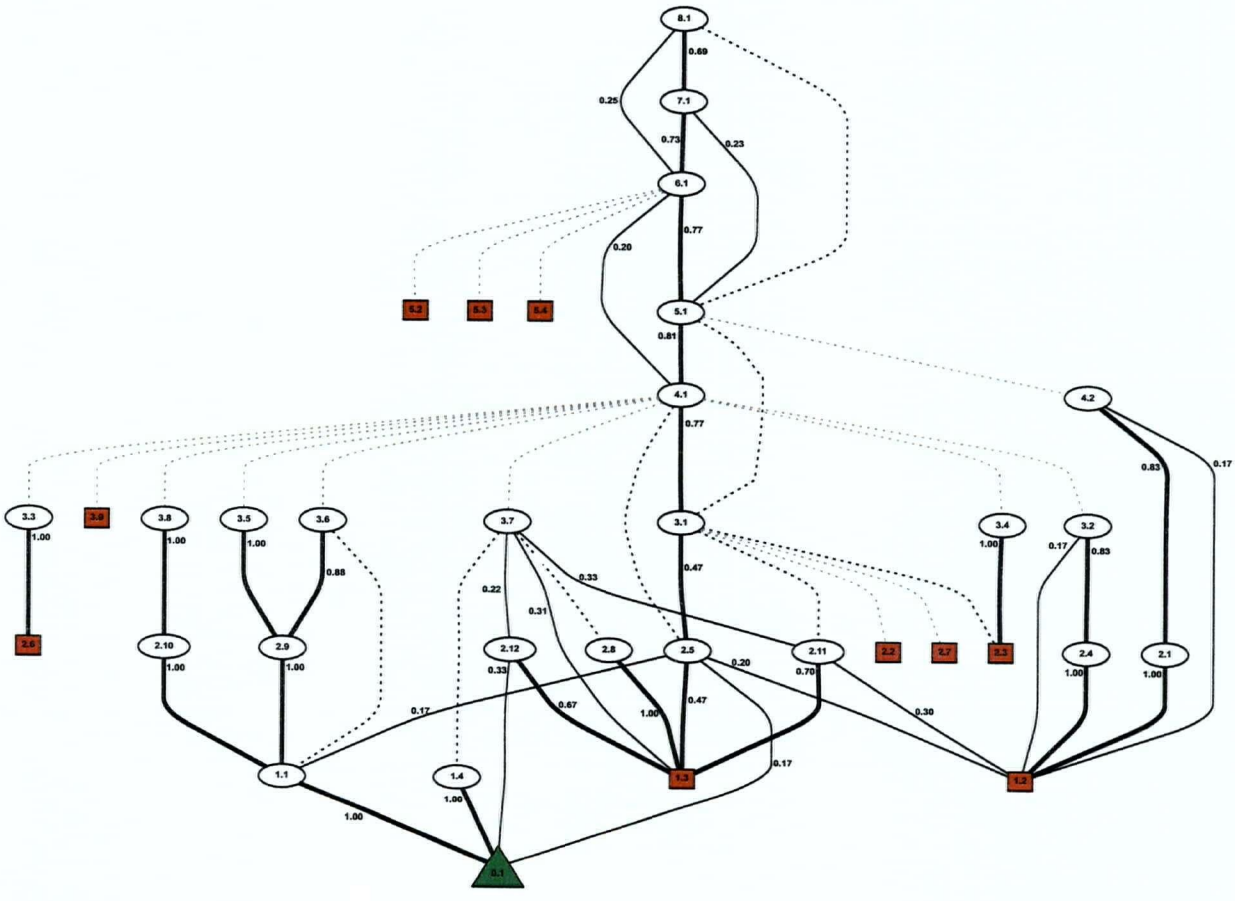

Figure 5.8: Effect on the PCG when instantiating the most critical variable in the hardest *uf25* instance.

To further test these conjectures, we closely examine the search space of the hardest *uf25* instance, both before and after the PCVA with the highest criticality factor is fixed (the most critical PCVA with respect to Novelty$^+$ has a criticality factor of 2.4). Figure 5.7 shows the weighted, partial plateau connectivity graphs for the original formula. It is obvious why this instance is hard. The majority of paths with significant weights lead down to the plateau labelled 2.1. From this point, most of the paths lead to closed plateaus, with only a few outgoing paths that can reach solutions. This hard instance has solution reachability $p(S \mid \Delta) = 0.15$.

After the most critical PCVA has been assigned, and the formula correspondingly simplified (all clauses containing the literal removed from the formula, and all occurrences of the negated literal removed from all clauses), the search space looks drastically different, as illustrated in Figure 5.8. Firstly, there are fewer plateaus at level 1, and more at level 3, and there are significantly fewer closed plateaus. The most interesting change, however, concerns the plateau connectivity. Most of the paths with significant weights still lead down to a level 2 plateau (labelled 2.5),[1] but from this point on the situation is drastically different. There are only four outgoing edges from plateau 2.5, and two of them lead to closed plateau regions. However, one edge (with weight 0.17) leads directly to the solution plateau and the last edge (also having weight 0.17) leads to a plateau that feeds directly into the solution plateau. Thus, fixing the most critical PCVA effectively made many of the traps much less attractive, and the solution more accessible. The solution reachability of the simplified instance is $p(S \mid \Delta) = 0.36$, which is now higher than 15% of the other instances in the test set. Furthermore, the simplified instance is now at the 60^{th} quantile of the hardness distribution for Novelty$^+$ (rather than the 100^{th}).

These results provide new insights into why variables are performance critical with respect to SLS algorithms. We conjecture that the most critical variable assignments are those appearing in very attractive traps, but having a value that is inconsistent with the set of solutions. Thus, the attractive trap effectively restricts the given variable to the (incorrect) value compatible with the traps. The next section discusses the possibility of exploiting performance critical variables based on this conjecture.

5.5.2 An Approximate PCV Oracle

Like in the case of backdoor sets and backbone variables, it is unclear whether there is a reasonably efficient way of determining PCVs or of guessing them with reasonable accuracy. Based on the intuition that in the case of SLS algorithms, the existence of PCVAs (and hence PCVs) could be the result of local minima attracting the search into non-solution areas of the respective search space, we performed initial experiments with a simple heuristic that counted the number of times each variable was set to $\top$ and $\bot$, respectively, in the local minima encountered during a small number of short independent runs of Novelty$^+$ on a given instance.

[1] Note that there is no correspondence between the node ids of the two PCGs.

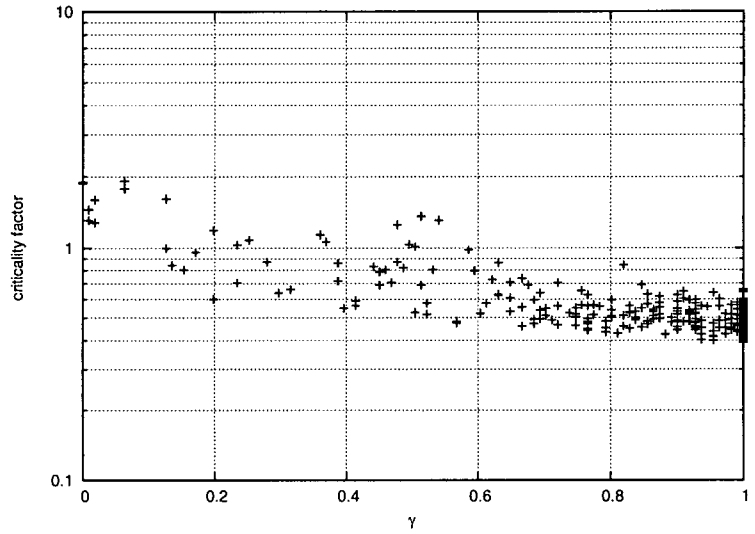

Figure 5.9: Correlation between γ and criticality factor for Novelty$^+$ on instance bwa . The γ values were collected by running Novelty$^+$ five times for 100 steps, and measuring the frequency of local minima in which each atomic assignment occurred. Each point represents one backbone-compatible atomic assignment.

As illustrated in Figure 5.9, for some instances we observe a negative correlation (correlation coefficient $r = -0.80$) between the criticality factor of atomic assignments $x := v$ compatible with the backbone and γ, the frequency of local minima in which x was assigned v. Thus, the PCVAs with highest criticality factors tended to have low frequencies — indicating that the corresponding variable was set to the opposite value in the majority of the local minima encountered during the short runs. The natural interpretation of this observation is that, while many variable assignments appear frequently in local minima, the critical ones are those that are frequently set to a value that is incompatible with any solutions. This analysis supports our conjecture from the last section, and further elucidates the relationship between search space features and algorithm behaviour.

Figure 5.9 suggests that it may be possible to heuristically determine PCVs in an efficient way — and to take advantage of them in practical algorithms. It may be noted that our method for determining γ values is similar the the method used in [ZRL03] for determining pseudo-backbone variables. However, contrary to their results, we find that setting these variables to their *opposite values* is more effective than setting them to the prevalent values in local minima. Unfortunately, it is beyond the scope of this thesis to provide an implementation and evaluation of a practical method for exploiting PCVs.

5.6 Summary

In this chapter we have introduced the notion of performance critical variables (PCVs). PCVs are related to previously studied concepts of backbone variables and minimal backdoor sets, but differ from these in interesting ways. Based on computational experiments with several high-performance SAT algorithms applied to a various sets of benchmark instances, we have demonstrated that almost all instances have a significant number of PCVs for both stochastic local search and systematic search algorithms. Particularly for structured instances, we observed a significant overlap between the PCV sets for different algorithms applied to the same problem instance as well as some correlation between the respective criticality factors. We have presented evidence for the additivity of the effects of PCVs, as well as for a positive correlation between size of PCV sets and the (algorithm-specific) hardness of instances.

We have also demonstrated a clear connection between performance critical variables and search space features. We have shown that the average criticality factor of an instance is correlated with $p(S \mid \Delta)$ and α_{max} (*c.f.* Sections 4.3 and 4.4, respectively), and we have illustrated, through the use of plateau connectivity graphs, how the search space is affected when the most critical PCVA is instantiated. Based on these observations, we conjecture that performance criticality is the result of attractive traps that force the critical variables to values that are inconsistent with all solutions.

These results raise a number of interesting questions. Possibly the most relevant of these concerns the existence of heuristics that approximate PCV oracles in an amortised way and hence can be used to improve existing SAT algorithms. We reported some preliminary encouraging results for SLS algorithms, but further work is required in order to obtain more conclusive answers. Overall, we believe that an understanding of the connection between performance criticality and search space features will give rise to further interesting results for SAT and other combinatorial problems and will ultimately lead to improvements in our ability to solve these problems.

Chapter 6

Conclusions and Future Work

In this thesis we have studied various search space properties, and empirically demonstrated that these search space features are responsible for making instances hard. We used the propositional satisfiability problem as our main application area because of its practical relevance, and conceptual simplicity — but we also note that most of the concepts and intuitions developed throughout the thesis generalise to other combinatorial problems. This chapter summarises the main contributions of the thesis, and provides interesting directions for future work.

Our contributions begin in Chapter 3, where we extend (and correct) results from the literature concerning the distribution of plateaus in the search space. We find plateaus that are significantly larger than those previously reported — an observation that clears up some apparently contradictory results in the literature. Next, we extend the analysis of plateaus by introducing a computationally expensive (but still feasible on problems of a nontrivial size) method based on binary decision diagrams for exhaustively exploring the plateau structure of SAT instances. This allows us to get a complete picture of the low-level regions of the search space, without resorting to sampling (as previous approaches were forced to do). We report a very interesting, novel result for the uniform random 3-SAT instances. We find that at high levels the vast majority of states at a given level all reside within a single, large, connected plateau, and that only at low levels does the search space really degenerate into more locally connected regions. We also report that closed plateaus are typically much smaller than open plateaus, and that closed plateaus become vanishingly rare as the level is increased, indicating that schemes which explicitly detect and avoid closed plateaus may be helpful.

We also extend our plateau analysis to include structured instances. This makes the results much more practically relevant, since structured instances are more interesting from a practitioner's point of view than randomly generated instances, and strengthens the contribution of this thesis since much of the existing literature focuses solely on randomly generated instances.

We perform a detailed analysis of the internal structure of plateaus, which has not been performed in the past. We measure plateau properties such as the branching factor of local minima states, the plateau diameter, the fraction of states in the plateau that are closed, and the average distance to an exit from the plateau.

Chapter 4 extends the results from Chapter 3 by studying the interconnectivity of plateaus in the search space. We demonstrate how we can abstract away irrelevant details of the search space and construct a simple, compact model of the search space that is still representive enough to capture the features responsible for instance hardness. We investigate simple graph-theoretic properties of this model, and discuss the consequences of these results on algorithm mobility. Next, we define a Markov process on the model, bound the mixing time of the Markov chain with a polynomial (in the size of the SAT instance), and demonstrate how this simplified model of the search space can give insight into the behaviour of SLS algorithms. Furthermore, we empirically demonstrate the effectiveness of this model by showing how certain features of the model correlate well with instance hardness.

We also provide a formal definition of "traps" in the search space, and demonstrate that, for uniform random 3-SAT instances, the effects of the single most attractive plateau if enough to account for the difficulty of the instance. We introduce a novel definition of the escape difficulty of traps, and demonstrate that in the vast majority of cases it is detrimental to initialise the search from the most attractive trap. Finally, we connect plateau connectivity with Hoos' "mixture models", and provide empirical evidence that it is attractive traps that are responsible for the hard component of the irregular RTDs.

The final chapter describing the empirical analyses of the thesis, Chapter 5, takes a slightly different direction that the previous chapters, and introduces the novel concept of performance criticality of variables in satisfiable SAT instances. After formally defining the notion of performance criticality, we provide a bound on the amortisable running time of a perfect PCV oracle. Our empirical analysis in this chapter demonstrates that there are a non-trivial number of PCV for all of the instances studied, indicating a widespread potential for practically exploiting PCVs. We empirically demonstrate that the fraction of PCVs that an instance has is positively correlated with local search cost — *i.e.* that hard instances tend to have more PCVs. We also show that sets of PCVs are additive in the sense that the run-time improvements realisable by simultaneously instantiating a number of PCVs is amortisable. In fact, we show that in a large number of cases (particularly for harder, and structured instances), only a few PCVs need be instantiated before the instance becomes completely trivial to solve. Also, we show that there is a significant amount of overlap between the sets of PCVs with respect to different algorithms, indicating that some PCVs are algorithm dependent.

We also show that performance criticality is closely related to the search space features studied in earlier chapters. In particular, we show that the average criticality factor of an instance is highly correlated with both solution reachability and the maximum trap attractivity of an instance. Furthermore, we demonstrate how the search space changes when the most critical variable is instantiated — not surprisingly, this increases solution reachability and decreases the attractivity of traps. Finally, we describe a mechanism that may be useful for constructing a practical PCV oracle.

It seems that with any scientific endeavour, resolving a few questions reveals a host of new problems, and this thesis was no exception. There are many possible directions in which this research may be extended. Firstly, we note again that our analyses throughout the thesis were restricted to fairly small SAT instances due to the computationally expensive nature of the empirical experiments being performed. One possible way to extend the analysis to larger instances is to only partially explore plateaus. In fact, we developed a method for the partial exploration of plateaus which works by storing only a single state to represent each plateau. When we are sampling states in order to measure the distribution of plateaus encountered during a series of SLS algorithm runs, a single state is stored to represent each encountered plateau. When we need to determine which existing plateau (if any) a given state s belongs to, we run a number of simple local search algorithms in parallel, each initialised to the plateau representative states, restricted to stay at the same level, and biased towards reducing the hamming distance to s. If a path is found from any of the representative states to s, then s is known to belong to the corresponding plateau, otherwise we use s as the representative state for a new plateau. Given the set of representative states, a partial (approximate) PCG can be constructed by perturbing each representative state, while restricting the perturbed state to the same level, in order to sample a number of states from the plateau. All of the lower level neighbours of these states can be found, and the containing plateau of each neighbour can be found using the technique described above. Preliminary experiments indicate that this technique holds a great deal of potential.

While we have attempted to make our experiments as complete and representative as possible, time constraints occasionally required that we only performed our experiments on a single representative case. Examples of this include the analysis of the connection between escape difficulty and mixture models in Section 4.4, the effect on the search space of instantiating the PCV with the highest criticality factor in Section 5.5, and the example PCV oracle heuristic, also in Section 5.5. While we do believe these examples are representative, we intend to expand these analyses to more instances in order to fully justify the respective claims.

The main focus of this thesis was the analysis of the relationship between search space structure and algorithm behaviour, and while we did suggest methods of exploiting the results we were not able to implement them. Some directions that we believe are particularly promising include schemes which explicitly detect and avoid closed plateaus, and the incorporation of a PCV heuristic (such as that described in Section 5.5) into a SAT solver.

Other directions for future work include the study of strong polynomial SAT solving techniques in the context of performance criticality (see, *e.g.*, [BW03]; such techniques could be used instead of unit propagation as a preprocessing stage after assigning a truth value to a PCV) and the investigation of the scaling of PCV set size and average criticality factors of PCVs with problem size for families of structured and random instances. It may also be interesting to study the concept of PCVs for unsatisfiable SAT instances;

this requires a slightly different definition, but similar intuitions and analysis techniques apply.

Finally, it should be noted that the concept of performance criticality can be easily applied to other combinatorial problems. Throughout this thesis, we have used SAT to introduce and study these concepts because of its conceptual simplicity and its prominent status in the theoretical and empirical study of algorithmic complexity and combinatorial problem solving. However, we believe it will be fruitful to study the concept of performance criticality for other combinatorial problems, particularly for widely studied optimisation problems, such as MAX-SAT, TSP, and more general constraint satisfaction problems.

It is our strong belief that in order to design high-performance SLS algorithms for SAT, it is necessary to understand, as completely as possible, properties of the space which is being searched. This thesis contributes to this understanding, and it is our hope that the concepts discussed within this thesis will provide insights leading to improvements in the state-of-the-art in SAT solving.

Bibliography

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3), 1992.

[BS03] D. Le Berre and L. Simon. The essentials of the SAT'03 competition. In *Submitted to LNAI*. June 2003.

[BW03] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Proc. SAT 2003*, pages 183–192, 2003.

[CA96] James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence Journal*, 81(1-2):31–57, 1996.

[CFG+96] D. Clark, J. Frank, I. Gent, E. MacIntyre, N. Tomov, and T. Walsh. Local search and the number of solutions. In E. Freuder, editor, *Principles and Practice of Constraint Programming - CP'96*, volume 1118 of *Lecture Notes in Computer Science*, pages 119–133. Springer Verlag, Berlin, Germany, 1996.

[CKT91] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.

[DD01] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proc. IJCAI-01*, pages 248–253, 2001.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[dlT90] T. Boy de la Tour. Minimizing the number of clauses by renaming. In *Proc. of the 10th Conference on Automated Deduction*, pages 558–572, 1990.

[FCS97] Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[GJ79] M.R. Garey and D.B. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York, 1979.

[GSK98] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proc. AAAI-98*, pages 431–437, Madison, Wisconsin, 1998.

[GW93] Ian Gent and Toby Walsh. An empirical analysis of search in gsat. *Journal of Artificial Intelligence Research*, 1:47–59, 1993.

[GW94] Ian Gent and Toby Walsh. The SAT phase transition. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94)*, pages 105–109, 1994.

[GW95] Ian Gent and Toby Walsh. Unsatisfied variables in local search. In Amsterdam J. Hallam, IOS Press, editor, *Hybrid Problems, Hybrid Solutions*, pages 73–85, 1995.

[HK95] S. Hampson and D. Kibler. Plateaus and plateau search in boolean satisfiability problems: When to give up searching and start again. In *DIMACS Challenge '95*, 1995.

[Hof85] Douglas R. Hofstadter. *Metamagical Themas: Questing for the Essence of Mind and Pattern*, chapter 13: Metafonts, Metamathematics, and Metaphysics. Basic Books, 1985.

[Hoo98] Holger H. Hoos. *Stochastic Local Search - Methods, Models, Applications*. PhD thesis, Technische Universität Darmstadt, 1998.

[Hoo99] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proc. AAAI-99*, pages 661–666, 1999.

[Hoo02] Holger H. Hoos. A mixture-model for the behaviour of sls algorithms for SAT. In *In Proc. AAAI'02*, pages 661–667, 2002.

[HS98] Holger H. Hoos and Thomas Stützle. Evaluating las vegas algorithms - pitfalls and remedies. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245, San Francisco, CA, 1998. Morgan Kaufmann Publishers.

[HS00] Holger H. Hoos and Thomas Stützle. SATLIB: An online resource for research on SAT. In *SAT 2000*. IOS Press, Amsterdam, The Netherlands, 2000.

[HS04] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann Publishers, USA, 2004.

[HTH02] Frank Hutter, Dave A.D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. CP 2002*, pages 233–248, 2002.

[Int03] Y. Interian. Backdoor sets for random 3-SAT. In *Proc. SAT 2003*, pages 231–238, 2003.

[KKRR92] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.

[KMS96] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proc. KR'96*, pages 374–384, 1996.

[KS96] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI'96*, pages 1194–1201, 1996.

[KS03] Henry Kautz and Bart Selman. Ten challenges redux: Recent progress in propositional reasoning and search. In *Ninth International Conference on Principles and Practice of Constraint Programming (CP 2003)*, 2003.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. DAC'01*, 2001.

[MSG97] B. Mazure, L. Sais, and É. Grégoire. Tabu Search for SAT. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 281–285. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1997.

[MSK97] David A. McAllester, Bart Selman, and Henry A. Kautz. Evidence for invariants in local search. In *Proc. AAAI-97*, pages 321–326, 1997.

[MT00] P. Mills and E. Tsang. Guided local search for solving SAT and weighted MAX-SAT problems. In I.P. Gent, H. van Maaren, and T. Walsh, editors, *SAT2000 — Highlights of Satisfiability Research in the Year 2000*, pages 89–106. 2000.

[NLBD+04] E. Nudelman, K. Leyton-Brown, A. Devkar, Y. Shoham, and H. Hoos. Understanding random sat: Beyond the clauses-to-variables ratio. In *International Conference on Principles and Practice of Constraint Programming (CP)*, 2004.

[Pap91] Christos H. Papadimitriou. On selecting a satisfying truth assignment (extended abstract). In *Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 163–169. IEEE Computer Society Press, 1991.

[Par97] Andrew J. Parkes. Clustering at the phase transition. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 340–345. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1997.

[PG86] D. A. Plaisted and S. Greenbaum. A structure preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice Hall, 1982.

[SBH02] Laurent Simon, Daniel Le Berre, and Edward A. Hirsch. The SAT2002 competition, 2002.

[Sch99] Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 410. IEEE Computer Society, 1999.

[SGS00] J. Singer, I. Gent, and A. Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, pages 235–270, 2000.

[SK93] B. Selman and H.A. Kautz. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proc. IJCAI-93*, 1993.

[SKC93] B. Selman, H.A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Proc. 2nd DIMACS Challenge on Cliques, Coloring, and Satisfiability*, 1993.

[SKC94] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. AAAI-94*, 1994.

[SLM92] B. Selman, H.J. Levesque, and D.G. Mitchell. A new method for solving hard satisfiability problems. In *Proc AAAI-92*, 1992.

[SW97] Y. Shang and B.W. Wah. Discrete lagrangian-based search for solving MAX-SAT problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, volume 1, pages 378–383. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.

[SW02] John Slaney and Toby Walsh. Phase transition behavior: from decision to optimization. In *In the Proceedings of the Fifth International Conference on Theory and Applications of Satisfiability Testing (SAT 2002)*, 2002.

[TH04] Dave A. D. Tompkins and Holger H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *In the Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, pages 37–46, 2004.

[Val79] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[WGS03] Ryan Williams, Carla Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proc. IJCAI-03*, Acaplco, Mexico, 2003.

[Yok97] Makoto Yokoo. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analysing landscapes of csp's. In *Proc. CP'97*, volume 1330 of *LNCS*, pages 357–370, 1997.

[ZRL03] W. Zhang, A. Rangan, and M. Looks. Backbone guided local search for maximum satisfiability. In *Proc. IJCAI-03*, pages 1179–1184, 2003.

Appendix A

Test Suite Description

The empirical nature of this thesis required that an interesting set of problem instances be selected for study. Choosing a representative set of SAT instances for an empirical study is a daunting, if not impossible, task. The instances must represent interesting problems that have been studied in the past and/or are relevant to practical applications. However, there are always restrictions on the number and size of instances that can be studied.

Many of the empirical experiments performed in this thesis were exceedingly computationally intensive, both with respect to computation time and storage. Because of this, we were restricted to fairly small SAT instances that are considered quite easy by modern standards. Nonetheless, we hope that even the study of small instances is interesting, because many of the observations and intuitions gained by studying small instances generalise to larger instances. When possible, we studied larger instances and contrasted the results with those for the smaller instances. Even after restricting the test sets to small instances, some of the experiments still took multiple CPU months to complete.

We study two main class of instances: random and structured. The random instances are interesting because they have been extensively studied in the past, so there is a wealth of literature to contrast results with. Another advantage of studying randomly generated instances is that distributions of syntactically identical instances can be generated and studied. The structured instances are generally much less available, and must be studied on and instance-by-instance basis. One exception to this rule is that we may generate random distributions of instances from other problem domains, such as graph colouring, and then encode these instances into propositional satisfiability.

The random instances that we study come from the Uniform Random 3-SAT distribution, near the phase-transition region [CKT91]. We generally refer to this entire class of instances as UF-3-SAT. Table A.1 shows the UF-3-SAT test sets that we study. The first collection of test sets, named *uf20-uf50*, were generated explicitly for this thesis. There are 100 instances in each test set, and the number of variables ranges from 20 to 50 in steps of 5. The clause to variable ratio in each test set is determined by Crawford and Auton's formula $C/V = 4.258 + 58.26v^{-5/3}$, which was empirically demonstrated to fit the phase transition region well, even for small instances [CA96]. We also used four set of larger instances, taken from SATLIB [HS00]. These sets, with 20, 50, 100, and 200 variables, also have a clause to variable ratio near the phase transition,

Benchmark	n	m	#
uf20	20	93	100
uf25	25	113	100
uf30	30	133	100
uf35	35	154	100
uf40	40	175	100
uf45	45	196	100
uf20	20	91	1000
uf50	50	218	1000
uf100	100	430	1000
uf200	200	860	1000

Table A.1: Description of the random benchmark sets studied in this paper; n, m, and # are the number of variables, clauses and instances, respectively, in each benchmark set.

Benchmark	n	m	#
ii8	(66-1068)	(186-8214)	14
`anomaly`	48	261	1
`medium`	116	953	1
`huge`	459	7054	1
`bwa`	459	4675	1
`loga`	828	6718	1
`ais6`	61	581	1
`flat20-easy`	60	116	1
`flat20-med`	60	116	1
`flat20-hard`	60	116	1
`par8-1-c`	64	254	1
`ssa7552-038`	1501	3575	1

Table A.2: Description of the structured benchmark sets studied in this paper; n, m, and # are the number of variables, clauses and instances, respectively, in each benchmark set. Note that most of the "sets" actually consist of a single instance.

but the actual number of clauses was determined empirically for each set by experimentally determining the point at which approximately 50% of the instances in the set were satisfiable.

Choosing interesting structured instances proved to be more challenging. We needed small instances, which were hard to come by and so already limited the set of potential instances. We also wanted instances syntactically different instances encoded from different problems. Table A.2 shows the structured instances used in the thesis, which are all available for download from SATLIB [HS00]. The *ii8* set consists of 14 instances of various sizes, and are from a SAT formulation of Boolean function synthesis problems [KKRR92]. The `anomaly`, `medium`, `huge`, and `bwa` sets all consist of a single instance, and correspond to SAT-encoded instances of the well-known blocks world problem domain. The `loga` instance is from a SAT formulation of a logistical planning problem. A more detailed description of the problem domains and the SAT encod-

ing used to generate these instances can be found in [KS96, KMS96]. The `ais6` instance a SAT formulation of the all-interval-series problem, inspired by a well-known problem in serial music composition [Hoo98]. The `flat20-easy`, `flat20-med`, and `flat20-hard` are SAT-encoded instances of the graph colouring problem. To obtain these instances, 1000 graph colouring instances were generated using Joe Culberson's flat graph generator,[1] each having 20 nodes, 3 colours, and a connectivity of 0.245. The graph colouring problems were then converted into propositional formulae using Joe Culberson's converting utility. These three instances are those with the lowest, median, and maximum local search cost for the GWSAT algorithm (*c.f.* Section 2.3) from the entire distribution of instances. The `par8-1-c` instance is a propositional version of a parity learning problem.[2] Finally, the `ssa7552-038` instance is from a hardware verification problem testing for *single-at-stuck* faults.[3]

There are quite a number of syntactically different structured instances represented in our test sets, though the number of instances is fairly low. Combined with the large number of randomly generated instances, we are confident that we have chosen an interesting set of instances for study, which are still small enough for our empirical experiments to be computationally feasible.

[1] Available online at `http://web.cs.ualberta.ca/~joe/Coloring/Generators/flat.html`

[2] More details regarding the problem and the propositional encoding may be found on the DIMACS website `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/crawford/README`.

[3] More details regarding the problem and the propositional encoding may be found on the DIMACS website `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances/cnf-ssa/README`.

Appendix B

Binary Decision Diagrams

Reduced Ordered Binary Decision Diagrams (ROBDDs, or BDDs for short) are a data structure useful for compactly representing Boolean functions using Directed Acyclic Graphs (DAGs). They represent functions canonically and efficiently (both in terms of space and time complexity), making them popular in practical applications. For a thorough introduction to BDDs, the reader is referred to [Bry92]. For all of the experiments conducted in this thesis, we used the popular BDD package CUDD version 2.4.0, available online at `http://vlsi.colorado.edu/~fabio/CUDD`.

Throughout this thesis, we use BDDs to store very large sets of (related) search space states. In order to do this, we assume that states are binary vectors of the form $\vec{X} = (x_1, x_2, \cdots, x_n)$ (where n is the number of propositional variables). Such a vector can be interpreted as the input to a Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$. Thus, we can represent a set of states with such a Boolean function by adopting the convention that a state $\vec{X}$ is in the set defined by f if and only if $f(\vec{X}) = 1$ (*i.e.* the output is interpreted as true). By using a BDD to represent f, we can efficiently store and manipulate the corresponding set of binary states.

Figure B.1 illustrates the relationship between Boolean functions, binary trees, and binary decision diagrams. Each of the three data structures represents the same set of three-variable states $\{011, 110, 111\}$, but the BDD does so using the least amount of space. The BDD uses two rules to reduce the number of nodes

X1	X2	X3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

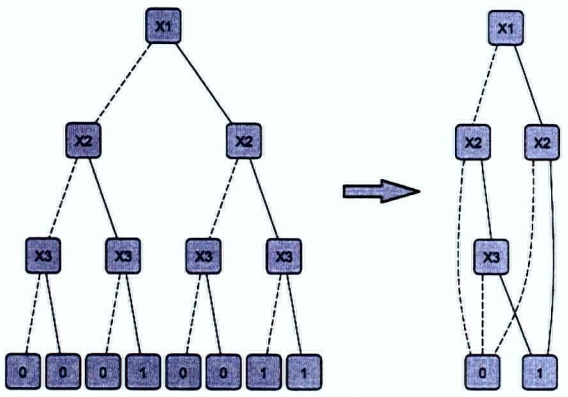

Figure B.1: Comparing three different representations of the same Boolean function. Left: Truth Table, Centre: Binary Tree, Right: Binary Decision Diagram. In the diagrams, dashed edges symbolise a value of 0 being assigned to the source variable, while solid edges symbolise a value of 1.

used. First, if both of the outgoing edges of a node n_1 point to the same node n_2, then n_1 is redundant and can be removed and any incoming edges can be pointed directly at n_2. Second, equivalent nodes must be removed.

In addition to the space savings, a further advantage of BDDs is that it is possible (and efficient) to calculate the intersection and union of sets of states simply by using the BDD *and* ($\wedge$) and *or* ($\vee$) operations. These operations were crucial for efficiently calculating $Levelset^k$ and partitioning the set into individual plateaus (recall that both $Levelset^k$ and plateaus were stored using BDDs).

The pseudo-code of Algorithms 3.1 and 3.2 reference non-standard BDD operations that we will now define. $BDDSatSet(LiteralSet\ c)$ takes as input a set of literals (interpreted as a disjunction, forming a clause) and returns the set of states consistent with the input clause. Of course, the output is returned as a BDD. Hence, this function is trivial: we simply build a small BDD representing the disjunction (it should be easy to see that the number of nodes in such a BDD must be linear in the number of literals). The minterms of this function represent the desired states. $BDDPickOneCube(BDD\ f)$ returns an arbitrary cube of the input BDD. A cube is simply a conjunction of literals (all variables do not necessarily need to be represented in the conjunction) that is consistent with at least one minterm of the BDD. In our context, a cube is a very compact representation of a subset of the states represented by the original BDD. Finding a cube of a BDD is simple; simply find a path to the node labelled "1", and then construct a new BDD out of all literals appearing along that path. Finally, $BDDNegate(BDD\ f, int\ var)$ exchanges the outgoing edges of all nodes associated with variable *var*. This has the effect of simultaneously flipping variable *var* in all of the states in the implicitly represented set.