Stochastic Local Search — Methods, Models, Applications

Dissertationsschrift in englischer Sprache
vorgelegt am Fachbereich Informatik
der Technischen Universität Darmstadt von

Dipl.-Informatiker Holger H. Hoos
aus Frankfurt am Main

zur Erlangungen des Grades eines
Doktors der Naturwissenschaft (Dr. rer. nat)

Darmstadt 1998
Hochschulkennziffer D17

Betreuer und Erstreferent: Prof. Dr. Wolfgang Bibel
Korreferent: Prof. Dr. Hermann K.-G. Walter
Externer Gutachter: Prof. Dr. Bart Selman (Cornell University, USA)

Tag der Einreichung: 30. September 1998
Tag der Disputation: 11. November 1998

Contents

Prologue **1**

1 Introduction: Logic, Search, and $\mathcal{NP}$-hard Problems **7**

1.1 AI Problems, Logic, and Search 7

1.2 Systematic versus Local Search 9

1.3 Generic Problem Solving Using Stochastic Local Search 11

1.4 $\mathcal{NP}$-Complete and $\mathcal{NP}$-Hard Problems 13

1.5 Propositional Logic and Satisfiability (SAT) 15

1.6 Constraint Satisfaction Problems (CSP) 19

1.7 Related Work 20

1.8 Conclusions 20

2 Empirical Analysis of Las Vegas Algorithms **23**

2.1 Motivation and Background 23

2.1.1 Las Vegas Algorithms 25

2.1.2 Application Scenarios 26

2.2 RTDs and RLDs 27

2.2.1 CPU-times *vs* Operation Counts 28

2.2.2 Empirical Evaluation Based on RLDs 29

2.3 Pitfalls of Inadequate Methodology 36

2.3.1 Problem Type 1: Superficial Analysis of Run-time Behaviour 36

2.3.2 Problem Type 2: Inhomogeneous Test-sets 40

2.4 Extension to Optimisation Problems 43
2.4.1 Norms of Behaviour 43
2.4.2 Application Scenarios 44
2.4.3 Qualified RTDs and RLDs 45
2.4.4 Evaluating Optimisation LVAs 47
2.5 Related Work 48
2.6 Conclusions 50

3 Generalised Local Search Machines 53

3.1 Local Search Algorithms 53
3.2 The Basic GLSM Model 55
3.3 GLSM Semantics 57
3.4 GLSMs and Local Search 59
3.5 Machine Types 60
3.6 Transition Types 62
3.7 State Types 65
3.8 Extensions of the Basic GLSM Model 66
3.9 Related Work 69
3.10 Conclusions 71

4 SLS Algorithms for SAT 73

4.1 Introduction 73
4.2 The GSAT Architecture 74
4.2.1 Basic GSAT 74
4.2.2 GSAT with Random Walk (GWSAT) 76
4.2.3 GSAT with Tabu Search (GSAT/TABU) 77
4.3 The WalkSAT Architecture 77
4.3.1 WalkSAT 78
4.3.2 WalkSAT Variants 80
4.3.3 Novelty 80

4.3.4 R-Novelty 82
4.4 The Benchmark Set 83
4.4.1 Uniform Random-3-SAT 84
4.4.2 Graph Colouring 86
4.4.3 Blocks World Planning 88
4.4.4 All-Interval-Series 90
4.5 Comparing SLS Algorithms 91
4.5.1 Random-3-SAT 92
4.5.2 Graph Colouring 98
4.5.3 Blocks World Planning 102
4.5.4 All-Interval-Series 104
4.6 Related Work 107
4.7 Conclusions 109

5 Characterising and Improving SLS Behaviour 111

5.1 Characterising SLS Behaviour 111
5.1.1 Asymptotic Run-time Behaviour 112
5.1.2 Functional Approximations for Optimal Noise 116
5.1.3 Modelling the Initial Search Phase 123
5.1.4 Functional Approximations for Non-optimal Noise 128
5.1.5 Consequences and Interpretations 132
5.2 Improving Existing SLS Algorithms 134
5.2.1 Hybrid GLSM Extensions 134
5.2.2 Parallelisation Based on Multiple Independent Tries 138
5.2.3 Parallelisation Based on Algorithm Portfolios 141
5.3 Related Work 143
5.4 Conclusions 144

6 Search Space Analysis 147

6.1 Motivation and Background 148

6.2 The Number of Solutions 149
6.3 Stratified Random-3-SAT 154
6.4 Sampling the Objective Function 156
6.5 State Types and their Distribution 162
6.6 Local Minima Branching 167
6.7 Related Work 173
6.8 Conclusions 174

7 SAT Encodings and Search Space Structure 177

7.1 Motivation and Background 178
7.2 Constraint Satisfaction Problems 179
7.2.1 SAT Encodings of the CSP 179
7.2.2 Benchmark Instances 181
7.2.3 Two SLS Algorithms for CSP 181
7.2.4 SLS Performance 182
7.2.5 Search Space Structure 189
7.3 Hamilton Circuit Problems 190
7.3.1 SAT Encodings of the HCP 191
7.3.2 Benchmark Instances 192
7.3.3 SLS Performance 193
7.3.4 Search Space Structure 196
7.4 Related Work 198
7.5 Conclusions 200

Epilogue 202

Bibliography 207

List of Symbols / Abbreviations 216

Prologue

Today, stochastic local search (SLS) algorithms belong to the standard methods for solving hard combinatorial problems from various areas of Artificial Intelligence (AI) and Operations Research (OR). Some of the most successful and powerful algorithms for prominent problems like SAT (the Satisfiability Problem in Propositional Logic), CSP (the Constraint Satisfaction Problem), TSP (the Traveling Salesperson Problem), or QAP (the Quadratic Assignment Problem), are based on stochastic local search. Many stochastic local search methods like stochastic hill-climbing, Tabu Search, and Simulated Annealing can be generically applied to a broad range of combinatorial decision and optimisation problems, including problems from practically important application areas, like planning and scheduling. Other prominent and successful algorithmic schemes, like Evolutionary Algorithms or Ant Colony Optimisation, are also based on the concept of stochastic local search.

In AI, the success story of stochastic local search is tightly linked to its application to SAT. SAT is a conceptually simple combinatorial decision problem which plays a prominent role in complexity theory, because of its status as *the* prototypical $\mathcal{NP}$-complete problem. All $\mathcal{NP}$-complete problems can be polynomially translated into SAT – these reductions are often used in the context of $\mathcal{NP}$-completeness proofs. However, recent successes in efficiently solving SAT-encoded planning problems using modern SLS algorithms suggest that SAT-reductions might provide a practically viable way for solving hard combinatorial problems from application domains. SAT has a number of advantages over other $\mathcal{NP}$-complete problems. Its conceptual simplicity facilitates the development and implementation of SLS algorithms, including hardware realisations. Furthermore, SAT encodings are often well-known and easily realised. Finally, SLS algorithms for SAT can typically be generalised to CSP, which allows somewhat more concise and natural representations, in a rather straightforward way. One of the big advantages of SLS algorithms is the fact that, different from most systematic search methods, the algorithms for solving decision problems can equally well be used to solve the related optimisation problems.

Considering these arguments, the research presented in this thesis focusses mainly on SLS algorithms for SAT. However, most of the methods and models we developed and used are not restricted to SAT, but merely applied to SAT as an exemplary problem domain. For this reason, two of the main contributions of this work, the novel methodology for empirically evaluating Las Vegas algorithms (Chapter 2 and the GLSM model for local search (Chapter 3), are presented in a more general context before they are applied to

SAT. Interestingly, we found substantial evidence that the characterisation results regarding the behaviour of some prominent SLS algorithms for SAT (Chapter 5) also apply to SLS algorithms for other combinatorial problems, like CSP.

Since the early 1990s, when simple SLS algorithms where shown to outperform the best systematic algorithms for SAT on a number of domains, SLS algorithms have become increasingly popular in AI. The general interest in SLS algorithms is documented by a growing body of research. Studies on SLS algorithms play an important role at all major AI conferences, such as the International Joint Conferences on Artificial Intelligence (IJCAI), the European Conferences on Artificial Intelligence (ECAI), and the conferences of the American Association for Artificial Intelligence (AAAI). Furthermore, research on SLS algorithms is regularly presented at more specialised AI conferences, such as the International Conferences on Principles and Practice of Constraint Programming (CP) and the Conferences on Uncertainty in Artificial Intelligence (UAI). There is also a substantial number of articles on SLS algorithms in prestigious journals, like the Artificial Intelligence Journal or the Journal on Artificial Intelligence Research. SLS algorithms are also prominent in the Operations Research community, which began to study and apply them extensively before they became as popular in AI as they are today. Generally, stochastic local search can be considered as a young, but prominent and rapidly evolving research area in computer science in general and within AI in particular.

The research described in this thesis has been done between spring 1996 and autumn 1998 at the Computer Science Department of the Darmstadt University of Technology (Technische Universität Darmstadt, TUD) in Darmstadt, Germany. This work is based on my earlier experience with local search algorithms for SAT which I acquired during my M.Sc. studies (German: *Diplomstudiengang*) and builds on research issues which evolved in the context of my master's thesis (German: *Diplomarbeit*). A significant part of the work was done during a one-year research visit at the Laboratory for Computational Intelligence at the Computer Science Department of the University of British Columbia (UBC) in Vancouver, Canada which was made possible by a scholarship from the German National Merit Foundation (Studienstiftung des Deutschen Volkes).

The results of this Ph.D. thesis have partly been presented at international conferences and in research colloquia at TUD and UBC. Furthermore, some of the methods and results presented in Chapters 2 and 5, although original work of the author, appeared in joint publications with Thomas Stützle (TUD). Specifically, a part of Chapter 2 was presented at the 14th Conference for Uncertainty in Artifical Intelligence [HS98a]; some of the results presented in Chapter 5 are covered in a journal article which is currently under preparation and also appear in a poster presentation at the Fourth International Conference on Principles and Practice of Constraint Programming [HS98b].

This thesis was written in English to make its results directly accessible to the international scientific community and to allow its being reviewed by Prof. Bart Selman (Cornell University, USA), one of the leading experts in the field of stochastic local search.

Outline of the Thesis

The thesis is divided into nine major sections: this Prologue, Chapters 1–7, and an Epilogue. Each of the Chapters 1–7 begins with a short introduction and ends with a discussion of related work and a summary of its main results and contributions. Thus, the individual chapters, although their contents are closely related to each other, are self-contained to a certain degree. This structure reflects the fact that each chapter focusses on different research issues and methodological approaches. It has been chosen to facilitate selective reading of this thesis, as in the case of a reader who is mainly interested in the specific subtopic covered by one of the chapters.

In Chapter 1, we give the background and motivation for the work presented in this thesis. The stage is set by giving a brief introduction to AI problems, logic, and search; then, we discuss systematic versus local search and informally introduce stochastic local search methods. Next, we introduce and discuss the concept of generic problem solving using SAT as an intermediate problem domain, and stochastic local search methods for solving these problems. This is followed by an introduction into $\mathcal{NP}$-complete and $\mathcal{NP}$-hard problems and the methods used for solving these in practice. Based on formal definitions of the syntax and semantics of Propositional Logic, we then define the tightly related Satisfiability and Validity problems and discuss their complexity. Finally, we formally introduce the finite discrete Constraint Satisfaction problem and discuss its relation to SAT.

Chapter 2 covers the empirical methodology which is later used for evaluating stochastic local search algorithms and for characterising their behaviour. The previously existing methodology in this field suffers from several weaknesses, which can lead to misinterpretations and false conclusions. We discuss these weaknesses and develop a novel methodology for empirically evaluating stochastic local search algorithms in the more general framework of Las Vegas Algorithms. Our new approach allows a considerably more detailed analysis of these algorithms' behaviour without requiring additional overhead in data aquisition. At the same time, it is not restricted to local search algorithms or particular problem domains, but can be applied to any algorithm of Las Vegas type.

Chapter 3 introduces the GLSM (Generalised Local Search Machine) model, a novel formal framework for local search algorithms based on the concept of an extended non-deterministic finite state machine. This new model is based on the intuition that adequate local search algorithms are usually obtained by combining simple search strategies. The GLSM model allows the adequate representation of most modern SLS algorithms for SAT and other combinatorial problems; additionally, it provides conceptually simple ways for extending and re-combining these into new, hybrid local search schemes and facilitates their implementation. In the context of this work, GLSM models are used for formalising SLS algorithms for SAT (Chapter 4), for improving their behaviour (Chapter 5), and, to some extent, for analysing the search space structure their performance depends on (Chapter 6). However, these applications merely indicate the full power of the GLSM model and its extensions.

The main topic of Chapter 4 is a comprehensive comparison of several modern SLS algorithms for SAT, including some of the latest and most successful SAT algorithms. We formalise and discuss these algorithms, which are instances of the GSAT and WalkSAT architectures, within the GLSM framework and empirically evaluate their performance using a newly compiled suite of benchmark problems. This benchmark suite comprises test-sets sampled from random problem distributions as well as SAT-encoded instances from other problem domains. Our comparative performance study shows that while some algorithms are clearly dominated by others, there is no single best algorithm. Instead, there is a group of algorithms the performance ranking of which depends on the problem domain and instance size. Nevertheless, we show that essentially the same problem instances tend to be hard for all algorithms considered here. Therefore, it makes sense to assume an intrinsic hardness of problem instances. Surprisingly, the top-performing algorithms show an exceedingly poor performance for a number of problem instances from different domains. This indicates that the corresponding local search processes suffer from early stagnation, which severly compromises their performance, especially for hard structured problem instances.

In Chapter 5, we analyse SLS behaviour in more detail, using theoretical argument as well as empirical methodology for characterising the behaviour of several prominent algorithms of the WalkSAT and GSAT families. For the first time, we prove that while GWSAT is approximately complete (*i.e.*, for arbitrarily long runs, the probability of solving a soluble problem instance approaches one), this does not hold for modern WalkSAT variants, like WalkSAT+tabu, Novelty, and R-Novelty. This explains the stagnation behaviour observed for these algorithms in Chapter 4. Based on this result, we show how to extend these algorithms such that approximately complete behaviour is achieved. This leads to new variants of Novelty and R-Novelty for which we observe significantly improved performance. We also develop a functional characterisation of the empirically observed SLS behaviour. This model shows that when applied to hard problem instances from various domains, all the algorithms we analysed exhibit approximately exponential run-time distributions. This novel and rather surprising result has interesting implications regarding the effectivity of using random restarts and suggests a novel interpretation of SLS behaviour. As another consequence, these algorithms can be optimally parallelised by using multiple independent runs, a particular simple and attractive parallelisation scheme. We also discuss the more general portfolio approach and its potential benefits in the context of SLS algorithms for SAT.

In Chapter 6, the main objective is to improve our understanding of SLS behaviour by analysing features of the search spaces underlying the local search process. While some of our analyses refine and extend previous work, such as the influence of the number of solutions on SLS performance, others are based on features which have not been studied before, such as the distribution of objective function values across the search space or local minima branching along SLS trajectories. Our analysis shows that, while the number of solutions has a dominating influence on SLS performance, other factors, such as the standard deviation of the objective function and the branching of local minima have also significant effects. Regarding the structure of local minima regions, we show that these resemble brittle canyon structures rather than compact basins, which explains the effectivity of simple escape mechanims like random walk. Interestingly, the local minimum branching is considerably lower for SAT-encoded instances from other domains than for randomly generated instances. Our analysis does not explain all the observed performance differences, but it improves on the previous understanding of SLS behaviour. While we focus on SLS algorithms for SAT, the same empirical methodology can be used to study search space structure and its influence on SLS performance for other domains of combinatorial problems.

Chapter 7 provides an initial investigation of the influence of encoding strategies on search space structure and SLS performance. Based on two case studies, covering Constraint Satisfaction Problems (CSPs) and the Hamilton Circuit Problem (HCP), we exemplify how some of the methods developed before (in particular, in Chapters 4–6) can be applied in this context. We study different SAT-encodings for CSP and HCP and analyse the differences between compact encodings, (which minimise the search space size) and sparse encodings, as well as the effects of eliminating symmetric solutions. Furthermore, for CSP, we compare the performance of state-of-the-art SLS algorithms for SAT applied to SAT-encoded CSP instances with the performance of conceptually related SLS algorithms for CSP, which are directly applied to the un-encoded problem instances. While the scope of our investigation is too limited to produce definitive answers to the problems we address, our results suggest some testable hypotheses on the effectivity of encoding strategies and the generic problem solving approach from Chapter 1. Furthermore, our study demonstrates how the techniques developed for the evaluation of SLS performance and the analysis of search space structure can be used for studying the impact of SAT-encodings on search space structure and SLS behaviour.

The work concludes with an Epilogue, where we summarise its main contributions and point out open questions as well as directions for further research.

Acknowledgements

This work would have been impossible without the support of many people. First and foremost, I would like to thank my family, especially my parents, my sister, and my brother, who sent me on my way and provided a stable and stimulating environment for my personal and intellectual development. Furthermore, I gratefully acknowledge the constant support of Sonia M. Ziesche, my loving and caring companion for many years now, who influenced my life and work in many ways, and whose commenting and proof-reading of this thesis was an important contribution to its final form. I am greatly indepted to my friend and colleague Thomas Stützle, whom I would like to thank for the many fruitful discussions, the joint work on research projects, and the fun we had while I was working on my Ph.D. project.

I would like to thank Prof. Wolfgang Bibel, my thesis supervisor, for his generous support and valuable advice during my work on this thesis. I am very grateful that I had the opportunity to work as a member of his Intellectics Group at TUD for the past few years; during this time, I enjoyed great freedom in pursuing my research interests and gathered a wealth of invaluable experience in many aspects of academic life. I also thank the members (and ex-members) of the Intellectics Group, in particular Maria Tiedemann, Steffen Hölldobler, Antje Strohmeier, Olaf Steinmann, Thomas Rath, and Stephan Schmitt, for their support and many interesting discussions.

Furthermore, I would like to thank David Poole, Craig Boutillier, Ronen Brafman, Alan Mackworth, and all the members of the Laboratory for Computational Intelligence for their hospitality, their support and the many interesting and helpful discussions we had during my stay at UBC. My special thanks to Valerie McRae for helping me find my way at UBC and in Vancouver, for sharing many thoughts and impressions, and for bringing some sunshine in my day when the skies were clouded.

I am also indepted to Prof. Hermann K.-G. Walter, who supported my work on the SALIERI Computer Music Project during the past years. Although being not directly related to this thesis, these activities have been very important for me in many ways. I would also like to thank my colleagues and friends, Jürgen Kilian and Kai Renz from the SALIERI Project Group, as well as Keith A. Hamel from UBC for the exciting joint work and the good times we had.

Finally, I gratefully acknowledge the help and support of Bart Selman, David McAllester, and Henry Kautz, who inspired my work with their research, generously shared their knowledge and resources, and helped to clarify and improve my ideas and presentation. I would also like to thank the German National Merit Foundation (Studienstiftung des deutschen Volkes), who sponsored my stay at UBC and, through their summer universities, provided many unique opportunities for meeting interesting people, as well as for forming and exchanging ideas. In this context, I would like to thank Prof. Manfred Kluge, Prof. Friedrich Beck, Gerhard Vöpel, Marius Böger, and Dr. Max Brocker for their help and support.

As undoubtedly, this list of acknowledgements is essentially incomplete, I would like to thank all the people who supported my work in many different ways and accompanied me on my way thus far.

Chapter 1

Introduction: Logic, Search, and $\mathcal{NP}$-hard Problems

The aim of this introductory chapter is to give the background and motivation for the work presented in this thesis. After a brief general introduction to Artificial Intelligence and the types of problems arising in this field of computer science, we discuss and compare the concepts of systematic and local search. We then present the model of generic problem solving which provides the motivation for the major part of the topics investigated in this thesis. Next, we give a short introduction to $\mathcal{NP}$-complete and $\mathcal{NP}$-hard problems, after which we introduce two fundamental $\mathcal{NP}$-hard problems, the satisfiability problem in propositional logic (SAT) and the constraint satisfaction problem (CSP), which form the basis for a major part of this thesis. Finally, we give some pointers to the literature related to the topics covered here and conclude with a short summary.

1.1 AI Problems, Logic, and Search

Within computer science, a relatively young and still rapidly growing academic discipline with a vast impact on a significant part of our modern societies, some of the most intriguing problems can be found in a field called Artificial Intelligence (AI), or Intellectics[1]. The general goal of AI is to realise intelligent behaviour in machines, an idea which by far predates the construction of the first universal computers. Since this basic motivation is tightly correlated with general questions regarding human intelligence, cognition, and the workings of the mind, AI problems are often, if not always, closely related to disciplines other than computer science, such as Psychology, Philosophy, and Neurobiology.

Some of the most prominent problems in AI are in the areas of language understanding, the-

[1]This name has been coined by Wolfgang Bibel, it refers to the discipline uniting traditional AI and certain areas of cognitive science [Bib80].

orem proving, game playing, pattern recognition, and musical composition. Unfortunately, most of these tasks involve computationally very hard problems and despite quite impressive successes in many areas of AI, today still most general AI problems lack completely satisfactory and feasible solutions. Like all problems in computer science, AI problems can be categorised in two major types: representational problems and computational problems. While the former problem type involves finding a suitable representation or formalisation of a problem from some domain, the latter type is about finding algorithms to solve these problems. Of course, both types of problems are closely related, since usually solving a real-world problem involves both finding a suitable representation and devising an algorithm to solve the formalised problem. Moreover, the construction and implementation of problem solving algorithms often critically depends on the problem being represented in a suitable way, just as certain representations almost naturally lead to specific algorithms.

In AI, problems are often represented in the framework of formal logics. A formal logic $\mathcal{L}$ is given by a formal language which specifies the syntax of logical expressions, and a mapping of these expressions to a system of truth values specifying the semantics of $\mathcal{L}$. Examples for formal logics which are used and studied in AI are propositional logic, first-order logic, second and higher order logics, modal logics, temporal logics, and action logics. Sentences which are true in a given logical system are called *theorems*. Theorems can be proven semantically, using formal reasoning outside the logical system. Alternatively, syntactical prove mechanisms called *calculi* can be used for establishing truth or falsehood of logical sentences. A logical calculus is a formal system consisting of a set of *axioms*, which are elementary true sentences, and a set of *rules*, which define truth preserving transformations of sentences. Calculi are the basis for the automatisation of logic. However, not for every logical system semantical truth can be characterised by a calculus, *i.e.*, for certain logical systems, there are semantically true sentences which cannot be algorithmically proven. In these logics, theoremhood is undecidable. First-order logic is of this type, while for propositional logic the validity of sentences can be decided algorithmically.

Many AI problems, particularly the ones arising in the context of formal logics, are decision problems, where solutions are characterised by a set of conditions which have to be satisfied in order to solve the problem. Classical theorem proving is usually formulated as a decision problem. Here, the solutions are the proofs of the given formula within a given proof system (or calculus). However, problems from application domains are often rather optimisation than decision problems. Optimisation problems can be seen as generalisations of decision problems, where the solutions are additionally valued by an objective function. There are two types of optimisation problems, a maximisation and a minimisation variant. For these problems, the goal is to find optimal solutions, *i.e.*, solutions with maximal resp. minimal value of the objective function. The theorem proving example from above can be formulated as an optimisation problem, if we take the proof length as the objective function and modify the problem goal to finding minimal length proofs. As we will see in the course of this work, many algorithms for decision problems can be extended to optimisation problems in a rather natural way. However, this is not always the case, since for many optimisation problems finding valid solutions is not very hard, whereas finding optimal or near-optimal solutions is computationally very expensive.

In many cases, decision and optimisation problems in AI can be characterised as search problems, *i.e.*, problems which involve some sort of searching through a number of solution candidates or partial solutions while trying to find an actual, valid (and maybe optimal) solution for the given problem instance. Many problems from game playing, theorem proving, and other areas of AI can be formulated as search problems in this sense. Note, however, that algorithms for solving search problems need not always be classical search algorithms. Actually, the search aspect does not not even have to be explicitly represented in such algorithms. Nevertheless, the abstract notion of search is fundamental for many AI problems and algorithms. As mentioned before, many AI problems are computationally very hard. Using the search metaphor, this is usually reflected by the fact that for a given problem instance, the set of solution candidates or partial solutions which defines the space in which the search takes place, is very large and no algorithmic methods for effectively finding actual solutions in these huge search spaces are known. A very typical situation is given in the case of problems where the time complexity of finding solutions grows exponentially with the problem size, while to decide whether a solution candidate is an actual solution can be done in polynomial time. It is in this situation, where algorithms based on the search metaphor are most frequently applied.

1.2 Systematic versus Local Search

Search methods can be roughly classified into two categories: systematic and local search.[2] Systematic search algorithms traverse the search space of a problem instance in a systematic manner which guarantees that eventually either a solution is found, or, if no solution exists, this fact is determined with certainty. This typical property of algorithms based on systematic search is called completeness. On the other hand, local search initialises the search process in some point of the search space and then procedes by iteratively moving from the present location to a neighbouring location, where the decision on each step is based on local knowledge only. Even if the local decisions are made in a rather systematic way, typically local search algorithms are incomplete, *i.e.*, even if the given problem instance has a solution, they are not guaranteed to find it eventually. Also, local search methods can visit the same location within the search space more than once. Actually, they are frequently prone to getting stuck in some part of the search space which they cannot escape from without special mechanisms like a complete restart of the search process or some other sort of diversification steps.

It might appear that, due to their incompleteness, local search algorithms are generally inferior to systematic methods. But as will be shown later, this is not quite true. Firstly, many problems are constructive by nature, it is known that they are solvable and what is required is actually the generation of a solution rather than just deciding whether a solutions

[2] Systematic search often exploits structural features of the problem representation and may thus also be referred to as "structural search"

does exist. This holds in particular for optimisation problems, like the Traveling Salesperson Problem (TSP) where the actual problem is to find a solution of sufficient optimality, but also for underconstrained decision problems which are quite common.

Secondly, in a typical application scenario often the time to find a solution is limited. Examples for such real-time problems can be found in virtually all application domains. Actually one might argue that almost every real-world problem involving interaction with the physical world, including humans, has real-time constraints. (Common examples are robot motion planning and decision making, most game playing situations, and speech recognition for natural language interfaces.) In these situations, systematic algorithms often have to be aborted after the given time has been exhausted, which — of course — renders them incomplete. This is particularly problematic for systematic optimisation algorithms which search through spaces of partial solutions; if those are aborted usually no solution candidate is available, while in the same situation local search algorithms typically offer the best solution found so far.[3] Ideally, algorithms for real-time problems should be able to deliver reasonably good solutions at any point during their execution. For optimisation problems this typically means that run-time and solution quality should be positively correlated, for decision problems one could imagine to guess a solution when a time-out occurs, where the accuracy of the guess should increase with the run-time of the algorithm. This so-called *any-time property* of algorithms is usually difficult to achieve, but in many situations the local search paradigm seems to be better suited for devising any-time algorithms than systematic search models.

As a matter of fact, systematic and local search algorithms are somewhat complementary in their applications. A nice example for this can be found in [KS96], where a fast local search algorithm is used for finding actual solutions for planning problems the optimality of which is proven by means of a systematic algorithm. As we will discuss later in more detail, different views of the same problem may in certain cases, particularly if reasonably good solutions are required within a short time using parallel computation and the knowledge about the problem domain is very limited, call for local search algorithms. In other cases, usually if exact or optimal solutions are required, time constraints are less important and some knowledge about the problem domain can be exploited, systematic search will be the better choice. Finally, there is some evidence that for certain problem classes, local and systematic search methods are most effective on different subclasses of instances. Unfortunately, to date the general question on when to prefer systematic and when local search methods remains mainly open.

Often, local search algorithms make heavily use of stochastic mechanisms, such as probabilistic tie-breaking heuristics. These algorithms are called *stochastic local search (*or *SLS) algorithms*, and have been used for many years now in the context of combinatorial optimisation problems. Among the most prominent algorithms of this kind we find the Lin-Kernighan algorithm [LK73] for the Traveling Salesperson Problem, as well as general methods like Tabu Search [Glo89], and Simulated Annealing [KJV83]. Lately it has be-

[3] However, it should be noted that, though quite uncommon so far, local search algorithms can be also designed in such a way that they are searching in a space of partial solutions.

come evident that stochastic local search algorithms can also be very successfully applied to the solution of $\mathcal{NP}$-complete decision problems such as the Graph Colouring Problem (GCP) [HD87, MJPL90, MJPL92] or the Satisfiability Problem in propositional logic (SAT) [SLM92, GW93a, Gu94]. The latter is particularly interesting for AI research because it is not only one of the basic problems arising in a (rather simple) context of logic-based approaches, but it can also be used in the context of a generic problem solving approach where basic problems from other domains, such as Blocks World Planning or Graph Colouring, are solved by encoding them into SAT and subsequently solving the encoded instances using stochastic local search techniques. Recent research results indicate that this approach is viable for at least some problem domains [KS96, JKS95] and consequently today there is considerable interest, not only in the AI community, in the questions connected with it and the research in this areas.

1.3 Generic Problem Solving Using Stochastic Local Search

Much of the interest in efficient algorithms for rather abstract but syntactically simple problems, like the satisfiability problem in propositional logic (SAT), is based on the notion of generic problem solving. There are two variants of this indirect approach, the weaker of which relies on the assumption that general techniques can be identified which are effective for dealing with a broad range of problem classes. If this is correct, developing and studying algorithms on syntactically simple problems is a very reasonable approach. The stronger variant assumes that it is possible to solve problems by transforming them into some domain for which an effective universal solver is available. The solutions that are thus obtained are then back-transformed into the original problem domain which yields the desired solutions for the problem at hand.

This approach of generic problem solving relies on a number of assumptions. First of all, the domain the problems are transformed into (the *intermediate domain*) has to provide sufficient expressive power, *i.e.*, a significant number of interesting problem classes have to be representable in this domain. Next, the complexity of the transformations between the domains has to be sufficiently low compared to the complexity of finding a solution, otherwise the overhead involved in this indirect approach might easily outweigh its advantages. Finally, the most crucial assumption is the existence of a sufficiently general solver in the intermediate domain.

One could argue, that these assumptions are critical in the sense that at least for some of them there is some doubt on whether they can be met in practice. There is a number of domains, which are sufficiently expressive to represent a wide variety of interesting problem classes. For some of these, like SAT and CSP, efficient transformations have been found for many problem classes [SW93] such that there is some indication that the extra costs of encoding/decoding are almost negligible when compared to the costs of finding solutions. It is, however, not clear whether there are sufficiently universal solvers for these intermediate domains and how these are affected by the choice of problem encoding. Very probably there

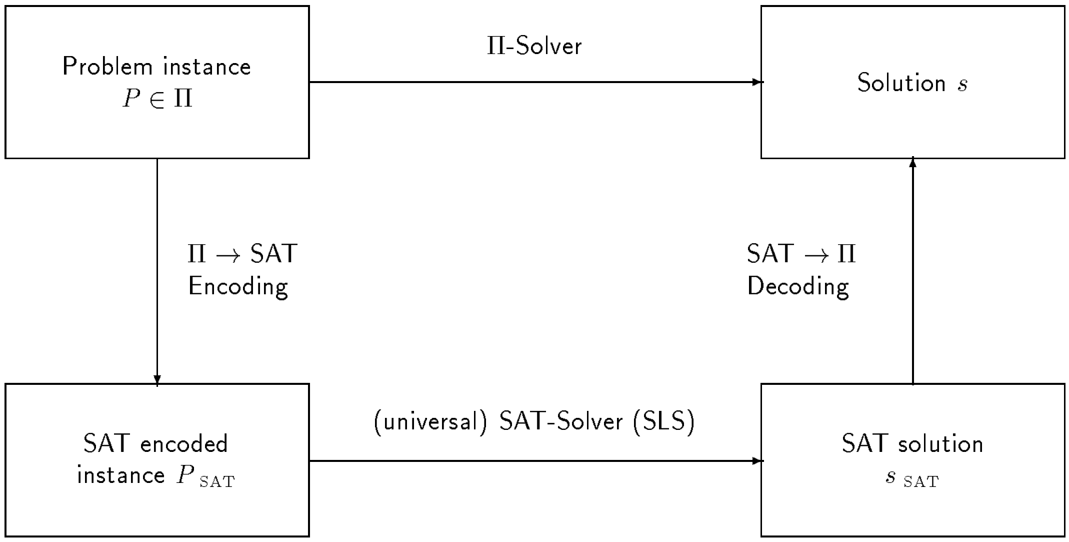


Figure 1.1: Generic problem solving by transformation.

will be a tight link between the problem encoding and a given algorithm's performance. But even assuming that one could find an intermediate-domain algorithm which achieves reasonable performance on *some* encoding for a sufficiently broad range of problems, it is still not clear whether finding suitable encodings is actually less difficult than devising specialised algorithms for the original application domains. Consequently, the crucial problem with the generic problem solving approach is to find an intermediate-domain algorithm and a number of sufficiently efficient and simple encoding strategies which together show a reasonable performance on a broad range of interesting problem domains.

On the other hand, the advantages of the generic solving approach are obvious. First of all, no specialised solver is needed for the original problem domain. Also, many domains benefit from improvements of intermediate-domain solvers. Thus, it pays off to invest significant effort in improving intermediate-domain solvers, and even special hardware support could be an effective way of speeding up these solvers [HM97]. The weak variant of the generic problem solving approach is widely accepted in AI and to date, many problem solving techniques are known which can be successfully applied to a variety of problem domains. The strong variant, however, is less commonly accepted. Nevertheless, recent results for solving SAT encoded problems using stochastic local search techniques give some indication that even the strong variant of generic problem solving might be feasible in practice.

There are several reasons for regarding SAT as a promising intermediate domain. First of all, SAT is $\mathcal{NP}$-complete which means that every $\mathcal{NP}$-complete problem can be polynomially reduced to SAT. Thus, reasonable efficient transformations between SAT and other $\mathcal{NP}$-complete decision problems exist. Since the class of $\mathcal{NP}$-complete decision problems covers a wide range of interesting application problems, SAT is expressive enough for acting as an intermediate domain. The second advantage of SAT is its conceptual simplicity. Because the syntax as well as the semantics of SAT is very simple, especially when restricted to CNF formulae, SAT offers substantial advantages for devising and evaluating algorithms.

So if SAT appears to be a reasonable choice for the intermediate domain, why should one consider SLS algorithms as SAT solvers? First, as mentioned before, SLS algorithms have been found to be competitive with or even superior to systematic search for both native SAT problems as well a broad range of SAT encoded problems from other domains. Furthermore, most popular SLS algorithms for SAT are relatively easy to implement because they are conceptually quite simple. Finally, most of these SLS algorithms can be parallelised in a rather straightforward way. On the other hand, there are some disadvantages of SLS algorithms to be mentioned. First, typical SLS algorithms are incomplete, *i.e.*, they are not guaranteed to find an existing solution. Then, these algorithms are highly non-deterministic which sometimes is seen as a problem by itself (for example, when it comes to the reliability and reproducibility of results). But a more important drawback of this property is the fact that these algorithms have been found extremely difficult to analyse and consequently to date, there is almost no theoretical understanding of their behaviour. Nevertheless, the advantages, especially the superior performance that could be achieved for different types of SAT problems, seem to outweigh these drawbacks such that SLS algorithms are considered to be among the most promising methods for solving large and hard satisfiability problems.

1.4 $\mathcal{NP}$-Complete and $\mathcal{NP}$-Hard Problems

Today, the theory of computational complexity is a well-established field with considerable impact on other areas of computer science. In the context of this work, complexity theory plays a certain role, because the primary field of application of stochastic local search algorithms is a class of computationally very hard problems, for which no efficient, *i.e.*, polynomial time, algorithms are known. Moreover, todate a majority of the experts in complexity theory believe that for principal reasons the existence of efficient algorithms for these problems is impossible.

The complexity of an algorithm is defined on the basis of formal machine models. Usually, these are idealised, yet universal models, designed in a way which facilitates formal reasoning about their behaviour. One of the first, and still maybe the most prominent of these models is the Turing machine. For Turing machines and other formal machine or programming models, computational complexity is defined in terms of the space and time requirements of computions.

Complexity theory usually deals with whole problem classes (generally countable sets of problem instances) instead of single instances. For a given algorithm or machine model, the complexity of a computation is characterised by the functional dependency between the size of an instance and the time and space required to solve this instance. For reasons of analytical tractability, many problems are formulated as decision problems, and time and space complexity are analyzed in terms of the worst asymptotic behaviour. Given a suitable definition of the computational complexity of an algorithm for a specific problem class, the complexity of the problem class itself can be defined as the complexity of the best algorithm on this problem class. Because generally time complexity is the more restrictive factor, problem classes are often categorised into complexity classes with respect to their asymptotic worst case time complexity.

Two particularly interesting complexity classes are $\mathcal{P}$, the class of problems that can be solved by a deterministic machine in polynomial time, and $\mathcal{NP}$, the class of problems which can be solved by a nondeterministic machine in prolynomial time. Of course, every problem in $\mathcal{P}$ is also contained in $\mathcal{NP}$, basically because deterministic calculations can be emulated on a nondeterministic machine. However, the question whether also $\mathcal{NP} \subseteq \mathcal{P}$, and consequently $\mathcal{P} = \mathcal{NP}$, is one of the most prominent open problems in computer science. Since many extremely application relevant fundamental problems are in $\mathcal{NP}$, but possibly not in $\mathcal{P}$ (*i.e.*, no polynomial time deterministic algorithm is known), this so-called $\mathcal{P} = \mathcal{NP}$-Problem is not only of theoretical interest. For these computationally hard problems, the best algorithms known so far have exponential time complexity. Therefore, for growing problem size, the problem instances become quickly intractable, and even the tremendous advances in hardware design have little effect on the size of the problem instances solvable with state-of-the-art technology in reasonable time. Well-known examples of $\mathcal{NP}$-hard problems include the Propositional Satisfiability Problem (see also below), the Graph Colouring Problem, the Knapsack Problem, the Traveling Salesperson Problem, scheduling problems, and time table problems, to name just a few [GJ79].

Many of these hard problems from $\mathcal{NP}$ are closely related to each other and can be translated into each other in polynomial deterministic time (polynomial reduction). A problem, which is at least as hard as any other problem in $\mathcal{NP}$ (in the sense that each problem in $\mathcal{NP}$ can be polynomially reduced to it) is called *$\mathcal{NP}$-hard*. Thus, $\mathcal{NP}$-hard problems in a certain sense can be regarded as at least as hard as every problem in $\mathcal{NP}$. But they do not necessarily have to belong to the class $\mathcal{NP}$ themselves, as their complexity might be actually much higher. $\mathcal{NP}$-hard problems which are contained in $\mathcal{NP}$ are called *$\mathcal{NP}$-complete*; in a certain sense, these problems are the hardest problems in $\mathcal{NP}$.

One fundamental result of complexity theory states that it suffices to find a polynomial time deterministic algorithm for one single $\mathcal{NP}$-complete problem to prove that $\mathcal{NP} = \mathcal{P}$. This is a consequence of the fact, that all $\mathcal{NP}$-complete problems can be encoded into each other in polynomial time. Today, most computer scientists believe that $\mathcal{P} \neq \mathcal{NP}$; however, so far all efforts of finding a proof for this inequality have been unsuccessful and there is some indication that today's mathematical methods might be too weak to solve this fundamental problem.

Many practically revelant problems from application domains, such as scheduling and planning problems, are $\mathcal{NP}$-complete and therefore generally not efficiently solvable today (and maybe, if $\mathcal{NP} \neq \mathcal{P}$, not efficiently solvable at all). However, being $\mathcal{NP}$-complete or $\mathcal{NP}$-hard does not mean for a problem that is impossible to practically solve it efficiently. Practically, there are at least three ways of dealing with these problems:

- Try to find an application relevant subclass of the problem which can be efficiently solved.
- Use efficient approximation algorithms.
- Use stochastic approaches.

Regarding the first strategy, we have to keep in mind that $\mathcal{NP}$-hardness is a property of a whole problem class P, whereas in practice, often only instances from a certain subclass $P' \subseteq P$ occur. And of course, P' does not have to be $\mathcal{NP}$-hard in general, *i.e.*, while for P an efficient algorithm might not exist, it might still be possible to find an efficient algorithm for the subclass P'.

If, however, the problem at hand is an optimisation problem which cannot be narrowed down to an efficiently solvable subclass, another option is to accept approximations of the solution instead of trying to compute optimal solutions. This way, in many cases the computational complexity of the problem can be sufficiently reduced to make the problem practically solvable. In some cases, allowing a comparatively small margin from the optimal solution makes the problem deterministically solvable in polynomial time. In other cases, the approximation problem remains $\mathcal{NP}$-hard, but the reduction in computational effort is sufficient to find acceptable solutions of practically occurring problem instances.

Sometimes, however, even reasonably efficient approximation schemes cannot be devised or the problem is a decision problem, to which the notion of approximation cannot be applied at all. In these cases, one further option is to focus on probabilistic rather than deterministic algorithms. At the first glance, this idea has quite an appeal: After all, according to the definition of the complexity class $\mathcal{NP}$, at least $\mathcal{NP}$-complete problems can be efficiently solved by nondeterministic machines. But this, of course, is of little practical use, since for an actual probabilistic algorithm this means only, that there is a chance, however small, that it can solve the given problem in polynomial time. In practice, the success probability of such an algorithm can be arbitrarily small. Nevertheless, in numerous occasions, probabilistic algorithms have been found to be considerably more efficient on $\mathcal{NP}$-complete or -hard problems than the best deterministic methods available. In other cases, probabilistic methods and deterministic methods complement each other in the sense, that for certain types of problem instances one or the other have been found to be superior. Among the most fundamental and best known problems from this category are the satisfiability problem in propositional logic (SAT) and the constraint satisfaction problem (CSP), which are formally introduced in the next sections.

1.5 Propositional Logic and Satisfiability (SAT)

Propositional logic is based on a formal language over an alphabet comprising propositional variables, truth values and logical operators. Using logical operators, propositional variables and truth values are combined into propositional formulae which represent propositional statements. Formally, the syntax of propositional logic can be defined by:

Definition 1.1 (Syntax of Propositional Logic)

$S = V \cup C \cup O \cup \{(,)\}$ is the *alphabet of propositional logic*, with $V = \{x_i \mid i \in \mathbb{N}\}$ denoting the countable infinite set of *propositional variables*, $C = \{\top, \bot\}$ the set of *truth values* and $O = \{\neg, \wedge, \vee\}$ the set of *propositional operators*.

The set of *propositional formulae* is characterised by the following inductive definition:

- the truth values $\top$ and $\bot$ are propositional formulae;
- each propositional variable $x_i \in V$ is a propositional formula;
- if F is a propositional formula, then $\neg F$ is also a propositional formula;
- if F_1 and F_2 are propositional formulae, then also $(F_1 \wedge F_2)$ and $(F_1 \vee F_2)$ are propositional formulae. □

Assignments are mappings from propositional variables to truth values. Using the standard interpretations of the logical operators on truth values, assigments can be used to evaluate propositional formulae. This way, we can define the semantics of propositional logic:

Definition 1.2 (Semantics of Propositional Logic)

The *variable set* $Var(F)$ *of formula* F is defined as the set of all the variables appearing in F.

A *variable assignment of formula* F is a mapping $B : Var(F) \mapsto C$ of the variable set of F to the truth values. The *complete set of assignments of* F is denoted by $Assign(F)$.

The *value* $Val_B(F)$ *of formula* F *under assignment* B is defined by induction on the syntactic structure of F:

- $F \equiv \top \Longrightarrow Val_B(F) = \top$
- $F \equiv \bot \Longrightarrow Val_B(F) = \bot$
- $F \equiv x_i \in V \Longrightarrow Val_B(F) = B(x_i)$
- $F \equiv \neg F_1 \Longrightarrow Val_B(F) = \neg Val_B(F_1)$
- $F \equiv F_1 \wedge F_2 \Longrightarrow Val_B(F) = Val_B(F_1) \wedge Val_B(F_2)$
- $F \equiv F_1 \vee F_2 \Longrightarrow Val_B(F) = Val_B(F_1) \vee Val_B(F_2)$

The truth values $\top$ and $\bot$ are also known as *verum (true)* and *falsum (false)*, resp.; the operators $\neg$ (*negation*), $\wedge$ (*conjunction*), and $\vee$ (*disjunction*) are defined by the following truth tables:

$\neg$	
$\top$	$\bot$
$\bot$	$\top$

$\wedge$	$\top$	$\bot$
$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$

$\vee$	$\top$	$\bot$
$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$

$\square$

Because the variable set of a propositional formula is always finite, the complete set of assignments for a given formula is also finite. More precisely, for a formula containing n variables there are exactly 2^n variable assignments.

Considering the values of a formula under all possible assignments, the fundamental notions of validity and satisfiability can be defined in the following way:

Definition 1.3 (Satisfiability & Validity)

A variable assignment B is a *model of formula* F, exactly if $Val_B(F) = \top$; in this case we say that B *satisfies* F.

A formula F is *valid*, if it is satisfied by all its variable assignments. Valid formulae are also called *tautologies*.

If for a formula F there exists at least one model, F is *satisfiable*. $\square$

Now, we can formally define the Satisfiability Problem (SAT) and the Validity Problem (VAL) for propositional logic.

Definition 1.4 (The SAT and VAL problems)

For the *Satisfiability Problem (SAT) in propositional logic*, the goal is to decide for a given formula F, whether or not it is satisfiable.

For the *Validity problem (VAL) in propositional logic*, the goal is to decide for a given formula F, whether or not it is valid. $\square$

For SAT, there are two variants of the problem, the decision variant and the model finding variant. In the decision variant, only a yes/no decision for the satisfiability of the given formula is required, while for the model finding variant, in case the formula is satisfiable, a model has to be found. Note that there is a simple relationship between SAT and VAL: For each formula F, F is valid if and only if $\neg F$ is not satisfiable, *i.e.*, a formula is valid exactly if its negation has no model. For that reason, decision procedures for SAT can be used to decide VAL and vice versa. This, however, does not hold for incomplete procedures, such as the SLS algorithms for SAT discussed in this work. Since these cannot determine whether a given formula is *not* satisfiable, they cannot be used for establishing validity.

Often, problems like SAT and VAL are studied for syntactically restricted classes of formulae. Imposing syntactical restrictions usually facilitates theoretical studies and can also be very useful for simplifying algorithm design. Normal forms are syntactically restricted formulae such that for an arbitrary formula F there is always at least one normal form formula F'. Thus, each normal form induces a subclass of propositional formulae which is as expressively powerful as full propositional logic. Because for certain syntactically restricted classes of formulae it is not proven whether or not they are normal forms in that sense, we will use the term "normal form" in a more general sense here, covering reasonably expressively powerful subclasses of propositional logic which are characterised by syntactical restrictions. Some of the most commonly used normal forms are subsumed by the following definition.

Definition 1.5 (Normal forms)

A *literal* is a propositional variable (called a positive literal) or its negation (called a negative literal). Formulae of the syntactic form $c_1 \wedge c_2 \wedge \ldots \wedge c_n$ are called *conjunctions*, while formulae of the form $d_1 \vee d_2 \vee \ldots \vee d_n$ are called disjunctions.

A propositional formula F is in *conjunctive normal form (CNF)*, if it is a conjunction over disjunctions of literals. The disjunctions are called *clauses*. A CNF formula F is in k-CNF, if all clauses of F contain exactly k literals.

A propositional formula F is in *disjunctive normal form (DNF)*, if it is a disjunction over conjunctions of literals. In this case, the conjunctions are called *clauses*. A DNF formula F is in k-DNF, if all clauses of F contain exactly k literals.

A CNF formula F is *Horn*, if every clause in F contains at most one unnegated variable. $\square$

SAT is the prototypical $\mathcal{NP}$-complete problem. Historically, it was the first problem for which $\mathcal{NP}$-completeness was established [Coo71]. $\mathcal{NP}$-completeness of SAT can directly be proven by encoding the calculations of a Turing machine M for an $\mathcal{NP}$ problem into a propositional formula the models of which correspond to the accepting computations of M. Furthermore, it is quite easy to show that SAT is $\mathcal{NP}$-complete when restricted to CNF or even 3-CNF formulae [Rei90]. At the other hand, SAT is decidable in linear time for DNF, for Horn formulae [DG84], and for 2-CNF [Coo71].

As outlined above, VAL is equivalent to the complement of SAT. Therefore, VAL is co-$\mathcal{NP}$-complete. While the same holds for VAL on DNF formulae, VAL is polynomially decidable on CNF formulae. Note that to date, it is unknown, whether co-$\mathcal{NP}$ is different from $\mathcal{NP}$. In an intuitive sense, the VAL problem could actually be harder than SAT because while a solution for SAT can be verified in polynomial (actually linear) time, it is not clear how this should be generally done for a solution of VAL.

1.6 Constraint Satisfaction Problems (CSP)

A *constraint satisfaction problem (CSP)* is a problem defined by a set of variables each of which can take a number of values from some domain and a set of constraining conditions involving one or more variables. Depending whether the variable domains are discrete or continous, finite or infinite, different types of CSPs can be defined. In the context of this work, we restrict our attention mainly to *finite discrete CSPs*. This problem class can formally be defined as follows:

Definition 1.6 (Finite discrete CSP)

A constraint satisfaction problem can be defined as a triple $P = (X, \mathcal{D}, \mathcal{C})$, where $X = \{x_1, \dots, x_n\}$ is a finite set of n variables, $\mathcal{D} = \{D_1, \dots, D_n\}$ a set of domains and $\mathcal{C} = \{C_1, \dots, C_k\}$ a finite set of constraints. Each constraint C_j is a relation $C_j \subseteq X_{j_1} \times \dots \times X_{j_{\#j}}$ on $\#j$ variables.

P is a *finite discrete CSP*, if all the D_i are discrete and finite.

A *variable assignment for* P is a vector $A = (d_1, \dots, d_n)$ containing exactly one value $d_i \in D_i$ for each variable x_i.

If $\mathcal{A}$ denotes the set of all possible assignments for P, then $S \in \mathcal{A}$ is a *solution of* P exactly if it simultaneously satisfies all constraints in $\mathcal{C}$, *i.e.*, if for $S = (s_1, \dots, s_n)$ and all $C_j \in \mathcal{C}$, $C_j \subseteq X_{j_1} \times \dots \times X_{j_{\#j}}$ we have $(s_{j_1}, \dots, s_{j_{\#j}}) \in C_j$. □

Of course, for this type of CSP the set of all possible assignments is finite. As SAT, the class of finite discrete CSPs is $\mathcal{NP}$-complete. This can be proven quite easily as there is a close relation between propositional SAT and finite discrete CSPs: SAT on CNF formulae corresponds directly to a finite discrete CSP where all the domains contain only the boolean values $\top, \bot$ and each constraint contains exactly all the satisfying assignments of one particular clause. Vice versa, each finite discrete CSP can be directly transformed into a SAT instance, where each propositional variable represents a particular assignment of one CSP variable and each constraint is represented by a number of clauses. Note, however, that this type of direct encoding does generally not yield a CNF formula, but a formula in CDNF (conjunctive disjunctive normal form, *i.e.*, a conjunction over disjunctions over conjunctions of literals) which can then be transformed into CNF.

Although CSP appears to be more adequate for the representation of hard combinatorial problems than SAT, SAT is a syntactically considerably simpler domain, and therefore designing and analysing algorithms tends to be easier for SAT than for CSP. At the same time, most SLS algorithms for SAT can be generalised to CSP in a straightforward way, and techniques which are working well for SAT tend to show good performance on CSP as well. Consequently, concepts and algorithms for SAT solving are in many respects more advanced than the corresponding CSP development, as can be seen in the recent literature. Mainly for this reason, we focus on SAT instead of CSP in the context of this thesis. Nevertheless, CSP is an important and interesting problem class; and, as we will see later, it is particularly useful as an intermediate step when transforming problems from other domains into SAT.

1.7 Related Work

Due to the introductory nature of this chapter, there is a huge body of literature related to the concepts presented here. Introductions to AI problems and search methods can be found in any modern or classis textbook on AI (such as [PMG98, RN95, Gin93, Nil80], *etc.*); for details on heuristic search, see also [Pea84]. For reference purposes, the Encyclopedia of Artificial Intelligence [Sha92] provides information on most AI concepts and approaches. An introduction to automated deduction and theorem proving can be found, *e.g.*, in [Bib92]. The generic problem solving approach is folklore in AI as well as in computer science in general; it is, however, tightly related to general issues of knowledge representation (see see [Bib93] for further references). For a detailed discussion of complexity theory, $\mathcal{NP}$-completeness, and $\mathcal{NP}$-hard problems, see [GJ79] or [Rei90]; Constrained Satisfaction problems and solving techniques are described in [Mac92]. Regarding stochastic local search methods for SAT and CSP, seminal studies are [SLM92], [Gu92], and [MJPL90]. Most of the literature on SLS algorithms for AI problems can be found in journals or the proceedings of the major AI conferences, as mentioned in the Prologue; the appropriate references are given in throughout this work, especially in the related work sections of the succeeding chapters.

1.8 Conclusions

In this chapter, we gave a brief introduction to AI problems. We distinguished various types of problems, representational versus computational, and decision versus optimisation problems. As we have argued, many important problems in AI are formalised in the context of logics, and can be interpreted as search problems. We introduced and briefly discussed two algorithmic approaches for solving these problems, systematic and local search algorithms, and reported some of the successes which could recently be achieved using stochastic local search techniques.

We then outlined a generic approach for solving hard problems by transforming them into an intermediate domain and solving them using stochastic local search methods. This technique has recently been successfully applied to some problem domains, yet, as of this writing, it is not clear whether these results can be achieved for a larger number of interesting problems. However, the overall approach of solving problems by mapping them onto alternate representations is widely used in computer science in general, and AI in particular. The propositional satisfiability problem (SAT) seems to be a good candidate for an intermediate domain, since it has sufficient expressive power for covering a large class of interesting problems. At the same time, even when compared to tightly related problems like CSP, SAT has the advantage of being syntactically and conceptionally very simple — which facilitates the development and analysis of algorithms.

Like many AI problems, SAT is $\mathcal{NP}$-hard, thus there is little hope for finding algorithms with better than exponential worst-case behaviour. However, there is still the chance that interesting or application relevant subclasses of these problems are algorithmically tractable. But as we argued, even if for a particular problem this is not the case, transforming problems into SAT and solving them using general SAT algorithms could still be advantageous, *e.g.*, because improving these algorithms pays off for a potentially large number of domains.

Certainly, the concept of generic problem solving by transformation into and solving within an intermediate domain is very elegant. The practical viability of this approach is suggested by the recent impressive successes which are mainly based on the dramatic improvements in SLS techniques for solving hard SAT instances. Thus, within the AI community there is a strong and continuously growing interest in topics like the design and analysis of modern SAT algorithms, SAT encodings, and their applications. These issues are investigated in the remainder of this thesis; while the question, whether generic problem solving using SAT encodings and SLS algorithms for SAT is a viable solution strategy for a large number of hard application problems, remains open, we are convinced that the results presented in the subsequent chapters can be regarded as important contributions to the analysis, improvement, and understanding of the overall approach.

Chapter 2

Empirical Analysis of Las Vegas Algorithms

In this chapter, we introduce a novel methodology for empirically investigating the behaviour of stochastic local search algorithms. We use this methodology later for the empirical analysis of SLS-Algorithms for SAT; here, it is developed in the more general context of Las Vegas Algorithms. After motivating the need for a more adequate empirical methodology and providing some general background on Las Vegas Algorithms, we introduce our new method which is based on characterising run-time distributions of algorithms on single problem instances. As we will show, our novel approach facilitates the evaluation and comparison of Las Vegas Algorithms; specifically, it can be used for obtaining optimal parameterisations and parallelisations. We then point out some pitfalls which arise by using inadequate empirical methodology and show how these are avoided when using our approach. Finally, we extend our methodology to optimisation problems. The chapter concludes with an overview of related work and a short resume of the main ideas and results.

2.1 Motivation and Background

Las Vegas Algorithms (LVAs) are nondeterministic algorithms for which, if a solution is found, its correctness is guaranteed. However, it is not guaranteed that for a soluble problem instance, such an algorithm eventually finds a solution. Because of its nondeterministic nature, the run-time of a Las Vegas Algorithm is a random variable.

Las Vegas Algorithms are prominent not only in the field of Artificial Intelligence but also in other areas of computer science and Operations Research. Because of their inherent randomness, stochastic local search (SLS) algorithms (*cf.* Chapter 1) are particular instances of LVAs. In the recent years, SLS algorithms have become quite prominent for solving both NP-complete decision problems and NP-hard combinatorial problems. These algorithms, such as Tabu Search [Glo89, Glo90, HJ90], WalkSAT [SKC94], the Min-Conflicts Heuristic [MJPL92], Simulated Annealing [KJV83], Genetic Algorithms [Hol75, Gol89], Evolution Strategies [Rec73, Sch81], Ant Colony Optimisation algorithms [DMC96], *etc.*, have been found to be very successful on numerous problems from a broad range of domains. But also a number of systematic search methods, like some modern variants of the Davis Putnam algorithm for propositional satisfiability (SAT) problems, make use of non-deterministic decisions (like randomised tie-breaking rules) and can thus be characterised as Las Vegas Algorithms.

However, due to their non-deterministic nature, the behaviour of Las Vegas Algorithms is usually difficult to analyse. Even in the cases where theoretical results do exist, their practical applicability is often very limited. This is, for example, the case for Simulated Annealing, which is proven to converge towards an optimal solution under certain conditions which, however, cannot be met in practice. On the other hand, theoretical results for algorithms which could be shown to be very effective in practice are usually very limited, as is the case for some of the most successful variants of tabu search. Given this situation, in most cases analyses of the run-time behaviour of Las Vegas Algorithms are based on empirical methodology. In a sense, despite dealing with algorithms which are completely known and can be easily understood on a step-by-step execution basis, computer scientists are in a sense in the same situation here as, say, an experimental physicist observing some non-deterministic quantum phenomenon or a microbiologist investigating bacterial growth phenomena.

The methods that have been applied for the analysis of Las Vegas (and particularly SLS) algorithms in AI, however, are rather simplistic. In most cases, the performance of the algorithms being tested is characterised by measuring basic statistics over a number of runs on a given problem instance, like the mean, median and/or standard deviation of the run-time. Run-times are often reported as CPU-times or, sometimes, using basic step counts for a particular algorithm scheme [AO96], rarely a combination of both. Quite often, the algorithms are evaluated on a set of problem instances drawn from some randomised problem distribution, like Random-3-SAT [MSL92, CA96], randomly generated constraint satisfaction problems (CSPs) [Smi94], or randomly generated graph colouring problems [MJPL92, HW94b]. Quite frequently, when comparing algorithms on such problem classes, the sets are generated using a given generator algorithm instead of being fixed, widely used test sets.

At the first glance, all these methods seem to be quite admissible, especially since the results are usually consistent in a certain sense. In the case of SLS algorithms for SAT, for instance, advanced algorithms like WalkSAT [SKC94] usually outperform older algorithms (such as GSAT [SLM92]) on a large number of problems from both randomised distributions and structured domains. The claims which are supported by empirical evidence are usually quite simple (like "algorithm A outperforms algorithm B"), and the basic methodology, as described above, is both easy to apply and powerful enough to get the desired results.

Or is it really? Recently, there has been some severe criticism regarding the empirical testing of algorithms [Hoo94, Hoo96a, McG96]. It has been pointed out that the empirical methodology that is used to evaluate and compare algorithms does not reflect the standards which have been established in other empirical sciences for quite a time. Also, it was argued that the empirical analysis of algorithms should not remain at the stage of collecting data, but should rather attempt to formulate hypotheses based on this data which, in turn, can be experimentally verified or refuted. Up to now, most work dealing with the empirical analysis of Las Vegas algorithms in AI has not lived up to these demands. Instead, recent studies still use basically the same methods that have been around for years, often investing tremendous computational effort in doing large scale experiments [PW96] in order to ensure that the basic descriptive statistics thus collected are sufficiently stable. At the same time more fundamental issues, such as the question whether the particular type of statistical analysis that is done (usually estimating means and standard deviations) is adequate for the type of evaluation that is intended, are often somewhat neglected or not addressed at all. Therefore, a more adequate methodology for the empirical analysis of Las Vegas algorithms is needed as a basis for application, investigation, and further development.

2.1.1 Las Vegas Algorithms

Formally, an algorithm A is a *Las Vegas algorithm (LVA)* if it has the following properties:

- If for a given problem instance π, algorithm A returns a solution s, s is guaranteed to be a valid solution of π.
- On each given instance π, the run-time of A is a random variable RT_A.

According to this definition, Las Vegas algorithms are always correct, while they are not necessarily complete, *i.e.*, even if a given problem instance has a solution, a Las Vegas algorithm is generally not guaranteed to find it. Since completeness is an important theoretical concept for the study of algorithms, we classify Las Vegas algorithms into the following three catogories:

- *complete Las Vegas algorithms* are those, which can be guaranteed to solve each problem instance for which a solution exists within run-time t_{max}, where t_{max} is an instance-dependent constant. Let $P_s(RT_{A,\pi} \leq t)$ denote the probability that A finds a solution for a soluble instance π in time $\leq t$, then A is complete exactly if for each π there exists some t_{max} such that $P_s(RT_{A,\pi} \leq t_{max}) = 1$.
- *approximately complete Las Vegas algorithms* solve each problem instance for which a solution exists with a probability converging to 1 as the run-time approaches ∞. Thus, A is approximately complete, if for each soluble instance π, $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t) = 1$.
- *essentially incomplete Las Vegas algorithms* are Las Vegas algorithms which are not approximately complete (and therefore also not complete), *i.e.*, for which there exists a soluble instance π, for which $\lim_{t\to\infty} P_s(RT_{A,\pi} \leq t) < 1$.

Examples for complete Las Vegas algorithms in AI are randomised systematic search methods like Satz-Rand [GSK98]. Many of the most prominent stochastic local search methods, like Simulated Annealing or GSAT with Random Walk, are approximately complete, while others, such as basic GSAT, the Min-Conflicts Heuristic, and most variants of Tabu Search are essentially incomplete.

In literature, approximate completeness is often also referred to as *convergence*. Convergence results are established for a number of SLS algorithms, such as Simulated Annealing [KJV83], or Genetic Algorithms [Bäc94]; another class of complete Las Vegas algorithms which is less well known in AI, are certain learning automata schemes [NT89]. Interestingly, completeness can be enforced for most SLS algorithms by providing a restart mechanism, as can be found in GSAT [SLM92], and selecting the starting configurations in a systematic way. However, both forms of completeness are mainly of theoretical interests, since the time limits for finding solutions are far too large to be of practical use. For SLS algorithms, essential incompleteness is usually caused by the algorithm getting trapped in local minima of the search space they are operating on. Modern SLS algorithms provide methods, like restart, random walk, probabilistic tabu-lists, etc., to escape from these local minima and thus achieve approximate completeness.

Since most relevant Las Vegas algorithms are either complete or approximately complete, or can at least be easily modified to achieve (approximate) completeness as indicated above, we assume for the following that the Las Vegas algorithms we are dealing with are at least approximately complete unless explicitly mentioned otherwise. Consequently, we know that each run of the algorithm terminates after some time, or, in other words, the algorithm never gets stuck.

2.1.2 Application Scenarios

Before even starting to evaluate any algorithm, it is crucial to find the right evaluation criteria. Especially for Las Vegas algorithms, since their run-time is non-deterministic, there are fundamentally different criteria for evaluating their run-time behaviour, depending on the characteristics of the environment they are supposed to work in. Thus, we classify possible application scenarios in the following way:

Type 1: There are no time limits, *i.e.*, we can afford to run the algorithm as long as it needs to find a solution. Basically, this scenario is given whenever the computations are done offline or in a non-realtime environment where it does not really matter how long we need to find a solution. In this situation we are interested in the expected run-time of the algorithm which can be estimated from averaging over a number of test runs.

Type 2: There is a time limit for finding the solution such that the algorithm has to provide a solution after a time of t_{max}, while solutions which are found later are of no use. In real-time applications, like robotic control or dynamic scheduling tasks, t_{max} can be very small. In this situation we are not so much interested in the expected time for finding a solution but in the probability that after a time of t_{max} a solution has been found.

Type 3 The usefulness or utility of a solution depends on the time which was needed to find it. Formally, if utilities are represented as values in $[0, 1]$, we can characterise these scenarios by specifying a utility function $U : \mathbb{R}^+ \mapsto [0, 1]$, where $U(t)$ is the utility of finding a solution after time t. As can be easily seen, types 1 and 2 are special cases of type 3 which can be characterised by utility functions which are either constant (type 1) or step functions $U(t) = 1$ for $t \leq t_{max}$ and $U(t) = 0$ for $t > t_{max}$ (type 2).

Based on these application types, we now discuss the criteria for evaluating the performance of Las Vegas algorithms in each scenario, respectively. While in the case of no time limits being given (type 1), the mean run-time might suffice to roughly characterise the run-time behaviour, in real-time situations (type 2) it is basically meaningless. An adequate criterion for type 2 situation with time-limit t_{max} is $P(RT \leq t_{max})$, the probability of finding a solution within the given time-limit. For type 3, the most general scenario, the run-time behaviour can only be adequately characterised by the run-time distribution function $rtd : \mathbb{R} \mapsto [0, 1]$ defined as $rtd(t) = P(RT \leq t)$ or some approximation of it. The run-time distribution (RTD) specified by this distribution function completely and uniquely characterises the run-time behaviour of a Las Vegas algorithm. Therefore, given this information, other criteria, like the mean run-time, its standard deviation, median, percentiles, or success-probabilities $P(RT \leq t')$ for arbitrary time-limits t' can be easily obtained.

In some sense, type 3 is not only the most general class of application scenarios, but these scenarios are also the most realistic. The reason for this is the fact that real-world problem solving usually involves time-constraints which are less strict than in type 2. Instead, at least within a certain interval, the value of a solution gradually decreases over time. In particular, this situation is given when taking into account the costs (like CPU time) for finding a solution. As an example, consider a situation where hard combinatorial problems have to be solved online, using extremely expensive hardware in a time-sharing mode. Even if the immediate benefit of finding a solution is invariant over time, the costs for performing the computations will diminish the final payoff. Two common ways of modelling this effect are constant or proportional discounting, *i.e.*, to use utility functions of the form $U(t) = \max\{u_0 - ct, 1\}$ or $U(t) = e^{-\lambda t}$, respectively [PMG98]. Based on the utility function, the weighted solution probability $U(t) \cdot P(RT \leq t)$ can be used as a performance criterion. If $U(t)$ and $P(RT \leq t)$ are given, optimal cutoff times t^* which maximise the weighted solution probability can be determined as well as the expected utility for a given time t'. These evaluations and calculations require detailed knowledge of the underlying run-time distributions.

2.2 RTDs and RLDs

As we have argued in the previous section, characterising their run-time distributions is a good basis for empirically studying the behaviour of LVAs. In practice, the empirical RTDs are determined by running the respective algorithm A for k times on a given problem instance up to some (very high) cutoff time and recording for each successful

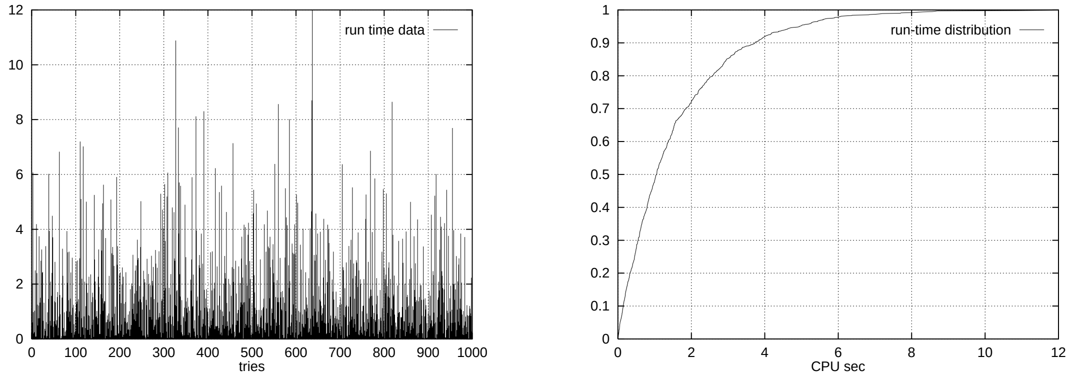


Figure 2.1: Run-time data of WalkSAT, applied to a hard Random-3-SAT instance for approx. optimal noise setting, 1,000 tries; *left:* bar diagramm of $rt(i)$, the vertical axis indicates the CPU-time; *right:* corresponding RTD; the run-times are measured in CPU-seconds on a Sun SPARC 20 machine.

run the time required to find a solution. For the empirical studies presented in this work, we chose k between 200 and 1,000 in order to get reasonable estimates. The empirical run-time distribution is the cumulative distribution associated with these observations. More formally, let $RT(j)$ denote the run-time for the jth successful run, the cumulative empirical RTD is defined by $\widehat{P}(RT \leq t) = \#\{j|RT(j) \leq t\}/k$. For essentially incomplete algorithms, we report the success-rate k'/k, where k' is the number of successful runs. For sufficiently high run-times, this approximates the asymptotic maximal success probability of A on the given problem instance π, which is formally defined as $\widehat{p_s}^* := \lim_{t\to\infty} P_s(RT_{A,\pi} \leq t)$. Furthermore, we filter out the unsuccessful runs and characterise the behaviour of A in the successful cases using the conditional empirical RTD defined by $\widehat{P}_c(RT \leq t) = \#\{j| \text{ run } j \text{ successful } \wedge RT(j) \leq t\}/k'$.

Figure 2.1 (left) shows the raw data from running a state-of-the-art SLS algorithm for SAT on a hard problem instance with 100 variables; each vertical line represents one try of the algorithm and the height of the lines corresponds to the CPU time needed. The right side of the same figure shows the corresponding RTD as a cumulative probability distribution curve $(t, \widehat{P}_c(RT \leq t))$; note that an extreme variability in run-time can be observed. We will see later that this is typical for SLS algorithms when applied to hard combinatorial problems like SAT.

2.2.1 CPU-times *vs* Operation Counts

Often, run-times are measured and reported as CPU-time for some concrete implementation and run-time environment (machine and operating system). But it has been argued that it is more advisable, especially in the context of comparative studies, to measure run-times using operation counts [AO96]. This way, an abstraction from the influence of the implementation and run-time environment is achieved, which facilitates comparing empirical

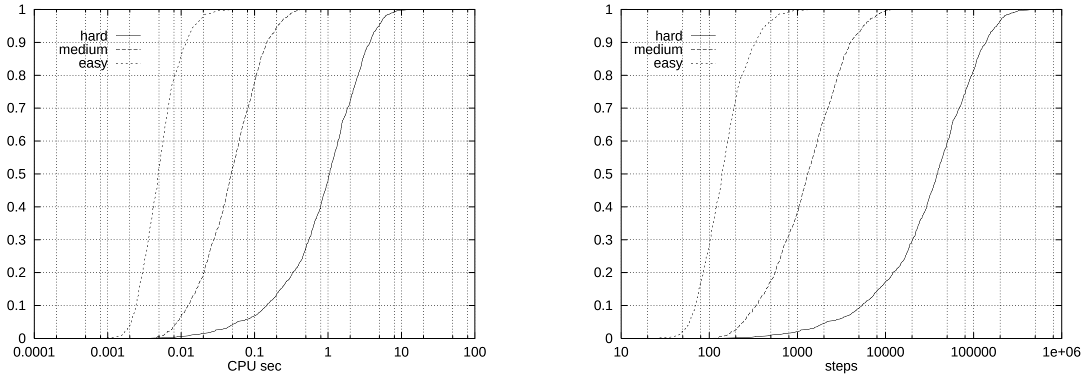


Figure 2.2: RTDs *(left)* and RLDs *(right)* for WalkSAT, applied to three Random-3-SAT instances with varying difficulty, using approx. optimal noise setting, 1,000 tries.

results across various platforms. At the same time, by reporting typical CPU-times for the relevant operations, concrete run-times can be easily calculated for a given implementation and run-time environment.

To make a clear distinction between actual CPU-times and abstract run-times measured in operation counts by referring to the latter as *run-lengths*. If the CPU-time per operation is roughly constant (as is the case for most SLS algorithms), run-times measured in CPU-time and run-lengths measured in basic operations are related to each other by scaling with a constant factor. Otherwise, to obtain a reasonable characterisation of the run-time behaviour, a more complex cost-model has to be used for weighting the different types of operations (or steps) which comprise the run-length. In the case of SLS algorithms for SAT and CSP, it is very natural and quite common to use local search steps as basic operations. Thus, run-times are measured in terms of local search steps instead of absolute CPU-times. In the following, the RTDs which are thus obtained will be called RLDs (run-length distributions).

Figure 2.2 shows RTD and RLD data for the same experiments (solving hard SAT instances with a state-of-the-art SLS algorithm). Note that, when comparing the RTDs and the corresponding RLDs in a semi-log plot, both distributions always have the same shape. This indicates that the CPU-time per step is roughly constant. However, closer examination of the example reveals that the CPU-time per step is not constant for the three instances; the reason for this is the fact that the hard problem was solved on a faster machine than the medium and easy instances. Using RLDs instead of RTDs abstracts from these machine specific aspects of experimental results.

2.2.2 Empirical Evaluation Based on RLDs

When analysing or comparing the behaviour of Las Vegas algorithms, the empirical RLD (or RTD) data can be used in different ways. In our experience, graphic representations

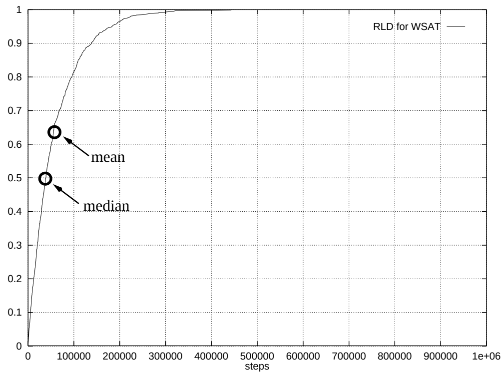


Figure 2.3: RLD for WalkSAT on a hard Random-3-SAT instance for approx. optimal noise parameter setting; median vs. mean.

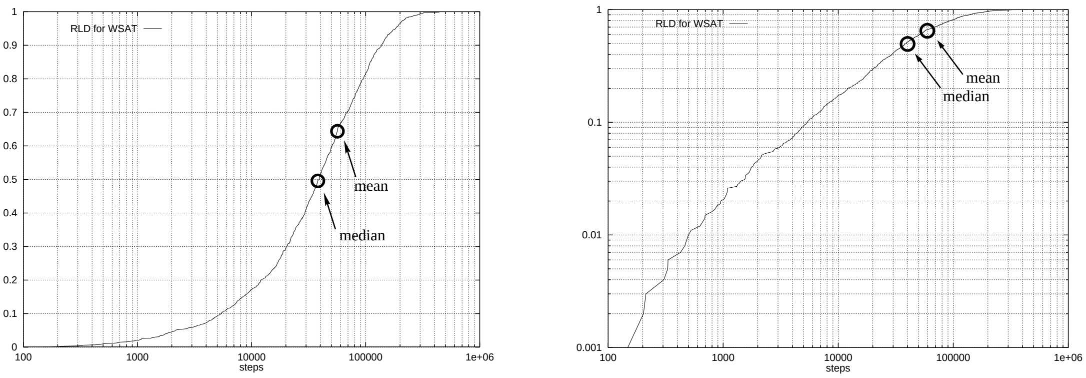


Figure 2.4: Same RLD data as above as a semi-log *(left)* and log-log *(right)* plot.

mean	stddev	stddev/mean	min	max	median
57,606.23	58,953.60	1.02	107	443,496	38,911

$Q_{0.25}$	$Q_{0.75}$	$Q_{0.1}$	$Q_{0.9}$	$Q_{0.75}/Q_{0.25}$	$Q_{0.9}/Q_{0.1}$
16,762	80,709	5,332	137,863	4.81	25.86

Table 2.1: Basic descriptive statistics for the RLD given in Figures 2.3 and 2.4; Q_x is the x-percentile, while the percentile ratios Q_x/Q_{1-x} are measures for the variability of the run-length data.

of empirical RLDs provide often a good starting point. As an example, Figures 2.3 and 2.4 show the RLD for the hard problem instance from Figure 2.2 in three different views. Compared to standard representations, semi-log plots (as shown in Figure 2.4, left), give a better view of the distribution over its whole range; this is especially important, since SLS algorithms show often RLDs with extreme variability. Also, when using semi-log plots to compare RLDs, uniform performance differences characterised by a constanct factor can be easily detected, as they correspond to a simple translation along the horizontal axis (for an example, see Figure 4.12, page 92, the curves for GWSAT and WalkSAT). On the other hand, when examining the behaviour of an algorithm for extremely short runs, log-log plots (as shown in Figure 2.4, right) are very useful.

While these graphical representations of RLDs are well suited for investigating and describing the qualitative behaviour of Las Vegas algorithms, quantitative analyses are usually based on the basic descriptive statistics of the RLD data. For our example, some of the most common standard descriptive statistics, like the empirical mean, standard deviation, minimum, maximum, and some percentiles, are reported in Table 2.1. Note again the huge variability of the data, as indicated by the large standard deviation and percentile ratios. The latter, like the *normalised the standard deviation* (stddev/mean), have the advantage of being invariant to multiplication of the data by a constant, which – as we will see later – is often advantageous when comparing RLDs.

Generally, it should be noted that for directly obtaining sufficiently stable estimates for statistics, basically the same number of test-runs have to be performed then for measuring reasonable empirical RLDs. Thus, measuring RLDs does not cause a computational overhead in data aquisition when compared to measuring only averages and empirical standard deviations. At the same time, arbitrary percentiles and other descriptive statistics can be easily calculated from the RLD data.

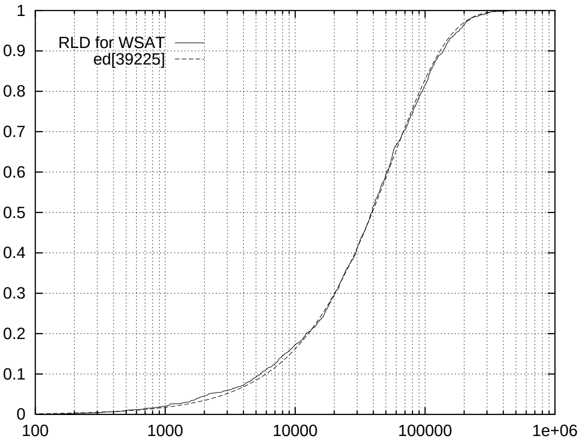


Figure 2.5: Same RLD data as in Figure 2.3 and best-fit approximation by an exponential distribution.

Functional Characterisation of RLDs

But of course, an RLD-based approach can go considerably beyond an empirical evaluation on the basis of basic descriptive statistics. One of the particularly attractive methodological approaches is the direct characterisation of run-time distributions using functional approximations. In this work, we make strong use of continuous distribution functions and statistical goodness-of-fit tests for characterising the behaviour of Las Vegas algorithms like stochastic local search for SAT and CSP. The main reason for using continuous distribution functions for characterising an essentially discrete behaviour, especially in the context of RLDs, is that these are mathematically easier to handle: by abstracting from the discrete nature of the RLDs, a more uniform characterisation is facilitated.

Coming back to the example from Figure 2.4, one might notice that the RLD resembles an exponential distribution. This leads to the hypothesis that on the given problem instance, the algorithm's behaviour can be characterised by an exponential RLD. The validity of this hypothesis can be tested using the χ^2-test, a goodness-of-fit test well-known from statistical literature. In our example, we first fit the RLD data with a cumulative exponential distribution function of the form $ed[m](x) = 1 - exp(x/m)$, using the Marquart-Levenberg algorithm (as realised in C. Gramme's Gnufit software) to determine the optimal value for the parameter m. This approximation is depicted in Figure 2.5. Then, the goodness-of-fit test gives a χ^2 value of 26.24, which means that the approximation passed the test at a standard significance level $\alpha = 0.05$. Therefore, our distribution hypothesis has been confirmed for the given example.

Generally, the actual run-time distribution of a given Las Vegas algorithm depends on the problem instance. Thus, it should be clear that, in the first place, RLDs should be estimated and characterised on single problem instances. Often, however, one is interested in

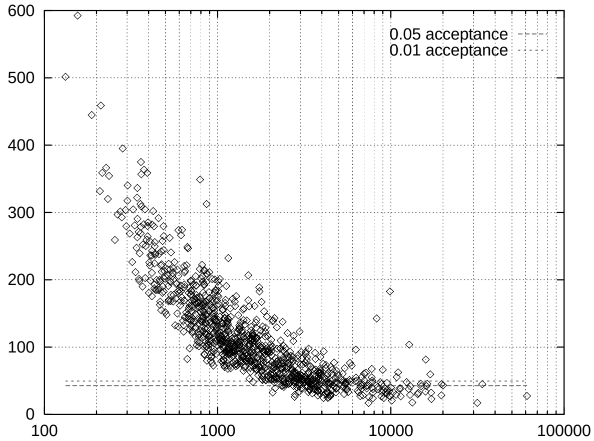


Figure 2.6: Correlation between hardness of problem instances and χ^2 values from testing RLDs of individual instances versus a best-fit exponential distribution for a test-set of 1,000 Random-3-SAT instances with 100 variables and 430 clauses each. The horizontal lines indicate the acceptance thresholds for the 0.01 and 0.05 acceptance levels of the χ^2-test.

an algorithm's performance on a family or distribution of problem instances. The more general type of analysis required in this situation can be easily based on the methodology used for single problem instances. We do this by formulating hypotheses on the type of RLDs which can be observed when applying a Las Vegas algorithm to sets or distributions of problem instances. These hypotheses can be empirically tested by estimating the parameters of the respective generic distribution functions for individual instances and using statistical goodness-of-fit-tests, like the well-known χ^2-test [Roh76] to ensure the validity of the corresponding functional approximations.

Following this approach, it is also possible to characterise the differences between problem instances within the same distribution, or to analyse and adequately describe the scaling of LVA behaviour dependent on problem size. As we will see, such characterisations have very direct consequences for important issues like parallelisation or optimal parameterisation of Las Vegas algorithms. At the same time, they suggest novel interpretations of LVA behaviour which, as in the case of stochastic local search in this work, contributes do improving our understanding of these algorithms. Thus, by using our RLD-based methodology, an experimental approach of testing Las Vegas algorithms along the lines proposed by Hooker [Hoo96a] can be undertaken.

As an example for this kind of methodology, we extend our example from Figure 2.5. Before, we verified the hypothesis that for the given problem instance, the performance of WalkSAT, a modern SLS algorithm for SAT, could be charactised by an exponential RLD. Now, we generalise this hypothesis by assuming, that for a whole problem distribution, WalkSAT's behaviour can be characterised by exponential run-time distributions. Concretely, we test

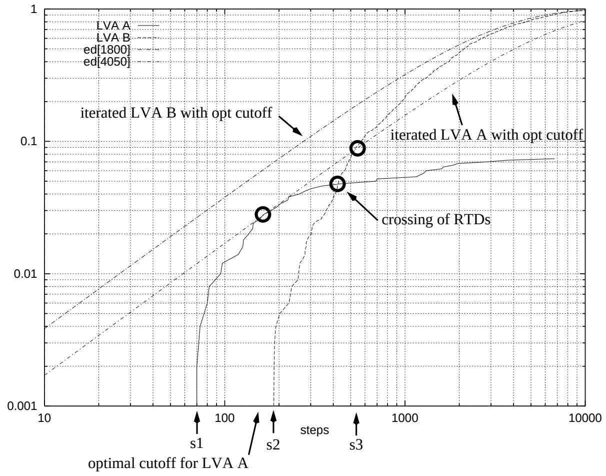


Figure 2.7: RLDs for two SLS algorithms (LVA A and LVA B) for the propositional satisfiability problem on a hard Random-3-SAT instance (the two RLDs cross over at ca. 420 steps), see text for a detailed discussion.

this hypothesis for a test-set of Random-3-SAT instances with 100 variables and 430 clauses.[1] Fitting the data for the individual instances with exponential distributions and calculating the χ^2 as outlined above, we get the result shown in Figure 2.6, where we plot the median values of the RLDs against the corresponding χ^2 values: although, for most instances, the distribution hypothesis is rejected, we observe a clear correlation between the hardness of the instances and the χ^2 values; and for almost all of the hardest instances, the distribution hypothesis passes the test. Thus, although our original generalised hypothesis could not be confirmed, it could be easily modified to match the results of our analysis.[2]

Comparing LVA performance based on RLDs

RLDs also provide a good basis for comparing the behaviour and performance of Las Vegas algorithms. Comparing two Las Vegas algorithms is relatively straight-forward if a dominance relation holds between them, *i.e.*, one of them consistently gives a higher solution probability than the other. Formally, this can be captured by the concept of domination, defined in the following way: LVA A dominates LVA B if $\forall t : P(RT_A \leq t) \geq P(RT_B \leq t)$ and $\exists t : P(RT_A \leq t) > P(RT_B \leq t)$. In practice, such a dominance relation cannot always be observed, which makes the comparison substantially more difficult. This situation is characterised by the occurrence of cross-overs between the corresponding RLDs, indicating,

[1] Random-3-SAT is a syntactically restricted subclass of SAT; the distribution we use here corresponds to the phase transition for Random-3-SAT, a region where particularly difficult problem instances can be found. Details on this will be given in Chapter 4.

[2] The result indicated here will be further investigated in Chapter 4.

that which of the two algorithms gives better performance, *i.e.*, higher solution probabilities, depends on the time-limit.

As an example for this, consider the situation presented in Figure 2.7 in which RLDs for two specific SLS algorithms (LVA A and LVA B) are plotted. LVA B is approximately complete w.r.t. the given problem instance and its performance monotonically improves with increasing run-length (as can be seen from the decreasing distance between the RLD and the projected optimal exponential distribution $ed[1800]$). LVA A is essentially incomplete, the success probability converges to ca. 0.08. For very short runs we also observe "convergent behavior": Apparently for both algorithms there exists a minimal (sample size dependent) number of steps (marked s_1 and s_2 on the x-axis) below which the probability for finding a solution is negligible. Interestingly, both curves cross in one specific point at ca. 420 steps; *i.e.*, without using restarts, LVA A gives a higher solution probability than LVA B for shorter runs, whereas LVA B is more effective for longer runs. Yet, for LVA A an optimal cutoff value of ca. 170 steps exists. Thus, repeated execution of LVA A for a reasonably well chosen cutoff time, after which the algorithm is restarted, gives a much higher solution probability as actually observed when running the algorithm for a long time. In fact, if the optimal cutoff parameter is chosen, one more point of interest is at ca. 500 steps (marked s_3 on the horizontal axis): for a lower number of steps as s_3, using independent runs of LVA A with optimal cutoff value one improves upon the performance of LVA B, while past s_3 LVA B is strictly superior to LVA A. Observations like this can generally be exploited to enhance the overall performance; in particular, any-time behaviour can be achieved by combining suitably parameterised SLS algorithms into algorithm portfolios (*cf.* Chapter 5).

Deriving Optimal Parameterisations from RLD Data

As shown in the example above, run-time distributions can be used for determining optimal parameters for Las Vegas algorithms. In particular, given the RLD data, it is relatively easy to decide whether restarting the algorithm after a fixed cutoff-time will improve performance, and if so, to derive optimal cutoff-times.

To decide wether restart should be used, we simply compare the observed RLD with an exponential distribution. As it is well-known that for exponentially distributed run-times, due to the properties of the exponential distribution [Roh76], we get the same solution probability running an algorithm once for time t or p times for time t/p. Therefore, if the observed RLD is exponential, restart will neither improve nor worsen the algorithms performance. As a consequence, in this case, we can run the individual tries in parallel on different processors and obtain an optimal speedup (this issue will be further discussed in Chapter 5).

If, however, the observed RLD is steeper than an exponential distribution, the probability of finding a solution within a given time interval increases when running the algorithm for a longer time (*cf.* LVA B in Figure 2.7). In this a case, restarting the algorithm after some fixed cutoff time would result in a loss of performance; consequently, when using multiple independent tries parallelisation, the speedup will generally be suboptimal. On the other side, if the run-time distribution of an algorithm is less steep than an exponential distribution, by restarting this algorithm its performance can be improved, as illustrated for algorithm LVA A in Figure 2.7. In this case, an optimal cutoff time RT_{opt} can be identified in the following way:

$$\begin{aligned} m' &= \min\{m \mid \exists x > 0 : ed[m](x) - rt(x) = 0\} && (2.1) \\ RT_{\text{opt}} &= \min\{x \mid x > 0 \wedge ed[m'](x) - rt(x) = 0\} && (2.2) \end{aligned}$$

Generally, there are two special cases to be considered. Firstly, we might not be able to determine m' because the set over which we minimise in the first equation has no minimum. In this case, it can be shown that the optimal cutoff is either equal to zero (if the infimum is equal to zero), or it is equal to $+\infty$, *i.e.*, no restart should be used at all. Secondly, if m' exists it might still not be possible to determine RT_{opt}, because the set in the second equation does not have a minimum. In this case, obviously there are arbitrary small common points of $ed[m']$ and rt, such that the optimal cutoff would be equal to zero. In practice, the cases with an optimal cutoff of zero will hardly occur, since they would only occur for algorithms which can solve a problem with some probability > 0 for arbitrary small run-times. The case with $RT_{\text{opt}} = \infty$ corresponds to the RLD being generally steeper than an exponential distribution.

Thus, if an optimal cutoff exists it can be easily determined from the RLD data. In this case, multiple independent tries parallelisation results in super-optimal speedup compared to the sequential case; this situation is given for many greedy local search algorithms like GSAT which easily get stuck in local minima. Generally, this demonstrates how the RLD-based methodology facilitates the analysis and improvement of Las Vegas algorithms.

2.3 Pitfalls of Inadequate Methodology

While in the last section, we discussed the benefits of an RLD-based empirical methodology for analysing, comparing, and improving the behaviour of Las Vegas algorithms, in this section we point out pitfalls which arise when using inadequate methodology. In particular, we identify two types of problematic methodology and show how in each of these cases, erroneous or inadequate conclusions are obtained. We give theoretical and empirical justifications for our arguments and highlight our criticism based on selected examples.

2.3.1 Problem Type 1: Superficial Analysis of Run-time Behaviour

In many studies, the evaluation or comparison of Las Vegas algorithms is solely based on average run-times, or averages and standard deviations, while nothing is known about the type of their run-time distributions. One problem with this method is that these statistics alone give only a very imprecise impression of the run-time behaviour. Thus, analysing and reporting only means and possibly variances is a very wasteful use of the empirical data when compared to the RLD-based approach which, as detailed before, essentially uses the same data as required for estimating stable means and standard deviations.

Before we demonstrate this, let us briefly recapitulate how the mean run-time of a Las Vegas algorithm is estimated in practice. First, the algorithm is executed n times on a given problem instance with some cutoff time $RT_{\max}$. Then, if k of these runs are successful and RT_i is the run-time of the ith successful run, the expected run-time is estimated by averaging over the successful runs and accounting for the expected number of runs required to find a solution:[3]

$$\widehat{E}(RT) = \frac{1}{k}\sum_{i=1}^{k} RT_i + \frac{n-k}{k} \cdot RT_{\max} \tag{2.3}$$

Now, to see the difference in precision when using means, means plus standard deviations, and functional approximations of the RLDs, consider the design of an algorithm for a type 2 application scenario (as defined in Section 2.1) and the specific question of estimating the cutoff time $RT_{\max}$ for solving a given problem instance with a probability p. If only the mean run-time $E(RT)$ is known, the best estimate we can obtain for $RT_{\max}$ is given by the Markov inequality [Roh76] $P(RT \geq t) \leq E(RT)/t$:

$$RT_{\max} = E(RT)/(1-p) \tag{2.4}$$

If also the standard deviation $\sigma(RT)$ is taken into account, and under the condition that RT has a finite variance, we can use the Tchebichev inequality $P(\mid RT - E(RT) \mid \geq \epsilon) \leq \sigma^2(RT)/\epsilon^2$ to obtain a better estimate:

$$RT_{\max} = \sigma(RT)/\sqrt{1-p} + E(RT) \tag{2.5}$$

If, however, we know that the run-time of a given Las Vegas algorithm is exponentially distributed,[4] we get a much more accurate estimate.

In the following example, we see the drastic differences between these three estimates. For a given Las Vegas algorithm applied to some problem the mean run-time and the standard deviation are 100 seconds each, a situation which is not untypical, *e.g.*, for stochastic local search algorithms for SAT. We are interested in the run-time required for obtaining

[3] This method is equivalent to the one used in [PW96], where a more detailed derivation of this estimate can be found.

[4] As we will discuss later, assuming an exponential RLD is not as far-fetched as it might seem, since we found that these can be observed for a number of modern stochastic local search algorithms on various problem classes.

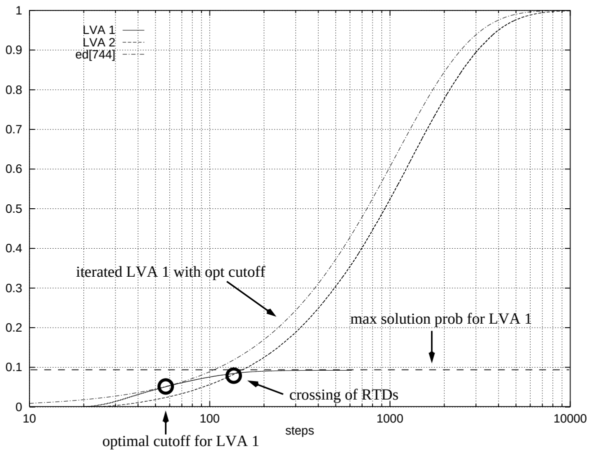


Figure 2.8: RLDs for two different Las Vegas algorithms on the same problem instance. Note the crossover of the two RLDs. By executing independent tries of LVA 1 using the optimal cutoff value, compared to LVA 2 for all run-lengths a speedup by a constant factor is achieved. A detailed discussion is given in the text.

a solution probability of 0.99. If nothing is known about the run-time distribution, we have to use the Tchebichev inequality and get an estimate of 1,100 sec. Assuming that the run-time is exponentially distributed, we get an estimate of 460 sec. If, on the other hand, even the standard deviation is unknown, we can only use the Markov inequality and estimate the run-time required for a solution probability of 0.99 as 10,000 sec! Intuitively, this is hardly surprising – just imagine some wildly different probability distributions having identical mean and standard deviation. The unspecific estimates based on the Markov and Tchebichev inequalities, of course, have to take into account all possible actual RLDs and therefore have to be quite inaccurate when compared to the precise estimates obtained from detailed knowledge of the actual RLD.

Parameter Dependencies

Another problem with numerous studies from recent literature on stochastic local search is the tacit assumption that several parameters of the considered algorithms can be studied independently. In specific cases, it is known that this assumption does not hold [HS96, SSS97]. For the evaluation of Las Vegas algorithms in general, it is crucial to be aware of possible parameter dependencies, especially those involving the cutoff time $RT_{\max}$ which plays an important role in type 2 and 3 application scenarios. By applying an RLD-based methodology, such parameter dependencies can be easier detected and characterised than when using basic descriptive statistics only. The following example shows, how basing the comparison of two Las Vegas algorithm's performance on expected run-times alone can lead to erroneos conclusions, because parameter dependencies are not taken into account.

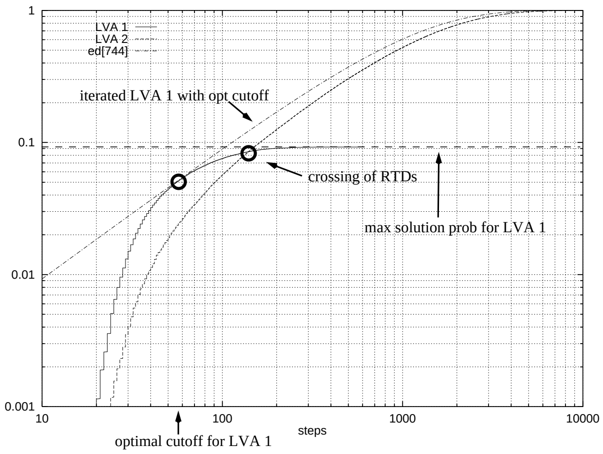


Figure 2.9: Same as Figure 2.8 using log-log representation. Note how for extremely short runs the relative difference in performance between LVA 1 and LVA 2 decreases again, while the performance of both algorithms rapidly deteriorates.

Figure 2.8 shows the RLDs of two different Las Vegas algorithms LVA 1 and LVA 2 for the same problem instance. The same data is shown as a log-log plot in Figure 2.9, giving a better view on the algorithm's behaviour for short runs. As can be easily seen, LVA 1 is essentially incomplete with an asymptotic solution probability approaching ca. 0.09, while LVA 2 is approximately complete. Now note the crossing of the two RLDs at ca. 120 steps. For smaller cutoff times, LVA 1 achieves considerably higher solution probabilities, while for greater cutoff times, LVA 2 shows increasingly superior performance. Consequently, if comparing only estimated run-times as described above, the result entirely depends on the cutoff chosen for conducting the experiments. Thus, when being not aware of the cross-over in the RLDs, there is a considerable risk of misinterpretation, as this comparison will suggest a (non-existing) dominance relation for almost all chosen cutoff-times.

Furthermore, we can easily observe that the RLD of LVA is less steep than an exponential distribution. Therefore, as argued in Section 2.2.2, using restart or multiple independent parallel runs of the algorithm, its performance can be considerably improved. Inn particular, using the optimal cutoff time of ca. 57 steps, ideally, the exponential run-time distribution marked "ed[744]" can be obtained, which realises a speedup of ca. 24% compared to LVA 2. Thus, the performance of LVA 2 is not only superior to that of LVA 1 for small cutoff times, but based on the RLDs, it is possible to modify algorithm LVA 1 such that its overall performance dominates that of LVA 2.

As this example demonstrates, basing the comparison of Las Vegas algorithms on expected run times is in the best case imprecise, in the worst case it leads to erroneous conclusions. The latter case occurs, when the two corresponding RLDs have at least one intersection. Then, obviously, the outcome of comparing the two algorithms depends entirely on the cutoff time $RT_{\max}$ which was chosen for the experiment. While in reality, one of the two algorithms is superior for small $RT_{\max}$ but is outperformed by the other candidate for large $RT_{\max}$, a comparison based on the mean run-times obtained for one specific $RT_{\max}$ value will give the incorrect impression that one algorithm is generally superior to the other. This fallacy arises especially in situations, where one or both of the candidate algorithms are not approximately complete. Then, while one algorithm might quickly converge to a lower asymptotic success probability, the other might be less effective in the initial phase of the search, but later on reach a higher asymptotic success probability. As we will show in Chapter 4, this situation can be often observed in practice.

2.3.2 Problem Type 2: Inhomogeneous Test-sets

Often, Las Vegas algorithms are tested on sets of randomly generated problem instances. For analysing a Las Vegas algorithm's behaviour on such a test-set, on each instance, a number of runs is performed and evaluated. Then, the final performance measure is usually obtained by averaging over all instances from the test set.

This last step, however, is potentially extremely problematic. Since the run-time behaviour on each single instance is characterised by an RLD (as discussed above), averaging over the test set is equivalent to averaging over these RLDs. Since in general, averaging over a number of distributions yields a distribution of a different type, the practice of averaging over RLDs (and thus the averaging over test sets) is quite prone to producing observations which do not reflect the behaviour of the algorithm on the individual instances, but rather a side-effect of this method of evaluation.

We exemplify this for Random-3-SAT near the phase-transition, a problem class which has been used in many studies of stochastic local SAT procedures, such as GSAT or GSAT with random walk [SLM92, GW95] (the problem class itself will be discussed in more detail in Chapter 4). Practically, Random-3-SAT test-sets are obtained by generating a number of sample instances and, as typical SLS algorithms for SAT are incomplete, filtering out insoluble instances using a complete SAT algorithm. By measuring the median run-lengths for each problem instance from a typical randomly generated and filtered test set, we obtain a distribution over the median hardness of the problems as shown in Figure 2.10. Since the median run-lengths were determined using 1,000 runs per instance, they are very stable which leaves the random selection of the instances as the main source of noise in the measured distribution.

Our own experimental studies have shown that some of the most popular SLS algorithms for SAT, such as WalkSAT [SKC94] or Novelty [MSK97], show approximately exponential RLDs. Specifically, under certain conditions, the RLDs on single instances from Random-3-SAT instance distribution can be well approximated by exponential distributions $ed[m]$, where m is the number of local search steps required to solve the problem with probability 0.5, *i.e.*, the median of the RLD; these results will be presented in detail in Chapter 5 of this work.

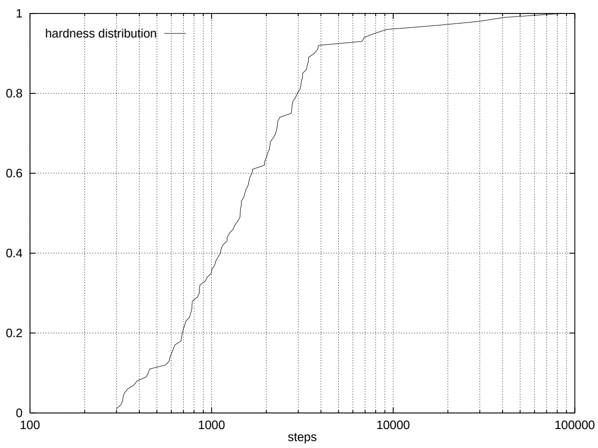


Figure 2.10: Distribution of median run-length for WalkSAT (noise=55/100) over a test set of 100 satisfiable Random-3-SAT formualae (unforced). Note the huge variance, especially for the hardest 10% of the test-set.

Now we assume that we average over two exponential RLDs with different median values m_1, m_2. Since the family of exponential distributions is not closed under averaging, this produces a combined distribution $d_{1,2}$ which is *not* exponentially distributed. For this combined distribution, shown in Figure 2.11, obviously an optimal cutoff time exists, which can be determined as detailed in Section 2.2.2. For the single RLDs, however, since they are exponential, optimal cutoffs does *not* exist (*cf.* Section 2.2.2). Thus, when averaging over RLDs, observations (here: existence of an optimal cutoff) can be made, which do not apply for *any* of the individual instances but are rather an effect of the averaging itself.

At the first glance, this might sound a bit paradoxical, but there is a perfect interpretation for this observation: when averaging over the RLDs, we do not distinguish between the probability of solving one or the other instance. Using the "optimal" cutoff inferred from the average RLD then simply means that solving the easier instance with a sufficiently high probability compensates for the very small solution probability for the harder instance going along with using this cutoff. So somehow, solving the easier instance gets priority over solving the harder instance. Under this interpretation, the "optimal" cutoff can be considered meaningful. Assuming, however, that in practice the goal in testing LVAs on test sets sampled from andom problem distributions is to get a realistic impression of the overall performance, *including hard instances*, the "optimal" cutoff inferred from the averaged RLD is substantially misleading.

Of course, a similar argument holds when averaging over test sets containing more than two instances. In Figure 2.11, we show the averaged RLD for the whole test set the hardness distribution of which is plotted in Figure 2.10. Obviously, here as well a "pseudo-optimal" cutoff can be observed, and the interpretation is exactly the same as in the two instance case. By increasing the size of the test-set, this situation is not changed, since it can

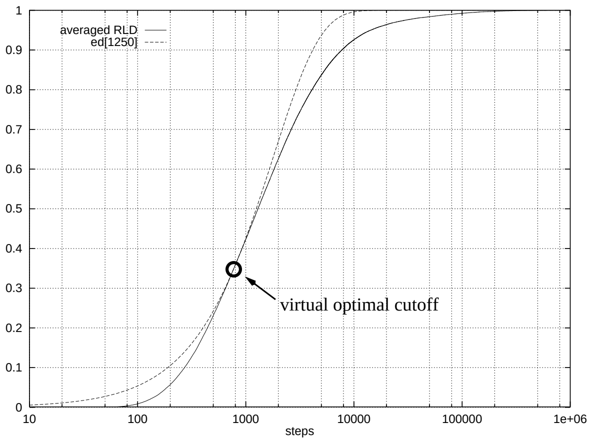


Figure 2.11: Average RLD for Walksat (noise=55/100, very large cutoff) on Random-3-SAT distr, 100 variables, 430 clauses; the virtual optimal cutoff at ca. 750 steps can be observed because the averaged RLD is not exponentially distributed (inhomogeneous test-set).

be theoretically shown that averaging over exponential distributions with different median values, one will *not* obtain another exponential distribution, *i.e.*, always "pseudo-optimal" cutoff times will be observed.

The above discussion shows, that by averaging over test sets, generally in the best case one observes knowingly a bias for solving certain problems, the practical use of which seems rather questionable. But far more likely, being not aware of these phenomena, the observations thus obtained are misinterpreted and lead to erroneous conclusions. One could, however, imagine a situation in which averaging over test sets is not that critical. This would be given if the test sets are very homogeneous in the sense that the RLDs for each single instance are roughly identical. Unfortunately, this sort of randomised problem distributions seems to be very rare in practice. Certainly, Random-3-SAT is not homogeneous in this sense, and at least the authors are currently not aware of any sufficiently complex homogenous randomised problem distribution for SAT or CSP.[5]

The fundamental problem with averaging over random problem distributions is the mixing of two different sources of randomness in the evaluation of algorithms: the nondeterministic nature of the algorithm itself, and the random selection of problem instances. Assuming that in the analysis of Las Vegas algorithms one is mostly interested in the properties of the algorithm, at least a very good knowledge of the problem distribution is required for separating the influence of these inherently different types of randomness.

[5] There is, however, some indication, that certain randomised classes of SAT encoded problems from other domains, such as compactly encoded subclasses of the Hamilton Circuit Problem [Hoo96c], are significantly more homogeneous than Random-3-SAT.

2.4 Extension to Optimisation Problems

Many problems occurring in practical contexts are optimisation problems rather than decision problems, *i.e.*, there is a notion of solution quality which, independently from the way a solution was obtained, characterises its value or usefulness. Las Vegas algorithms for optimisation problems (optimisation LVAs) are defined exactly as in the case of decision problems; only now, both run-time RT and solution quality SQ are random variables. Thus, for fixed solution quality sq', the run-time RT is a random variable, and for fixed run-time rt' the solution quality SQ is a random variable.

Often, for a given optimisation problem finding valid solutions is relatively easy, while finding optimal or near-optimal valid solutions might be extremely hard. We therefore make the assumption that for each instance of an optimisation problem there exists always a valid solution and Las Vegas algorithms for solving such an optimisation problem always return a valid solution. In this case, the optimisation problem Π is tightly related to the following family of decision problems: given a problem instance π and a desired solution quality q', is there a solution s of π such that the solution quality $q(s)$ is at most q', *i.e.* (for a minimisation problem), $\exists s : s \text{ solves } \pi \wedge q(s) \leq q'$. If the optimal solution q^* is known, it is often more elegant to use relative solution qualities $r(s) = q(s)/q^*(s)$, such that $r(s) = 1$ always characterises optimal solutions. In the following, we will use relative solution qualities unless explicitly indicated otherwise.

Interestingly, many LVAs for decision problems have been directly derived from optimisation LVAs; and vice versa, most LVAs for decision problems can be extended into optimisation LVAs in a natural and simple way. This merely reflects the close correspondence between certain decision problems and their optimisation variants which will be discussed later in this work. Good examples for LVAs which can be analogously applied to decision and optimisation problems can be found in the stochastic local search domain; some of the most popular SLS algorithms, like Simulated Annealing, Tabu Search, Genetic Algorithms, or Ant Colony Optimisation belong to this category.

2.4.1 Norms of Behaviour

With respect to the related decision problems introduced in above, the behaviour of a Las Vegas algorithm can be analysed using the methodology introduced before. Formally, we first extend the definitions of the norms of behaviour introduced in Section 2.1.1:

- A Las Vegas algorithm A is *q'-complete* for a problem class Π, if for each soluble instance $\pi \in \Pi$ it can be guaranteed to find a solution s with quality $q(s) \leq q'$ within run-time t_{max}, where t_{max} is an instance-dependent constant. Let $P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q')$ denote the probability that A finds a solution of quality $\leq q'$ for a soluble instance π in time $\leq t$, then A is complete exactly if for each π there exists some t_{max} such that $P_s(RT_{A,\pi} \leq t_{max}, SQ_{A,\pi} \leq q') = 1$.

- A Las Vegas algorithm A is *approximately q'-complete* for a problem class Π, if for each soluble instance $\pi \in \Pi$ its probability for finding a solution s with quality $q(s) \leq q'$ converges to 1 as the run-time approaches ∞. Thus, A is approximately q'-complete, if for each soluble instance π, $\lim_{t \to \infty} P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q') = 1$.

- A Las Vegas algorithm A is *essentially q'-incomplete* for a problem class Π, if it is not approximately q'-complete, *i.e.*, if there exists a soluble problem instance π, for which $\lim_{t \to \infty} P_s(RT_{A,\pi} \leq t, SQ_{A,\pi} \leq q') < 1$.

With respect to finding optimal solutions, we use the terms *complete*, *approximately complete*, and *essentially incomplete* synonymously for *q'-complete*, *approximately q'-complete*, and *essentially q'-incomplete*, where q' is the optimal solution quality for the given problem instance.

2.4.2 Application Scenarios

As for the application scenarios discussed in Section 2.1.2, the situation is similar as for decision problems; only now, there is the solution quality as an additional factor to be considered. Generally, there will be a tradeoff between run-time and solution quality. This trade-off will strongly depend on the specific requirements of the application situation: while in some cases run-time does not matter but solution quality is essential, other scenarios will impose tight restrictions on run-time, whereas solution quality is relatively less important. But since we are dealing with probabilistic algorithms, the situation is actually even more complex: for a given time-limit t', one might have to decide whether to prefer an algorithm with a high mean solution quality but a relatively large standard deviation over another algorithm which produces slightly less optimal solutions in a more consistent way. This becomes very obvious when generalising type 3 application scenarios (*cf.* Section 2.1.2) to optimisation problems: the utility of a solution now depends on its quality and the time needed to find it. Compared to the decision problem case, there is an additional degree of freedom in choosing the utility function $U(t,q) : \mathbb{R}^+ \times \mathbb{R} \mapsto [0,1]$, and the nature of the utility function to be used will generally reflect application specific tradeoffs between solution time and solution quality.

Therefore, for optimisation problems it seems to be even more important to avoid making implicit assumptions on the application scenario by prematurely reducing the data from empirical testing of Las Vegas algorithms to some basic statistics of the bivariate distribution $P_s(RT, SQ)$. This distribution characterises uniquely and completely the behaviour of a Las Vegas optimisation algorithm; running experiments is equivalent to sampling this distribution.

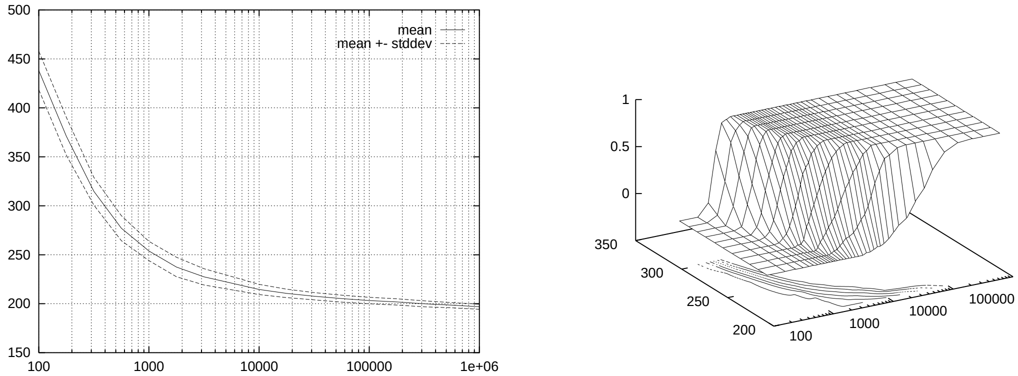


Figure 2.12: *Left:* development of mean solution quality and standard deviation for an SLS optimisation algorithm on a hard minimisation problem; *right:* qualified RLD for the same algorithm and problem instance.

2.4.3 Qualified RTDs and RLDs

Formally, a qualified RTD is a bivariate probability distribution characterising the two-dimensional random variable $(RT_{A,\pi}, SQ_{A,\pi})$ which models the run-time behaviour of Las Vegas optimisation algorithm A. The most natural and effective way of empirically approximating qualified RTDs is by measuring solution quality traces over multiple runs of the algorithm. This is most effectively done by recording the run-time and solution quality whenever during a run a solution is found, the quality of which strictly improves on the best solution found so far in this run. More formally, let k be the number of runs and let $sq(t,j)$ denote the quality of the best solution found in run j until time t. Then the qualified (cumulative empirical) run-time distribution is defined by $\widehat{P}(RT \leq t', SQ \leq q') = \#\{j | sq(t',j) \leq q'\}/k$. Again, to avoid the type of problems which arise in the context of averaging over test-sets (cf. Section 2.3), the qualified RTDs are based on single problem instances.

Like in the case of Las Vegas algorithms for decision problems, these RTDs describe the behaviour of the algorithm uniquely and completely. Therefore, all kinds of basic descriptive statistics can be easily obtained from the RTD data. Popular choices are mean and standard deviation of the solution quality for a given run-time (cutoff time) as well as mean, standard deviation, and percentiles of the run-time required for obtaining a given solution quality. Beyond these simple descriptive statistics, the two types of marginal distributions of the qualified (bivariate) RTD offer a good basis for characterising the run-time behaviour of optimisation LVAs: the first of these corresponds to the standard RTDs for the related decision problems discussed before and describe the distribution of the run-times needed for obtaining a specific solution quality. The other type of marginal distributions are solution quality distributions (SQDs) for a fixed run-time. Furthermore, when dealing with SQDs, it is often interesting to report and analyse their time-dependent basic statistics, like $\overline{SQ}(t)$, the function characterising the development of the mean solution quality over time.

Figure 2.12 (left) shows for a SLS optimisation algorithm on a particular instance of a

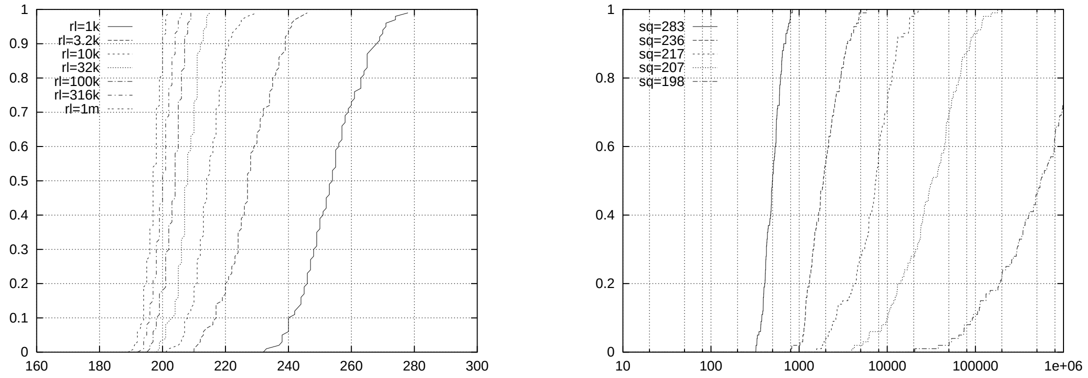


Figure 2.13: *Left:* marginal SQDs for the qualified RLD from Figure 2.12; *right:* marginal RLDs for the same qualified RLD.

run-length	mean	stddev	stddev/mean	min	max	median	Q_{25}	Q_{75}	Q_1	Q_{90}
100	438.21	19.64	0.04	397	491	439	425	452	411	460
1,000	253.83	9.81	0.04	232	278	253	247	261	240	267
10,000	214.55	5.19	0.02	201	230	214	211	218	207	221
100,000	203.22	3.31	0.02	195	209	204	201	206	198	207

Table 2.2: Basic SQD statistics for different run-lengths (same data as Figure 2.13, left).

minimisation problem the development of solution quality and standard deviation over time. From this type of evaluation, which is often used in the literature, we can easily see that in the given example the algorithm behaves in a very desirable way: with an increasing number of local search steps, the mean solution quality improves monotonically while the standard deviation of the solution quality decreases. Thus, for longer runs the algorithm tends to find better solutions in a more consistent way. However, this view of the algorithm's behaviour is rather simplistic compared to the complete qualified RLD shown in Figure 2.12 (right). Here, the contours of the three dimensional qualified RLD surface projected into the run-length / solution quality plane reflect the tradeoff between run-time and solution quality: for a given probability level, better solution qualities require longer runs, while vice versa, shorter runs yield lower quality solutions. The change in slope indicates the differences in variance of the run-time and solution quality distributions.

Details can be easier seen when examining the corresponding marginal distributions shown in Figure 2.13. The marginal SQDs (left) reflect the behaviour observed before, although they provide more information, like the fact that the solution quality distributions are consistently unimodal, which cannot be seen from the development of mean solution quality. Also, based on the marginal SQD data, other basic statistics, such as percentiles and their distances or ratios, can be easily obtained. For our example, some of these statistics can be seen in Table 2.2. Note, how the mean as well as the 10% to 90% percentiles (Q_{10} to Q_{90}) and the median monotonically decrease for growing run-lengths.

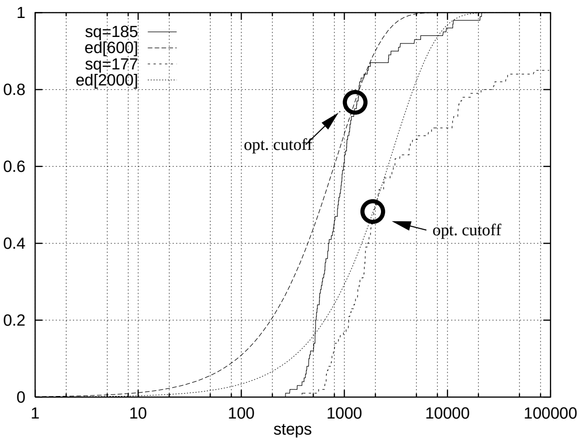


Figure 2.14: Two marginal RLDs for an SLS optimisation algorithm on a hard minimisation problem; for each curve, a different optimal cutoff time can be observed (the optimal cutoff is located where the RTD meets the corresponding exponential distribution).

A different view on the same data is given by the marginal RLDs, shown on the right side of Figure 2.13. Note that when tightening the solution quality bound, the marginal RLDs get less steep. This reflects the fact that for obtaining better solutions, not only the mean run-time increases, but also its variance. Thus, the qualified RLD data, which is the basis for these marginal distributions, provides a much more detailed picture of an SLS optimisation algorithm's behaviour than more established methods for evaluation based on simple solution quality statistics or their development over time.

2.4.4 Evaluating Optimisation LVAs

The methodology for evaluating and comparing optimisation LVAs is basically the same as for decision LVAs. Key techniques, as the approximation of empirical RTDs using continuous probability distribution functions, can be analogously applied to the optimisation case. However, directly dealing with bivariate qualified RTDs can be difficult. Therefore, it is usually more feasible to do functional approximations of the marginal RTDs and SQDs or time-dependent SQD statistics. In this case, the methodology for finding optimal approximations and testing their goodness-of-fit is exactly the same as for decision LVAs.

Again, the question of finding optimal cutoff times or optimal parallelisations is of considerable interest when analysing a given optimisation Las Vegas algorithm. Only here the problem is more complex than in the case of decision LVAs, where an optimal cutoff could be easily defined in an intuitive way. Consider a situation where an optimisation LVA A exhibits marginal RLDs for solution qualities q_1, q_2 as shown in Figure 2.14. Here, depending on the desired solution quality, two different optimal values for the cutoff time can be derived. Thus, generally the optimal cutoff can be regarded as a function of the desired solution quality rather than as a single value. Consequently, in most cases there is a trade-off between solution quality and run-time; one has to choose between optimal behaviour (*i.e.*, maximal probability) for short run-times while accepting lower quality solutions, and maximal probability for finding high-quality solutions while accepting long run-times.

Similiar considerations hold for comparing optimisation LVAs. The concept of domination which we introduced in Section 2.2.2 can be analogously applied to the optimisation case: For minimisation LVAs A and B, A dominates B if for any given time t and any solution quality q, A has a higher probability than B for finding a solution of quality $\leq q$ in time $\leq t$, *i.e.*, $\forall t, q : P(RT_A \leq t, SQ_A \leq q) \geq P(RT_B \leq t, SQ_B \leq q)$ and $\exists t, q : P(RT_A \leq t, SQ_A \leq q) > P(RT_B \leq t, SQ_B \leq q)$. But often, for two given algorithms no such dominance relation holds; instead the two qualified RTDs have at least one intersection. In this situation, analogously to the decision problem case, switching between the algorithms can enhance overall performance. However, for optimisation problems there is considerably more room for tradeoffs. For example, algorithm A might give a better mean solution quality up to some time limit t', while for $t > t'$, B will produce better solutions on average. But although for $t < t'$ A seems to be superior to B, B could still be better than A when just comparing the probabilities for finding high quality solutions within short runs. Formally, in this situation for all $t < t'$, the mean solution quality for algorithm B would be lower than for A, while, *e.g.*, when comparing the 0.1 percentiles of the corresponding SQDs, B would be superior to A. In situations like this, it depends entirely on the application scenario: sometimes, only high quality solutions are useful – then, in our example, algorithm B would be preferable. If, however, all solutions have to be used and average solution quality counts, A would be preferable for $t < t'$.

When evaluating and comparing optimisation algorithms, very often the application scenario is unknown or unspecified. Our considerations and examples have shown that for such general empirical analyses, it is extremely important to get a most comprehensive view of an algorithm's run-time behaviour, since by prematurely reducing the data to specific simple descriptive statistics, like the development of mean or standard deviation of the solution quality over time, important aspects of run-time behaviour are neglected. As for decision LVAs, the RTD-based methodology gives a considerably more detailed and realistic view of the algorithms' behaviour while not requiring a significant overhead in data aquisition.

2.5 Related Work

The term *Las Vegas Algorithm* was originally introduced by Laszlo Babai [Bab79]; it was first embraced by European researchers and only later picked up by Americans. The literature on Las Vegas algorithms is relatively sparse Some interesting theoretical work on the parallelisation of Las Vegas algorithms has been done by Michael Luby and Wolfgang Ertel [LE93]. They discuss parallelisation schemes for algorithms with known and unknown run-time distributions. Closely related is a theoretical analysis of optimal strategies for selecting cutoff times presented by Luby, Sinclair, and Zuckerman in [LSZ93]. For the case of a Las Vegas algorithm with known run-time distribution they theoretically derive an optimal strategy based on restarting the algorithm after a fixed cutoff time which can be exactly determined from the RTD. They also derive an optimal strategy for selecting optimal cutoff times in the case where the RTD is unknown. One might argue that this case is more realistic, since usually when solving a problem the characteristics of the particular instance are not known *a priori*. While in the most general case, this is undoubtedly true, it is not unusual to assume that for certain problem classes, general properties which hold for the whole problem class can be determined by studying a small number of instances [Min96]. In the case of known RTDs or distribution types, significantly better strategies for selecting cutoff times can be devised than in the case of unknown RTDs. Therefore, it is advantageous to study the run-time behaviour of a Las Vegas algorithms on specific problem classes and to use this information for improving the algorithm's peformance.

To our best knowledge, SLS algorithms for SAT or related problems have not been investigated in the more general context of Las Vegas Algorithms before. There is some work on the empirical analysis of complete search algorithms for SAT and CSP on randomly generated instances from Random-3-SAT and binary CSPs. Kwan shows that for random CSPs from the under- and overconstrained regions the run-time behaviour of modern complete algorithms cannot be characterised by normal distributions [Kwa96]. Frost, Rish, and Vila use continuous probability distributions for approximating the run-time behaviour of complete algorithms applied to randomly generated Random-3-SAT and binary CSPs from the phase transition region [FRV97]. In [RF97], this approach is extended to cost distributions for unsolvable problems from the overconstrained region. All these investigations concentrate on the cost distribution for a sample of instances from a fixed random distribution. In this approach, two sources of randomness are mixed: the randomness which is inherent to the algorithm and the variation in the characteristics of problem instances which are randomly drawn from a random problem distribution. In Section 2.3 we pointed out some pitfalls of this approach, which are avoided using our methodology.

Run-time distributions have been used for other problem domains, but again, this approach was mainly applied to complete algorithms. One exception is the work of Taillard, who reports exponential RTDs for a Tabu Search based algorithm for the Job-Shop Scheduling problem [Tai94]. In earlier work, the same author used a similar approach to improve on the best known solutions of small Quadratic Assignment Problems [Tai91]. Taillard also points out that for certain types of run-time distributions restarting the given algorithm can be used to obtain higher solution probabilities. Similar results are reported by Hogg and Williams, who investigate the behaviour of a backtracking algorithm for the Graph Colouring problem based on the Brelaz heuristic[6] [HW94a]. They use RTDs in the context of parallelisation and find that the obtainable speed-up strongly depends on the actual RTDs. Recently, a similar approach was used by Gomes and Selman, who tried to design algorithm portfolios using backtracking algorithms based on the Brelaz heuristic for solving a

[6] Since the Brelaz heuristic involves random tie-breaking, backtracking algorithms using this heuristic are Las Vegas algorithms, even if they are deterministic in any other respect.

special kind of CSP [GS97b]. Using run-time distributions it was found that one algorithm dominated all others. Consequently, using the optimal portfolio would contain only this algorithm. Closer examination of the RTDs showed that these can be approximated by "heavy-tailed" distributions, a fact which can be exploited for improving the performance of these heuristics by using restart [GSC97]. Latest results from this line of research indicate that a similar approach can be used for improving state-of-the-art complete SAT algorithms (like *Satz* [LA97]) [GSK98]. There is some evidence that the randomised algorithms thus obtained might be the best complete SAT algorithms which are currently known.

Finally, it should be noted that run-time distribution also have been observed occasionally in the Operations Research literature, where Las Vegas algorithms, specifically local search algorithms, play a prominent role for solving many hard problems. However, this kind of approach has been limited to concrete examples and specific application aspects.

2.6 Conclusions

In this chapter, we introduced a novel approach for the empirical analysis of Las Vegas Algorithms. Our method is based on measuring and analysing run-time distributions (RTDs) for individual problem instances. Based on a classification of application scenarios for Las Vegas Algorithms, we have shown that in general, only RTDs provide all the information required to adequately describe the behaviour of the algorithm. We demonstrated, how based on functional approximations of RTDs for individual problem instances, the behaviour of Las Vegas Algorithms can be adequately characterised even for sets or distributions of problem instances. This way, our refined methodology can be used to obtain interesting characterisations for the run-time behaviour of some of the most popular stochastic local search algorithms in recent AI research (this issue will be further elaborated in Chapter 5). Generally, compared to the methodology which is commonly used for empirical analyses for LVAs in AI, our approach gives a considerably more detailed and realistic view of the behaviour of these algorithms. At the same time it does not require an additional overhead in data aquisition, but only uses the collected data in a more efficient way.

Furthermore, we have shown how methods which have been commonly used in AI literature on stochastic local search can lead to misinterpretations and erroneous conclusions. In particular, we identified and discussed two pitfalls which are commonly arising in the context of an inadequate empirical methodology: superficial analysis of the run-time behaviour and averaging over inhomogeneous test-sets. As we have shown, by using our RTD-based methodology, these fallacies are effectively avoided. Finally, we demonstrated how the RTD-based empirical methodology can be generalised to Las Vegas Optimisation Algorithms in a straightforward way.

Although run-time distributions have been observed occasionally before, their use has been limited to concrete examples and specific application aspects. Our improved and refined empirical methodology is considerably more general; we believe that it will prove to be very useful for analysing the run-time behaviour of Las Vegas Algorithms in general, and SLS algorithms in particular. In the past, even the most successful applications of Las Vegas Algorithms in various areas of AI have been based on a fairly limited understanding of their behaviour. It appears to be not too bold to assume that using a more adequate empirical methodology will be the basis for improving the understanding of these algorithms and thus facilitate further successes in the development and application of this type of algorithms.

In the following chapters of this work, we will use the methodology developed here as a basis for analysing and comparing the behaviour of SLS algorithms for SAT. As we will see, it is crucial for obtaining many interesting results (particularly the ones presented in Chapter 5) of theoretical as well as practical interest.

Chapter 3

Generalised Local Search Machines

In this chapter, we introduce a novel formal framework for stochastic local search algorithms, the Generalised Local Search Machines (GLSM) model. The underlying idea is that adequate SLS algorithms are obtained by combining simple (pure) search strategies using a control mechanism; in the GLSM framework, the control mechanism is essentially realised by a non-deterministic finite state machine (FSM). This model provides a uniform framework capable of representing most modern SLS algorithms in an adequate way; it facilitates representations which clearly separate between search and search-control. As a consequence, the GLSM model can be very useful in developing and testing new, hybrid SLS algorithms. Furthermore, it offers analytical advantages, as well-known concepts from FSM theory can be applied to analyse SLS algorithms and search control mechanisms.

Here, we will mainly concentrate on general aspects of the GLSM model, while concrete GLSM realisations of SLS algorithms will be discussed in the following chapters. After a short introduction to local search algorithms, we introduce the basic GLSM model and define its semantics. Then, we establish the relation between the general scheme of local search and the GLSM model. Next, we discuss several structural GLSM types, transitions types, and state types. Finally, we will address extensions of the basic GLSM model, such as cooperative, evolutionary, and learning GLSMs. The chapter ends, as usual, with a brief overview of related work and a conclusion summarising its main contents.

3.1 Local Search Algorithms

Generally, local search algorithms are working in the following way: After initialising the search process at some point of the given problem instance's search space, the search iteratively moves from one position to a neighbouring position where the decision on each step is based on information about the local neighbourhood only. Thus, the following components are required to define a local search procedure for a problem class Π applied to a given problem instance $\pi \in \Pi$:

- the *search space* S_π of P which is a set of *positions* $a \in S_\pi$ (also called locations or states)
- a *set of solutions* $S' \subseteq S_\pi$
- a *neighbourhood relation* $N \subseteq S_\pi \times S_\pi$ on S_π
- an *initial distribution* $init : S_\pi \mapsto \mathbb{R}$ for selecting the initial search positions
- a *step function* $step : S_\pi \mapsto (S_\pi \mapsto \mathbb{R})$ mapping each position onto a probability distribution over its neighbouring states, for specifying the local search steps.

Often, local search algorithms make use of an *objective function* $f : S_\pi \mapsto \mathbb{R}$, mapping each search space position onto a real number in such a way, that the global optima correspond to the solutions. Without loss of generality, in this work we will always assume that the solutions correspond to the global minima of the objective function. This objective function can also be used to define the step function. For optimisation problems, the values of the objective function usually correspond to the quantity which is optimised. This way, in the context of local search, decision problems and optimisation problems can be treated quite analogously; only for the former, the result of the local search algorithm is generally useless if it is not a global minimum, while for optimisation problems, suboptimal solutions (usually local minima) can be useful on their own.

Examples for local search algorithms are stochastic local hill-climbing [MJPL90, MJPL92, SLM92], steepest descent, Simulated Annealing [KJV83], Tabu Search [Glo89, Glo90], iterated local search [MOF91, Joh90], Evolutionary Algorithms [Rec73, Sch81], and Ant Colony Optimisation algorithms [DMC96]. These algorithms are applied to a variety of hard combinatorial optimisation and decision problems, like Satisfiability in Propositional Logic (SAT), Constrained Satisfaction Problems (CSPs), the Travelling Salesperson Problem (TSP), scheduling and planning problems, etc.

Note that for $\mathcal{NP}$-complete problems like SAT or CSP, the search space is typically exponentially larger than the problem size. However, for a problem like SAT, local search algorithms are not restricted to operate on a search space defined by a set of candidate solutions (*i.e.*, assignments in the SAT case). It is also possible to use search spaces consisting of a set of partial solutions (for SAT, partial assignments) which contain the actual solution candidates as a subset. Also for SAT, local search methods need not operate on assignments at all; instead, they could, for example, use the set of all resolution proofs of a given formula as a search space and thus be used to solve the complementary UNSAT or the equivalent VAL problem.

Modern local search algorithms are often a combination of several pure strategies, like steepest descent and random walk, tabu search and random restart, or the search and diversification phases in iterated local search. This suggests that these algorithms operate on two levels: at a lower level, the pure search strategies are executed, while activation of and transitions between different strategies is controlled at a higher search control level. The GLSM model which we introduce in the following section is based on this distinction between search strategies and search control.

3.2 The Basic GLSM Model

Intuitively, a Generalised Local Search Machine (GLSM) for a given problem class Π is a finite state machine (FSM) each state of which corresponds to a simple local search strategy for instances of Π. The machine starts with an initial state s_0 and executes one step of the local search method associated with the current state. Then, according to a transition relation Δ, a new state is selected in a nondeterministic manner. This is iterated until either (1) a solution for the given problem instance is found or (2) a given maximal number ms of local search steps have been performed without satisfying condition (1). Note, that in this model the machine has no final states. Although it would be possible, and maybe more elegant from a theoretical point of view, to base the model on the notion of absorbing final states, this requires the introduction of additional states and transition types, which is avoided here for the sake of simplicity.

A *GLSM* is formally defined as a tuple $M = (S, s_0, \Delta, \sigma_S, \sigma_\Delta, \tau_S, \tau_\Delta)$ where S is a set of *states* and $s_0 \in S$ the initial state. $\Delta \subseteq S \times S$ is the *transition relation for* M; σ_S and σ_Δ are sets of state-types and transitions types resp., while $\tau_S : S \mapsto \sigma_S$ and $\tau_\Delta : \Delta \mapsto \sigma_\Delta$ associate the corresponding types to states and transitions. We call $\tau_S(s)$ the *type of state* s and $\tau_\Delta((s_1, s_2))$ the *type of transition* (s_1, s_2), respectively. The number $ms \in \mathbb{N}$ which specifies an upper bound on the number of steps, *i.e.*, state transitions, is not part of this definition because it is considered rather a parameter controlling the execution time behaviour than a structural aspect of the model.

It is useful to assume that σ_S, σ_Δ do not contain any types which are not associated with at least one state or transition of the given machine (*i.e.*, τ_S, τ_Δ are surjective). In this case, we define the *type of machine* M by $\tau_M := (\sigma_S, \sigma_\Delta)$. However, we allow for several states of M having the same type (*i.e.*, τ_s need not be injective). Note, that we do not require that each of the states in S can be actually reached when begining in state s_0; as we will shortly see, it is generally not trivial to decide this reachability. Thus, however, by adding unreachable states, the type of a given machine can be extended in an arbitrary way such that for any two GLSMs M_1, M_2, one can always find functionally equivalent models M_1', M_2' of the same type (*i.e.*, $\tau_{M_1'} = \tau_{M_2'}$).

The semantics of a GLSM M are defined by specifying an interpretation for its state- and transition types and then introducing a function $\pi_M^* : \mathbb{N} \mapsto \mathcal{D}(\Sigma)$, where $\mathbb{N}$ denotes the set of natural numbers and $\mathcal{D}(\Sigma)$ a distribution over the positions of search space Σ which is underlying the local search strategies for the given problem instance. Note that in the actual search process, there is only one position at each given instant. However, to capture the non-determinism of the search process, we have to use distributions over positions instead

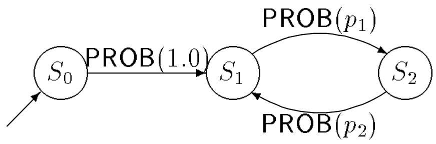


Figure 3.1: 3-state GLSM

when formalising the semantics of a GLSM. Intuitively, $\pi^*_M(n)$ determines the distribution over search space positions of the overall local search process realised by M at time n; it is therefore called the *search trajectory* of M.

Before giving a formal definition of the general semantics here, we demonstrate the specification and application of the GLSM concept by giving several examples. However, it should be noted that the specific semantics of a given GLSM always depends on the given problem class Π and the set of local search strategies which are used as interpretations of the state-types in σ_S. For the sake of simplicity, we will assume here that different state-types are always interpreted by different local search strategies (*i.e.*, the interpretation function for state-types is injective).

We will usually specify GLSMs by giving a graph representation for the finite state machine part (as it is commonly used in FSM theory) and additionaly labelling the states and transitions with their resp. types. As long as the meaning is clear from the context, we use the same symbol for denoting a state and its type. The initial state is marked by an ingoing arrow without a source. This is demonstrated for a small example in Figure 3.1; the corresponding machine is defined as

$$M = (\{S_0, S_1, S_2\}, S_0, \Delta, \sigma_S, \sigma_\Delta, \tau_S, \tau_\Delta)$$

with

$$\begin{aligned}
\Delta = \;& \{(S_0,S_1),(S_1,S_2),(S_2,S_1),(S_1,S_1),(S_2,S_2)\} \\
\sigma_S = \;& \{S_0,S_1,S_2\} \\
\sigma_\Delta = \;& \{\mathrm{PROB}(p) | p \in \mathbb{R}\} \\
\tau_S(S_i) = \;& S_i, \quad i \in \{1,2,3\} \\
\tau_\Delta((S_0,S_1)) = \;& \mathrm{PROB}(1.0) \\
\tau_\Delta((S_1,S_2)) = \;& \mathrm{PROB}(p_1) \\
\tau_\Delta((S_2,S_1)) = \;& \mathrm{PROB}(p_2) \\
\tau_\Delta((S_1,S_1)) = \;& \mathrm{PROB}(1-p_1) \\
\tau_\Delta((S_2,S_2)) = \;& \mathrm{PROB}(1-p_2)
\end{aligned}$$

The generic transition types PROB(p) correspond to unconditional, probabilistic transitions with an associated transition probability p. For simplicity's sake, we omit the transitions between a state and itself in the diagrammatic representation, using the default assumption, that whenever no other transition is selected, the next state is identical to the current state.

Thus the semantics of this small example can be intuitively described in the following manner: For a given problem instance π, the local search process is initialised by setting the machine to its initial state S_0 and executing one local search step corresponding to state type S_0. With a probability of 1.0, the machine then switches to state S_1, executing one step of the local search strategy corresponding to state type S_1. Now, with a probability of p_1, the machine switches to state S_2, performing one local search step of type S_2; otherwise it remains in S_1 and does an S_1-step. When in S_2, an analogous behaviour is observed; resulting in a local search process which repeatedly and nondeterministically switches between S_1 and S_2 steps. However, only once in each run of the machine, an S_0 step is performed, and that is at the very begin of the local search process. As described above, the local search process terminates when either a solution for the given problem instance is found or a given number ms of local search steps has been performed without finding a solution. Note, that if a solution is found, the machine terminates in the state in which the solution was discovered.

3.3 GLSM Semantics

To formally define the semantics of a GLSM as described above, we assume that the semantics of each state type τ are already defined in form of a trajectory $\pi_\tau : \Sigma \mapsto \mathcal{D}(\Sigma)$, where Σ denotes the set of positions in the search space induced by the given problem instance, and $\mathcal{D}(\Sigma)$ the set of distributions over Σ. Intuitively, π_τ determines for each position in the search space the resulting position after one τ-step has been performed.

We further need the functions $\pi_s : \Sigma \times S \mapsto \mathcal{D}(S)$ which model the direct transitions between states of the GLSM. These are defined on the basis of the transitions from a given state s and their respective types.

The $\pi_s(a, s)$ are given by the specific transition types of $\tau((s, s'))$;
for $\tau((s_i, s_k)) = \mathrm{PROB}(p_{i,k})$, $\pi_s(a, s_i) = D''_s$ with $D''_s(s_k) = p_{i,k}$.

Remark: To facilitate a concise formulation of the definitions, we often use the functional form of discrete probability distributions; thus for $D = \{\ldots, (e, p), \ldots\}$, $D(e)$ is equal to p and denotes the probability of event e.

The direct state transition functions π_s can be generalised into functions $\pi'_s : D(\Sigma) \times D(S) \mapsto D(S)$, mapping state distributions onto distributions of seachspace positions instead of a single, fixed positions.

$\pi'_s(D, D_s) = D'_s$
with $D'_s(s_k) = \sum_{a \in \Sigma, s \in S} D''_s(s_k) \cdot D(a) \cdot D_s(s)$
where $D''_s = \pi_s(a, s)$.

Based on the functions π_τ and π_s, we next define the direct search transition function

$\pi : \Sigma \times S \mapsto D(\Sigma)$ which determines for a given search space position and a state distribution the search position distribution after one step of the GLSM.

$\pi(a_k, s) = D''$
with $D''(a_j) = P(go\ from\ state\ s\ to\ s') \cdot P(in\ state\ s'\ go\ from\ a_k\ to\ a_j)$.
$P(go\ from\ state\ s\ to\ s') = D'_s(s)$
where $D'_s = \pi_s(a_k, s)$
$P(in\ state\ s'\ go\ from\ a_k\ to\ a_j) = D'_a(a_j)$
where $D'_a = \pi_{\tau(s')}(a_k)$

Again, this is generalised into the corresponding function $\pi' : D(\Sigma) \times D(S) \mapsto D(\Sigma)$.

$\pi'(D, D_s) = D'$
with $D'(a_k) = \sum_{a \in \Sigma, s \in S} D''(a_k) \cdot D(a) \cdot D_s(s)$
where $D'' = \pi(a, s)$.

Finally, we inductively define the state and search position trajectories $\pi_s^* : \mathbb{N} \mapsto D(S)$ and $\pi^* : \mathbb{N} \mapsto D(\Sigma)$. The interlocked inductive definitions reflect the operation of a GLSM, where in each step, the next search space position and the next state are determined based on the current state and position. The initial search space position $a_0 \in S$ can be arbitrarily chosen, since usually the state distribution determined in the first step does not depend on a_0.

π^* and π_s^* are defined inductively by:
$\pi^*(0) = D_0$ with $D_0(a_0) = 1, \forall a_k \in S - \{a_0\} : D_0(a_k) = 0$
$\pi^*(t+1) = \pi'(\pi^*(t), \pi_s^*(t))$

$\pi_s^*(0) = s_0$
$\pi_s^*(t+1) = \pi'_s(\pi^*(t), \pi_s^*(t))$

Remark: To keep the definitions simple and concise, we did not consider the termination condition here (*cf.* Section 3.2); however, this can be easily incorporated into the semantics of the individual search states.

Actual GLSM trajectories

Based on the semantics defined above, it is quite simple to define the notion of an *actual search trajectory* $\delta^* : \mathbb{N} \mapsto \Sigma$ which determines a sequence of search space positions visited by a given GLSM when actually being executed. Note, that due to the inherent non-determinism of GLSMs, generally each actual search trajectory will only be observed with a certain probability. To formally define δ^*, we use a function $draw(D)$ which for each probability distribution D, randomly selects an element e' from its domain such that $P(draw(D) = e') = D(e')$. Based on this, we define functions $\delta : \Sigma \times S \mapsto \Sigma$ and $\delta_s : \Sigma \times S \mapsto S$ which for each given position and state, randomly determine the position and state after one step of the GLSM:

$$\delta(a, s) = draw(\pi'(a, s))$$
$$\delta_s(a, s) = draw(\pi'_s(a, s))$$

Assuming that the given GLSM is started in state s_0 and at search position a_0, we now define the actual position and state trajectory by another double induction:

$\delta^* : \mathbb{N} \mapsto \Sigma$ and $\delta^*_s : \mathbb{N} \mapsto S$ are defined inductively by:
$$\delta^*(0) = a_0$$
$$\delta^*(t+1) = \delta(\delta^*(t), \delta^*_s(t))$$

$$\delta^*_s(0) = s_0$$
$$\delta^*_s(t+1) = \delta_s(\delta^*(t), \delta^*_s(t))$$

Note the similarity between these definitions and the ones for π^* and π^*_s; the only difference is in the use of the *draw* function to randomly select elements from the underlying probability distributions.

3.4 GLSMs and Local Search

The GLSM model has been introduced to provide a generalisation of the standard local search scheme presented in Section 3.1. Each GLSM, however, still realises a local search scheme and can therefore be described using the components of such a scheme. The notions of search space and solution set are not part of the model. This is mainly because both are not only problem- but actually instance-specific, they thus form the environment in which a given GLSM operates. Consequently, to characterise the behaviour of a GLSM when applied to a given instance, both the machine definition and this environment are required. The initial distribution is also not an explicit part of the GLSM model. The reason for this is the fact that the initial state, which is part of the model, can be easily used to generate arbitrary initial distributions of search space positions. The general local search scheme's step function is what is actually realised by the states of the GLSM and the finite control mechanism as given by the state transition relation.

The remaining component of the general local search scheme, the neighbourhood relation, generally does not have a direct representation in the GLSM model. This is because for a GLSM, each state type can be based on a different neighbourhood relation. However, for each GLSM as a whole, a neighbourhood relation can be defined by constructing a

Figure 3.2: Sequential *(left)* and alternating *(right)* 1-state+init GLSM.

generalised relation which contains the neighbourhood relation for each state type as a special case.

Taking a slightly different point of view, each GLSM state represents a local search algorithm of its own. While all these share one search space and solution set, they generally differ in their neighbourhood relations and step functions. In this case, however, initial distributions for the individual local search algorithms are not needed since they are defined by the context in which a GLSM state is activated.

3.5 Machine Types

One of the major advantages of using the GLSM model for characterising hybrid local search schemes is the clear distinction between search control and the actual search strategies thus facilitated. Abstracting from state and transition types, and thus concentrating on the structure of the search control mechanism alone, GLSMs can be categorised into the following structural classes:

1-state machines This is the minimal form of a GLSM. Since initialisation of the local search process has to be realised using a GLSM state, 1-state machines realise a very basic form of local search which basically only picks search space positions without doing actual local search. Consequently, the practical relevance of this machine type is extremely limited. It can, however, be used for analytical purposes, *e.g.* as a reference model when evaluating other types of GLSMs.

1-state+init machines These machines have one state for search initialisation and one working state, realising the search strategy. There are two sub-types of these machines: *1-state+init sequential machines* visit the initialisation state only once, while *alternating machines* might visit it again in the course of the search process, causing re-initialisations. As we will see in Chapter 4, most of today's popular SLS algorithms for SAT can be modelled by machines of this type.

2-state+init sequential machines This machine type has three states, one of which is an init state that is only visited once while the other two are working states. However, once the machine has switched from the first of these to the second, it will never switch back to the former again. Thus, each trajectory of such a machine can be partitioned into three phases: one initialisation step, a number of steps in the first working state and a number

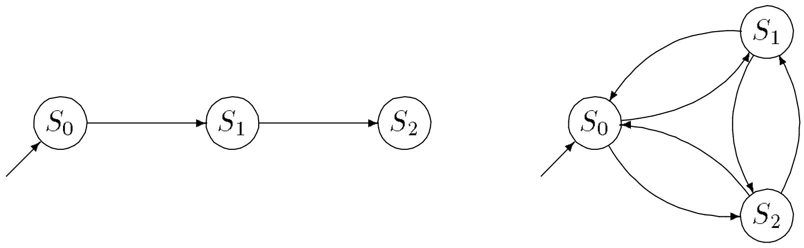


Figure 3.3: Sequential *(left)* and alternating *(right)* 2-state+init GLSM.

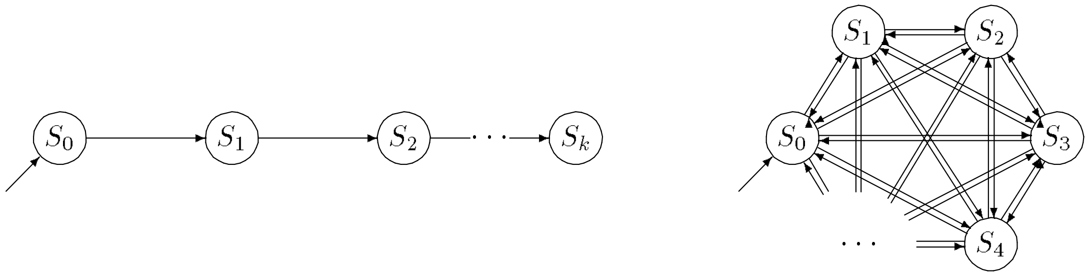


Figure 3.4: Sequential *(left)* and alternating *(right)* k-state+init GLSM.

of steps in the second working state.

2-state+init alternating machines Like the 2-state+init sequential machine, this machine type has one initialisation state and two working states. Here, however, arbitrary transitions between all states are possible. An interesting sub-type of these machines is given by the special case in which the initial state is only visited once, while the machine might arbitrarily switch between the two working states. Another sub-type that might be distinguished is a uni-directional cyclic machine model which allows the three states to be visited only in one fixed order.

Of course, the categorisation can be easily continued in this manner by successively increasing the number of working states. However, as we will later see, to describe state-of-the-art stochastic local search algorithms, usually three-state-machines are sufficient. We therefore conclude this categorisation with a brief look at two potentially interesting cases of the k-state+init machine types:

k-state+init sequential machines As a straightforward generalisation of the sequential 2-state+init machines, in this machine type we have $k+1$ states which are visited in a linear order. Consequently, after a machine state has been left, it will never be visited again.

k-state+init alternating machines These machines allow arbitrary transitions between the $k+1$ states and may therefore re-initialise the search process and switch between strategies as often as desired. Some special cases worth noting are the uni- and bi-directional cyclic machine models which allow to switch between states in a cyclic manner. In the

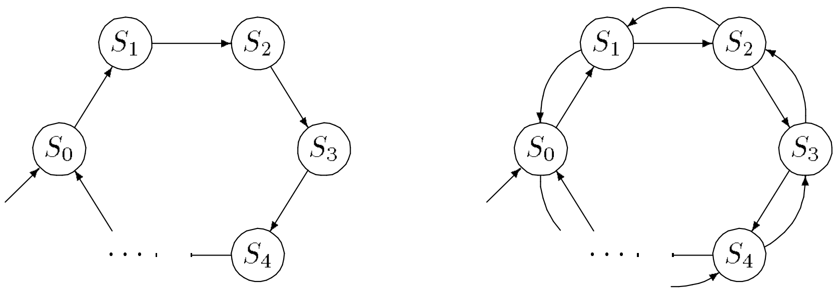


Figure 3.5: Uni-directional *(left)* and bi-directional *(right)* cyclic k-state+init GLSM.

former case, the cyclic structure can be traversed only in one direction, in the latter case the machine can switch from any state to both its neighbouring states.

This categorisation of machines according to their structure is useful for characterising the structural aspects of the search control as realised by the GLSM model. Of course, this is a very high-level view of GLSMs which can be refined in many ways, but nevertheless in the context of this work it will prove to be useful for capturing some fundamental differences between various stochastic local search schemes.

3.6 Transition Types

In refining the structural view of GLSMs given above, we next focus on transition types. As mentioned before, properties of the transition types are used as a basis for defining GLSM semantics. Here, we will introduce transition types in terms of a hierarchy of increasingly complex (or expressive) types and define the semantics in terms of the state transition functions π_s for each transition type.

Unconditional deterministic transitions, DET

This is the most basic transition type; DET transitions from state s_i to state s_k cause, when the GLSM is in state s_i, always an immediate transition into state s_k. Formally, if $\tau((s_i, s_k)) = \text{DET}$, this behaviour is captured by $\pi_s(a, s_i) = D''_s$ with $D''_s(s_k) = 1$ for arbitrary $a \in \Sigma$. Note that because D''_s is a probability distribution, the above condition implies $D''_s(s) = 0$ for all states $s \neq s_k$.

The use of this transition type is fairly limited, because it causes a state with such a transition as its source to be left immediately after being entered. This obviously implies that for each state there can be only one transition leaving it. Consequently, using exclusively DET transitions, one can only realise a very limited class of GLSM structures. However, at least for the transition leaving the initial state, DET transitions are frequently used in practically occurring GLSMs.

Unconditional probabilistic transitions, PROB(p)

A PROB(p) transition from the actual state is executed with probability p. Thus, DET transitions are actually equivalent to a special case of this transition type, namely to PROB(1). For the moment, we can therefore assume without loss of generality that all transition types in a given GLSM are of type PROB. To define the semantics of this transition type, we consider an arbitrary GLSM state s_i and assume that the set of transitions leaving s_i is given as $\{t_1, \ldots, t_n\}$. If further $\tau(t_j) = \text{PROB}(p_j)$, we can define the semantics of PROB transitions by $\pi_s(a, s_i) = D''_s$ with $D''_s(s_k) = p'$ for arbitrary $a \in \Sigma$ where p' is given by $\tau((s_i, s_k)) = \text{PROB}(p')$. Note that to guarantee that D''_s is a probability distribution, and $\sum_{j=1}^{n} p_j$ must be equal to one.

Note that by using PROB(0) transitions we can restrict ourselves to fully connected GLSMs, where for each pair of states (s_i, s_k), a transition of type PROB(p_{ik}) is defined. This allows a more uniform representation of this class of GLSMs which in turn will facilitate both theoretical investigations and practical implementations of this GLSM type. Furthermore, the behaviour of these GLSMs can be easily modeled using Markov processes [Çin75], which facilitates their analysis, as well-known techniques for studying Markov processes can be directly applied.

Conditional probabilistic transitions, CPROP(C, p)

While until now we have focused on transitions the execution of which only depends on the actual state, the next generalisation from PROB(p) introduces context dependent transitions. A CPROP(C, p) transition from state s_i to state s_k is executed with a probability proportional to p only when a condition predicate C is satisfied. If C is not satisfied, all transitions CPROP(C, p) from the current state are blocked, *i.e.*, they cannot be executed. For practical reasons, the condition predicates C will depend on local information only; this includes information on the current search space position, its immediate neighbourhood, the number of local search steps performed up to this point, and, possibly, some simple statistics on these. We will see later some predicates which can be practically used. Generally, the crucial condition restricting the choice of condition predicates C is that these have to be efficiently computable (when compared to the cost for executing local search steps).

Obviously, PROB(p) transitions are equivalent to CPROB($\top, p$) conditional probabilistic transitions, where $\top$ is the predicate which is always true. Without loss of generality, we can therefore assume that for a given GLSM all transitions are of type CPROB(C, p). To define the semantics of this transition type we consider the actual GLSM state s_i. As s_i is the actual state, we have all the local information to decide the condition predicates of all transitions leaving s_i. Since in this situation only a *non-blocked* transition can be executed, *i.e.*, a transition for which C is satisfied and therefore equivalend to $\top$ in the given situation, we can now define the semantics like in the case of PROB(p) transitions. To this end, we assume that the set of *non-blocked* transitions leaving s_i is given as $\{t_1, \ldots, t_n\}$, and for $\tau(t_j) = \text{CPROB}(C_j, p_j)$, C_j currently satisfied, we define $\pi_s(a, s_i) = D''_s$ with $D''_s(s_k) = p'/c$

$\top$	always true
$\mathsf{tcount}(k)$	total number of local search steps $\geq k$
$\mathsf{mcount}(k)$	total number of local search modulo $k = 0$
lmin	current local search position is local minimum w.r.t. its direct neighbours
$\mathsf{obf}(x)$	current objective function value $\leq x$
$\neg\ \mathsf{impr}(k)$	objective function value could not be improved within last k steps

Table 3.1: Some examples for simple condition predicates.

for arbitrary $a \in \Sigma$ where p' is given by $\tau((s_i, s_k)) = \mathrm{CPROB}(C', p')$ and $c = \sum_{j=1}^{n} p_j$ is the normalisation factor which ensures that D''_s is a probability distribution.

An important special case of conditional transitions are *conditional deterministic* transitions. These occur, if for a given GLSM state s_i, from all the transitions leaving it at most one transition is not blocked. One way to obtain deterministic GLSMs using conditional transitions is to make sure that by their logical structure, the condition predicates for the transitions leaving each state are mutually exclusive (or non-overlapping). Generally, depending on the nature of the condition predicates used, the decision whether a conditional transition is deterministic or not can be very difficult. For the same reasons it can be difficult to decide for a given GLSM with conditional probabilistic transitions, whether a particular state is reachable from the initial state.

For practically using GLSMs with conditional transitions it is important to make sure that the condition predicates can be evaluated in a sufficiently efficient way. There are two kinds of condition predicates, the first of which is based on the search space position and/or its local neighbourhood. The other, however, is based on search-control aspects alone, like the time that has been spent in the current GLSM state, or the overall run-time. Of course, these two kinds of conditions can also be mixed. Some concrete examples for condition predicates can be seen in Table 3.1. Note that all these predicates are based on only local information and can thus be efficiently evaluated during the search.

Usually, for each condition predicate, a positive as well as a negative (negated) form will be defined. Using propositional connectives like "$\wedge$" or "$\vee$", these simple predicates can be combined into complex predicates. However, it is not difficult to see that every GLSM using complex condition predicates can be reduced to an equivalent GLSM using only simple predicates by introducing additional states and/or transitions. Thus, using complex condition predicates can generally be avoided without restricting the expressive power of the model.

Transition actions

After discussing a hierarchy of increasingly general transition types, we now introduce another conceptual element into the context of transitions: transition actions. Transition actions are associated with the individual transitions and are executed whenever the GLSM executes the corresponding transition. At this point, the motivation for adding this notion to the GLSM model might not be obvious. However, as we will see, there are some situations in which transition actions provide an adequate method of modelling SLS algorithms. One such case is the manipulation of global search parameters, like adapting the length of a tabu list or a noise parameter.

Generally, transition actions can be added to each of the transition types defined above, while the semantics of the transition (in terms of π_s) is not affected. If T is a transition type, by $T : A$ we denote the same transition type with associated action A. The nature of the actions A has to be such that they neither directly affect the state of the GLSM, nor its search space position. Instead, the actions generally can be used for

- modifying global parameters of one or more state types,
- realisation of input / output functionality in actual GLSM realisations,
- communication between GLSMs in cooperative GLSM models.

By introducing an action NOP without any effects we obtain uniform GLSMs in which all transitions have associated actions. Note, however, that we do not need multiple actions (*i.e.*, sequences or sets of actions which are associated with the same transition), because by introducing copies of a transition's destination state the (intuitive) semantics of multiple actions can be emulated.

From a minimalist point of view, of course, even simple transition actions are not strictly required because they, too, can be emulated by embedding the corresponding actions into the search strategies associated with the GLSM states. This, however, means to mix conceptually different notions, namely the local search strategies and the actions which are rather part of the search control mechanism that is represented by the modified finite state machines underlying the GLSM model. Because here our main motivation is to devise an *adequate* representation of SLS algorithms, if the notion of transition actions occurs naturally, we prefer to rather model them explicitly in the way outlined above.

3.7 State Types

At this point, the only component for specifying concrete GLSMs which is still missing are state types. As outlined above, for formally specifying the semantics of a GLSM, the semantics of the individual state types are required to be specified in the form of a trajectory $\pi_\tau : \Sigma \mapsto D(\Sigma)$. For practical purposes, however, state types will usually be rather defined in a procedural way, usually by using some form of pseudo-code. In some cases, more adequate descriptions of complex state types can be obtained by using other formalisms, such as decision trees. Concrete examples for various state types will be give in Chapter 4, where we show how existing local search algorithms for SAT can be represented as GLSMs.

Here, we want to concentrate on some fundamental distinctions between certain state types. One of these concerns the role of the state within the general local search scheme presented in Section 3.1. Since we are modelling the search initialisation and local search steps using the same mechanism, namely GLSM states, there is a distinction between initialisation states and search step states. An initialisation state is usually different from a search step state in that it is left after one corresponding step has been performed. Also, while search step states correspond to moves in a restricted local neigbourhood (like flipping one variable in SAT), one initialisation step can lead to arbitrary search space positions (like a randomly chosen assignment of all variables of a SAT instance). Formally, we define an *initialising state type* as a state type τ, for which the local search position after one τ-step is independent of the local search position before the step; the states of an initialising type τ are called *initialising states*. Generally, each GLSM will have at least one initialising state, which is also its initial state. A GLSM can, however, have more than one initialising state and use these states for dynamically restarting the local search process.

If for a given problem there is a natural common neighbourhood relation for local search, we can also distinguish *single-step states* from *multi-step states*. For the SAT problem, most local search algorithms use a neighbourhood relation where two variable assignments are direct neighbours if they differ in exactly one variable's value. In this context, a single-step state would flip one variable's value in each step, whereas a multi-step state could flip several variables per local search step. Consequently, initialising states are an extreme case of multi-step states, since they can affect all variable's values at the same time.

3.8 Extensions of the Basic GLSM Model

In this section we discuss various extensions of the basic GLSM model. One of the strengths of the GLSM model lies in the fact that these extensions arise quite naturally and can be easily realised within the basic framework. However, in the context of this work none of the extensions proposed here has been studied in detail, with the exception of the homogeneous cooperative model discussed later in this section. The main reason for this lies in the fact that in the context of SAT and CSP, the existing state-of-the-art SLS algorithms are conceptually fairly simple but nevertheless very powerful. Consequently, as we will see later, improvements of these algorithms can be achieved using simple GLSM techniques, but this requires a rather detailed understanding of their behaviour. The extensions as outlined in the following are of a more complex nature, but at the same time can be applied to all domains for which local search techniques are available.

Learning via dynamic transition probabilities

One of the features of the basic GLSM model with probabilistic transitions is the fact that the transition probabilities are static, *i.e.*, they are fixed when designing the GLSM. An obvious generalisation, along the lines of learning automata theory [NT89], is to let the transition probabilities evolve over time as the GLSM is running. The search control in this model corresponds to a variable structure learning automaton. The environment such a dynamic GLSM is operating in, is given by the objective function induced by an individual problem instance or a class of objective functions induced by a class of instances. In the first case (*single-instance learning*), the idea is to optimise the control strategy on one instance during the local search process. The second case (*multi-instance learning*), is based on the assumption that for a given problem domain (or sub-domain), all instances share certain features to which the search control strategy can be adapted.

The modification of the transition probabilities can either be realised by an external mechanism (external adaption control), or within the GLSM framework by means of specialised transition actions (internal adaption control). In both cases, suitable criteria for transition probability updates have to be developed. Two classes of such criterias are those based on trajectory information, and those based on GLSM statistics. The latter category includes state occupancies and transition frequencies, while the former comprises primarily basic descriptive statistics of the objective function value along the search trajectory, possibly in conjunction with discounting of past observations. The approach as outlined here captures only a specific form of parameter learning for a given parameterised class of GLSMs. Conceptually this can be further extended to allow for dynamic changes of transition types (which is equivalent to parameter learning for a more general transition model, such as conditional probabilistic transitions).

Concepts and methods from learning automata theory can be used for analysing and characterising dynamic GLSMs; basic properties, such as expedience or optimality can be easily defined. We conjecture, however, that theoretically proving such properties will be extremely difficult, as the theoretical analysis of standard SLS behaviour is already very complex and limited in its results. Nevertheless, we believe that based on empirical methodology, it should be possible to devise and analyse dynamic GLSMs.

Cooperative GLSM models

Another line of extending the basic GLSM model is to apply several GLSMs simultaneously to the same problem instance. In the simplest case, such an ensemble consists of a number of identical GLSMs and there is no communication between the individual machines. We call this the *homogenous cooperative GLSM model without communication*; its semantics are conceptually equivalent to executing multiple independent tries of an individual GLSM. In Chapter 5 we will use this scheme in the context of efficiently parallelising SLS algorithms for SAT. It is particularly attractive for parallelisation, because it is very easy to implement, involves virtually no communications overhead, and can be almost arbitrarily scaled.

The restrictions of this model can be relaxed in two directions. One is to allow ensembles of different GLSMs. This *heterogeneous cooperative GLSM model without communication* is particularly useful for modelling robust combinations of various SLS algorithms, each of which shows superior performance on certain types of instances, when the features of the given problem instances are not known *a priori*. This approach has been recently studied in the context of complete algorithms for hard combinatorial problems [GS97a]; in this context the heterogenous ensembles were called *algorithm portfolios*. Generally, this cooperative model has almost the same advantages as its homogeneous variant; it is easy to implement and almost free of communication overhead.

Another generalisation is to allow communication between the individual GLSMs of a cooperative model. This communication can be easily realised by means of transition actions (*e.g.*, `send` and `receive`); possible communication schemes include using a blackboard, synchronous broadcasting, and one-to-one message passing in a fixed network topology. These different variants of *cooperative GLSMs with communication* are more difficult to design and to realise, since issues like preventing and detecting deadlocks and starvation situations generally have to be considered. Furthermore, the communication between individual GLSMs usually involves a certain amount of overhead. This overhead has to be amortised by the performance gains which can be achieved by using this model in a given application situation. These gains may be realised in terms of speedup when applied to a specific problem class, but also in terms of increased robustness w.r.t. different problem types.

Generally, one way of using communication to improve the performance of cooperative GLSMs is to propagate search space positions with low objective function values (or other attractive properties) within the ensemble such that individual GLSMs which detect that they are not doing particularly well can pick up these "hints" and restart their local search from there. This can be easily realised as a homogeneous cooperative GLSM with communication. In such a model, the search effort will be more focussed on exploring promising parts of the search space than in a cooperative model without communication. Another general scheme uses two types of GLSMs, analysts and solvers. Analysts do not attempt to find solutions but rather try to analyse features of the search space. The solvers try to use this information to improve their search strategy. This architecture is an instance of the heterogeneous cooperative GLSM model with communication. It can be extended in a straightforward way to allow for different types of analysts and solvers, or several independent sub-ensembles of analysts and solvers.

Evolutionary GLSM models

From the cooperative GLSM models discussed in the previous section it is only a short step to evolutionary GLSMs. These are cooperative models where the number or type of the individual GLSMs may vary over time; these population dynamics can be interpreted as another form of learning. As for the learning GLSMs described earlier, we can distinguish between *single-instance* and *multi-instance learning* and base the dynamic adaption process on similar criteria. In the conceptually most simple case, the evolutionary process only affects the composition of the cooperative ensemble: machines which are doing well will spawn off numerous offspring replacing individuals showing inferior performance. This mechanism can be applied to both, homogeneous and heterogeneous models for single-instance learning. In the former case, the selection is based on the trajctory information of the individual machines and achieves a similar effect as described above for the homogeneous cooperative GLSM with communication: The search is concentrated on exploring promising parts of the search space. When applied to heterogeneous models, this scheme allows to realise self-optimising algorithm portfolios, which can be useful for single-instance as well as multi-instance learning.

This scheme can be further extended by introducing mutation, and possibly cross-over operators. It is also possible to combine evolutionary and indivual learning by evolving ensembles of learning GLSMs. And finally, these models can also allow communication within the ensemble. Thus, combining different extensions we arive at very complex and potentially powerful GLSM models; while these are very expressive, in general they will also be extremely difficult to analyse. Nevertheless, their implementation is quite simple and straightforward and an empirical approach for analysing and optimising their behaviour seems viable enough. We expect that such complex models, which allow for a very flexible and fine-grained search control, will be most effective when applied to problem classes which contain a lot of structural features. There is little doubt that, to some extent, this is the case for most real-world problem domains.

Continuous GLSM models

The basic GLSM model and all extensions thereof discussed until here model local search algorithms for solving discrete decision or optimisation problems. But of course, the model can easily be applied to continuous problems; the only changes required are to use continuous local search strategies for the GLSM state types instead of discrete ones. Although our experience and expertise is mainly limited to discrete combinatorial problems, we assume that the GLSM model's main feature, the clear distinction between simple search strategies and search-control, is also a useful architectural and conceptual principle for continuous optimisation algorithms.

3.9 Related Work

The main idea underlying the GLSM model, namely to realise adequate algorithms as a combination of several simple strategies, seems to be common lore. However, our application of this general metaphor to local search algorithms for combinatorial decision and optimisation problems using suitably extended finite state machines for search control, is to our best knowledge novel and original. The GLSM model is partly inspired by Amir Pnueli's work on hybrid systems [MMP92] and Thomas Henzinger's work on hybrid automata; the latter uses finite state machines to model systems with continuous and discrete components and dynamics [ACHH93, Hen96] and is therefore conceptually related to the GLSM model.

The GLSM definition and semantics are heavily based on well-known concepts from automata theory (for a general references, *cf.* [Har78, RS97]). However, when using conditional transitions or transition actions, the GLSM model extends the conventional model of a finite state machine. In its most general form, the GLSM model bears close resemblance to a restricted form of Petri nets [Kri89], where only one token is used. Of course, the same type of search control mechanism could be represented using formal systems other than a FSM-based formalism. First of all, other types of automata, such as pushdown automata or Turing machines could be considered. However, we feel that the FSM model, one of the simplest types of automata, is powerful enough for representing most interesting search-control strategies we found in the local search literature. Furthermore, it is our impression that it leaves enough room for extension, while being analytically more tractable than more complex automata models. Finally, FSMs offer the potential advantage of being implementable in hardware in a rather straightforward way, which might be interesting in the context of applying local search algorithms to time-critical problems (*cf.* [HM97]). Summarising these arguments, the use of FSMs for formalising the search-control mechanism seems to be sufficient and adequate. Of course, formalisms equivalent to the FSM model, such as rule-based descriptions, could be chosen instead. While this could be easily done and might be advantageous in certain contexts (such as reasoning about properties of a given GLSM), we find that the automaton model provides a slightly more intuitive and easier-to-handle framework for designing and implementing local search algorithms, the nature of which is mainly procedural.

The GLSM model allows to adequately represent existing algorithmic frameworks for local search, such as GenSAT [GW93a] or iterated local search [MOF91, Joh90]. These frameworks are generally more specific and more detailed than the GLSM model; however, they can be easily realised as generic GLSMs without losing any detail of description. This is done by using structured generic state types to capture the more specific aspects of the framework to be modeled. The same technique will be used in Chapter 4 to model modern SLS algorithms for SAT, such as WalkSAT or R-Novelty. In these cases, the structure of the simple search strategies can be represented by decision trees which are associated with the state types. While the GLSM model can be used to represent *any* local search algorithm, many of these do not really make use of the search control mechanism it provides. Note, however, that some of the most successful local search algorithms for various problem classes (such as R-Novelty for SAT [MSK97], *hrts* for MaxSAT [BP96], and iterated local search schemes for TSP [MOF91, Joh90, JM97]) rely on search control mechanisms of the type provided by the GLSM model.

The various extensions of the basic model discussed in this chapter are closely related to established work on learning automata [NT89], parallel algorithm architectures [Jáj92], and evolutionary algorithms [Bäc94]. While most of the proposed extensions have not been implemented and empirically evaluated so far, they appear to be promising, especially when taking into account our results on homogeneous cooperative GLSM models reported in Chapter 5 and recent work on multiple independent tries parallelisation [Sho93, GSK98], algorithm portfolios [GS97a], and learning local search approaches for solving hard combinatorial problems [BM98, Min96].

3.10 Conclusions

Based on the intuition that adequate local search algorithms are usually obtained by combining several simple search strategies, in this chapter we introduced and discussed the GLSM model. This framework formalises the search control using a finite state machine (FSM) model, which associates the pure search strategies with the FSM states. FSMs belong to the most basic and yet fruitful concepts in computer science; using them to model local search control offers a number of advantages. First, FSM-based models are conceptually simple; consequently, they can be implemented easily and efficiently. At the same time, the formalism is expressive enough to allow for the adequate representation of a broad range of modern local search algorithms (this will be exemplified in Chapter 4, where GLSM-realisations of several state-of-the-art SLS algorithms for SAT are given). Secondly, there is a huge body of work on FSMs; many results and techniques can be directly applied to GLSMs which is especially interesting in the context of analysing and optimising GLSMs. And finally, in our experience, the GLSM model facilitates the development and design of new, hybrid local search algorithms. In this context, both conceptual and implementational aspects play a role: due to the conceptual simplicity of the GLSM model and its clear representational distinction between search strategies and search control, hybrid combinations of existing local search algorithms can be easily formalised and explored. Using a generic GLSM simulator, which is not difficult to implement, new hybrid GLSM algorithms can be realised and evaluated in a very efficient way.

As we have shown, based on a clean and simple definition of a GLSM, the semantics of the model can be formalised in a rather straightforward way. We then discussed the tight relation between the GLSM model and a standard generic local search scheme. By categorising GLSM types according to their structure and transition types, we demonstrated how the general model facilitates the systematic study of search control mechanisms. Finally, we pointed out several directions into which the basic GLSM model can be extended. Most of these are very natural generalisations, such as continous or cooperative GLSMs; however, these proposed extensions demonstrate the scope of the general idea and suggest numerous routes for further research.

In the context of this work, GLSMs will be used for formalising, realising, and analysing SLS algorithms for SAT. In Chapter 5 we will show how by applying minor modifications to these GLSMs, some of the best known SAT algorithms can be further improved. In Chapter 6, we will use very simple GLSMs as probes for analysing the search space structure of SAT instances. While these applications demonstrate the usefulness of the GLSM model, they hardly exploit its full power. Most modern SLS algorithms can be represented by very simple GLSMs; the fact that the most efficient of them are structurally slightly more complex than others suggests that further improvements can be achieved by developing more complex combinations of simple search strategies. While some of our results presented in Chapter 5 support this hypothesis, its general validity remains to be shown by developing novel hybrid GLSM algorithms for different domains.

Chapter 4

SLS Algorithms for SAT

In this chapter, we give an overview of some of the most relevant SLS algorithms for SAT and analyse their behaviour when applied to a number of benchmark problems. We present and discuss these algorithms in the context of the GLSM model, which provides a convenient and uniform framework especially suited for adequately representing some of the more recent schemes, like R-Novelty. Using the GLSM model as a unifying framework enables us to present these algorithms in a systematic way, showing more clearly the relations and differences between some of the most prominent SLS algorithms. We then present a detailed comparative study of these algorithms' performance based on the methodology developed in Chapter 2. The benchmark set we use in this context contains instances from randomised distributions as well as SAT-encoded problems from the Graph Colouring, the Blocks World Planning, and the newly developed All-Interval-Series domains. Our empirical analysis gives a very detailed picture of the algorithms' performance and reveals some interesting differences between the problem domains.

4.1 Introduction

Stochastic local search approaches for SAT became prominent in 1992, when independently Selman, Levesque, and Mitchell [SLM92] as well as Gu [Gu92] introduced algorithms based on stochastic local hill-climbing which could be shown to outperform state-of-the-art systematic SAT algorithms like ASAT [DABC93] on a variety of hard subclasses of SAT [BKB92, SKC94]. Since then, numerous other SLS schemes for SAT have been proposed. To date, state-of-the-art SLS algorithms can solve hard SAT problems up to several thousand variables, including SAT-encoded problems from other domains.

All algorithms considered here are model finding algorithms for CNF formulae. The underlying state space is always defined as the set of all assignments for the variables appearing in the given formula. Local search steps generally modify at most the value assigned to one of the propositional variables appearing in the formula; such a move is called a *variable* *flip*. The objective function is always defined as the number of clauses which are unsatisfied under a given variable assignment; thus, the models of the given formula are always the global minima of this function. The general idea for finding these is to perform stochastic hill-climbing on the objective function, starting from a randomly generated initial assignment.

The main difference between the individual algorithms lies in the strategy used to select the variable to be flipped next. This is generally done based on a scoring function which is used to preselect a number of variables from which then at most one is chosen to be flipped. In the following, we focus on two families of algorithms, the GSAT and WalkSAT architectures, which provided a substantial driving force for the development of SLS algorithms for SAT and have been extremely succesful when applied to a broad range of problems from different domains. Another important reason for choosing these algorithms is the fact that extremely efficient implementations are available, which is of crucial importance for the empirical studies we conducted. However, there are other successful SLS algorithms for SAT some of which will be briefly discussed in the "Related Works" section of this chapter.

4.2 The GSAT Architecture

The GSAT algorithm was introduced in 1992 by Selman, Levesque, and Mitchell [SLM92]. It is based on a rather simple idea: Using the number of clauses which are unsatisfied under the current assignment as the objective function, GSAT tries to minimise this function by greedy hill-climbing in the space of variable assignments. The neighbourhood relation used for GSAT is defined in such a way that two assignments are neighbours if and only if they differ in the value of exactly one variable, *i.e.*, their Hamming distance is equal to one. Variable selection in GSAT and most of its variants is based on the score of a variable x under the current assignment a; this is defined as the difference between the number of clauses unsatisfied by a and the assignment obtained by flipping x in a.

4.2.1 Basic GSAT

The basic GSAT algorithm works as follows. Starting at a randomly selected initial assignment, in each local search step, one of the variables with maximal score is flipped. If there are several variables with maximal score, one of them is randomly selected. If after a number $\mathsf{maxSteps}$ of these local search steps no solution was found, GSAT restarts the local search by selecting a new, randomly chosen initial assignment followed by local search. If after $\mathsf{maxTries}$ of such tries still no solution was found, the algorithm is aborted unsuccessfully.

GSAT can be conveniently modelled by a 1-state+init GLSM (*cf.* Section 3.5) using conditional deterministic transitions. The init state RI of this GLSM just selects a random assignment:

If F is the given formula, for all assignments $a \in \mathit{Assign}(F)$, we define $\pi_{\mathsf{RI}}(a) := D$,

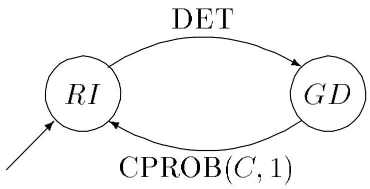


Figure 4.1: GLSM realising GSAT; transition condition $C \equiv$ mcount(maxSteps).

```
function RI(a) is
  for 1 ≤ i ≤ n do
    a'(i) := draw({⊤, ⊥});
  end for;
  return (a');
end RI;
```

```
function GD(a) is
  A' := {a' | N(a, a')};
  s' := max({score(a') | a' ∈ A'});
  A'' := {a' ∈ A' | score(a') = s'};
  a' := draw(A'');
  return (a');
end GD;
```

Figure 4.2: GLSM states for realising basic GSAT.

where $\forall a' \in \mathit{Assign}(F) : D(a') = 2^{-n}$ and $n = \#\mathit{Var}(F)$ is the number of propositional variables appearing in F.

The local search state GD models the basic local search steps in GSAT and can be defined in the following way: If ϕ is the given formula and a the current assignment, then let $A' = \{a' \mid N(a, a')\}$ be the set of assignments which are neighbours of a and $s' = \mathit{max}(\{\mathit{score}(a') \mid a' \in A'\})$ the maximal score within this set. Now, we define $A'' = \{a' \in A' \mid \mathit{score}(a') = s'\}$, the set of neighbouring assignments with maximal score. Based on A'', we finally define the state semantics for state type GD:

For all $a \in \mathit{Assign}(F)$, we define $\pi_{\mathsf{GD}}(a) := D$, where for all $a' \in \mathit{Assign}(F)$, $D(a') = 1/\#A''$ if $a' \in A''$, and $D(a') = 0$, otherwise.

Now, the GSAT algorithm is obtained by connecting these states in such a way, that RI is always directly followed by GD, while we switch from GD to RI after maxSteps steps. This is achieved by introducing a deterministic transition with condition $\top$ from RI to GD, and a deterministic transition with condition $C \equiv$ mcount(maxSteps) from GD to RI (*cf.* Figure 4.1). According to our convention from Chapter 3, we do not explicitly specify the self-transitions (GD,GD) and (RI,RI) since they are uniquely determined by the remaining transitions.

Note that the above definition of the state types is mathematically precise and reflects the requirements of GLSM semantics, as introduced in Chapter 3, Section 3.3. At the same time,

```
function RW(a) is
   C' := the set of all currently unsatisfied clauses;
   c' := draw(C');
   V' := Var(c');
   x := draw(V');
   a' := a with x flipped;
   return (a');
end RW;
```

Figure 4.3: GLSM state for realising random walk.

however, it is often cumbersome to use this kind of definition, while procedural definitions are more concise. Using the *draw* function from Chapter 3, Section 3.3, the two GSAT states can be defined as shown in Figure 4.2. Although these procedural descriptions can be implemented in a rather straightforward way, they are generally very inefficient. The same holds, of course, for the equivalent purely procedural representation, as it can be found in [SLM92] and many subsequent studies. When implementing GSAT this way, the local search is drastically slowed down, when compared to Kautz' and Selman's implementation which is commonly used when comparing GSAT with other SAT algorithms. The key to efficiently implementing GSAT is to evaluate the complete set of scores only once at the beginning of each try, and then after each flip to update only those scores which were possibly affected by the flipped variable. Details on these implementation issues for GSAT and related algorithms can be found in [Hoo96b].

4.2.2 GSAT with Random Walk (GWSAT)

An important extension of the basic GSAT algorithm is GSAT with Random Walk (GWSAT) [SKC94]. Here, besides the GD steps, (as defined above) a second type of local search steps, the so-called *random walk steps*, are introduced. In a random walk step, first a currently unsatisfied clause c' is randomly selected. Then, one of the variables appearing in c' is flipped, thus effectively forcing c' to become satisfied. The basic idea of GWSAT is to decide for each local search step with a fixed probability wp whether to do a standard GSAT step or a random walk step.

We model GWSAT as a 2-state+init GLSM, which is obtained by extending the GSAT GLSM by a new state type RW, the definition of which is shown in Figure 4.3. Between the GD state and the RW state we introduce a pair of probabilistic transitions with probabilities wp and 1-wp respectively; for technical reasons these have to be conditioned such that whenever a restart is due, they cannot be executed. To be precise, we also have to replace the transitions between RI and GD such that the overall structure shown in Fig. 4.4 is obtained.

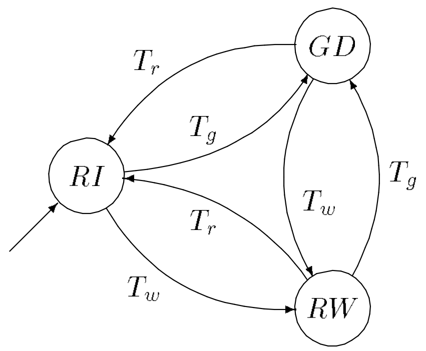


Figure 4.4: GLSM realising GWSAT; transition types: $T_r \equiv \text{CPROB}(C, 1)$ with condition $C \equiv \mathsf{mcount}(\mathsf{maxSteps}); T_w \equiv \text{CPROB}(\neg C, \mathsf{wp}); T_g \equiv \text{CPROB}(\neg C, \mathsf{1} - \mathsf{wp})$.

4.2.3 GSAT with Tabu Search (GSAT/TABU)

Another important extension of basic GSAT is achieved by introducing a tabu list with maximal length tl [MSG97]. Upon initialisation of the local search, this list is empty. After each local search step, the flipped variable is added to the head of this tabu list. If by doing so the maximal length tl is exceeded, the tail element is removed from the list. When selecting a variable x to flip, the variables in the tabu list are not considered such that effectively, after x is flipped, it becomes clamped to its current value for the next tl steps.

Formally, GSAT/TABU is realised just as GSAT with the GD state being replaced by a GDT state which implements the modified local search steps using the tabu list. Note that in the GLSM model, the tabu list is a global data structure which is initialised either within the initialisation state (a modified RI state), or using a transition action reset-tabu for the transition leading from the standard RI into the GDT state.

Like HSAT [GW93b], another GSAT variant where the local search steps make use of history information, GSAT/TABU shows generally superior performance when compared to plain GSAT. It has also been claimed that GSAT/TABU outperforms GWSAT [SSS97]; however, in the light of the results presented in this and the following chapter, this is generally not true.

4.3 The WalkSAT Architecture

The WalkSAT architecture is based on ideas first published by Selman, Kautz, and Cohen in 1994 [SKC94] and was later formally defined as an algorithmic framework by McAllester, Selman, and Kautz in 1997 [MSK97]. It is based on a 2-stage variable selection process focused on the variables ocurring in currently unsatisfied clauses. For each local search step, in a first stage a currently unsatisfied clause c' is randomly selected. In a second step, one of

```
function WS(a) is
  % stage 1: clause selection:
  C' := the set of all currently unsatisfied clauses;
  c' := draw(C');
  % stage 2: variable selection:
  s' := max({score_b(x) | x ∈ Var(c')});
  if s' = 0 then
    V' := {x ∈ Var(c') | score_b(x) = s'};
  else
    with probability wp do
      V' := Var(c');
    otherwise
      V' := {x ∈ Var(c') | score_b(x) = s'};
    end with;
  end if;
  x := draw(V');
  a' := a with x flipped;
  return (a');
end WS;
```

Figure 4.5: GLSM state for realising basic WalkSAT.

the variables appearing in c' is then selected and flipped to obtain the new assignment. Thus, while the GSAT architecture is characterised by a static neighbourhood relation between assignments with Hamming distance one, using this procedure, WalkSAT algorithms are effectively based on a dynamically determined subset of the GSAT neighbourhood relation. Another major difference between GSAT and the original WalkSAT algorithm is the scoring function used to choose the variable within the selected clause. While in GSAT, the score of a variable x is defined as the difference in the number of unsatisfied clauses before and after flipping it, WalkSAT counts only the number of clauses which are broken — *i.e.*, which are currently satisfied, but will become unsatisfied by the flip. We denote the negative value of this score by $score_b(x)$;[1] however, as we will see, WalkSAT variants are based on different scoring functions.

4.3.1 WalkSAT

WalkSAT, as originally introduced in [SKC94], is characterised by the following variable selection scheme: If there is a variable with $score_b(x) = 0$ in the clause c' selected in stage 1, *i.e.*, if c' can be satisfied without breaking another clause, this variable is flipped. If no such variable exists, with a certain probability wp the variable with maximal $score_b$ value is selected; in the remaining cases, one of the variables from c' is randomly selected.

[1] We use the negative value for technical reasons; this way, as for GSAT, maximal scores are most desirable when selecting the variable to be flipped.

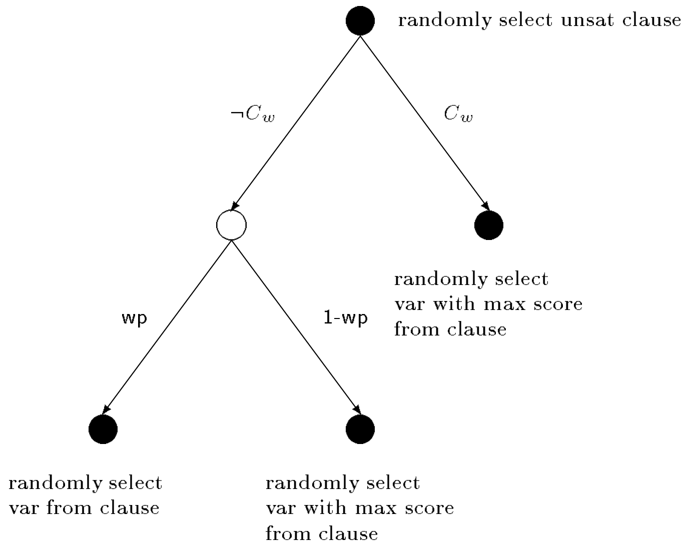


Figure 4.6: Decision tree representation for GLSM state WS; condition $C_w \equiv \exists x \in Var(c') : score_b(x) = 0$, where c' is the selected clause.

The WalkSAT algorithm can be modelled as a 1-state+init GLSM, using the same GLSM model as for GSAT, where the GD state is replaced by a suitably defined WS state which can be defined as shown in Figure 4.5. This procedural description can be represented more adequately using a probabilistic decision tree (*cf.* Figure 4.6). Conceptually as well as historically, WalkSAT is quite closely related to GWSAT. However, there is a number of significant differences between both algorithms, which in combination account for the generally superior performance of WalkSAT. First of all, there is a very obvious difference in GLSM structure: The GLSM representing GWSAT has one more state. On the other hand, WalkSAT's WS state is considerably more complex than both the GD and the RW states constituting GWSAT.

While both algorithms use the same kind of random walk steps, WalkSAT executes them in a conditional probabilistic way, namely only when there is no greedy step that would not break any currently satisfied clause. In GWSAT, however, random walk steps are done in an unconditional probabilistic way. From this point of view, WalkSAT is greedier, since random walk steps, which usually increase the number of unsatisfied clauses, are only done when each variable occurring in the selected clause would break some clauses when flipped. The 2-stage selection scheme that was already used in GWSAT's RW state is also used in WalkSAT's WS state, which can actually be seen as a refined version of RW. Because for WalkSAT, the first stage of the selection process (selecting an unsatisfied clause) is purely random, and clauses are typically rather short, even for small walk probabilities, WalkSAT will generally be less greedy than GWSAT in the sense, that the former chooses from a significantly reduced set of neighbours.

Finally, WalkSAT uses a different scoring function than GWSAT: while the latter is based on the total difference in the number of unsatisfied clauses caused by a local search step, the former counts only the clauses which are broken. Here, in some sense, GWSAT shows a greedier behaviour: In its GD state, it would prefer a variable which breaks some clause but compensates for this by fixing some other clauses, while in the same situation, WalkSAT would select a variable whith a smaller total score, but breaking also a smaller number of clauses.

4.3.2 WalkSAT Variants

There are three variants of WalkSAT which we want to address here briefly. The first one, called WalkSAT/G [MSK97, PW96], is different from WalkSAT as introduced above, in that it uses the same scoring function as GSAT. Furthermore, a different variable selection process is used in the second stage of each step: with a fixed probability wp, WalkSAT/G chooses a random variable from the selected clause, otherwise the best variable (according to its scoring function), is chosen. Note that this variant is very closely related to GWSAT; actually, the only difference is the two-stage selection process where the unconditional choice between walk and greedy steps is made in the local context of the selected clause. The second variant, WalkSAT/B differs from WalkSAT/G only in its scoring function, which is $score_b$ just as for WalkSAT. These two variants are of interest, because they are conceptionally in some sense "between" GWSAT and WalkSAT. Interestingly, the same relation holds for their performance on standard benchmark problem classes: Generally, WalkSAT is superior to WalkSAT/B, which in turn shows better performance than WalkSAT/G. GWSAT's performance is roughly in between WalkSAT/G's and WalkSAT/B's.[2]

A third variant, which is called WalkSAT/TABU [MSK97], combines features of WalkSAT/G and GSAT/TABU. Like for GSAT/TABU, a constant length tabu list of variables is used. According to the two level variable selection scheme which is generally used for WalkSAT algorithms, it may however happen that all variables appearing in the selected clause are tabu. In this case, no variable is flipped (a so-called *null-flip*). Like in the case of GSAT, adding tabu search to the basic WalkSAT algorithm results in a significantly improved performance, as we will show in Section 4.5.

4.3.3 Novelty

The Novelty algorithm, as introduced in [MSK97], is one of the latest SLS algorithms for SAT. Conceptually, it combines features of WalkSAT/G with a history-based variable selection mechanism in the spirit of HSAT [GW93b]. Novelty, too, is based on the intuition, that repeatedly flipping back and forth the same variable should be avoided. Interestingly, Novelty uses a different scoring function from WalkSAT: like GSAT, it selects the variable

[2] Although to our best knowledge direct comparisons have never been done, this can be conjectured indirectly from the data given in [MSK97] and the experimental results reported later in this chapter.

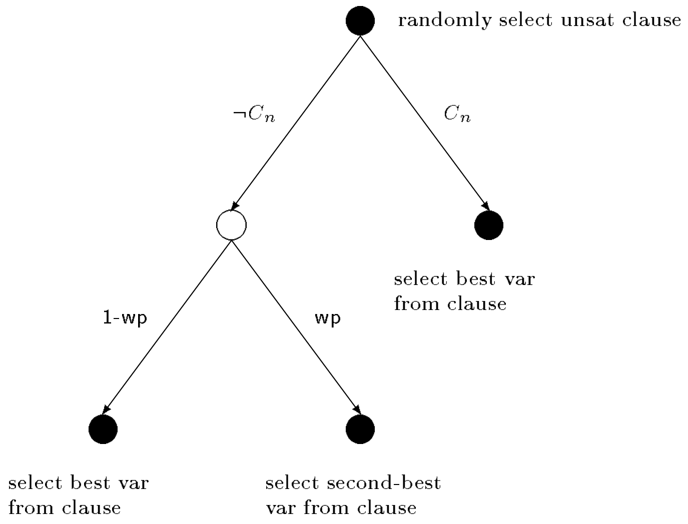


Figure 4.7: Decision tree representation for GLSM state NOV; condition $C_n \equiv$ "best var does not have minimal age"; "best" and "second-best" relate to GSAT-score of variables.

to be flipped within the selected clauses based on the difference in the number of unsatisfied clauses caused by the corresponding local search step. Additionally, like for tabu search variants, the number of local search steps which have been performed since it was last flipped (also called the variable's *age*) is taken into consideration.

In Novelty, after an unsatisfied clause has been chosen, the variable to be flipped is selected as follows. If the variable with the highest score does not have minimal age among the variables within the same clause, it is always selected. Otherwise, it is only selected with a probability of 1-wp; in the remaining cases, the variable with the next lower score is selected. In Kautz' and Selman's implementation, if there are several variables with identical score, always the first of them is chosen. Novelty can be realised as a GLSM exactly like WalkSAT, where only the WS state is replaced by a NOV state which captures the variable selection mechanism as defined by the decision tree given in Figure 4.7.

Note that for wp > 0, the age-based variable selection of Novelty probabilistically prevents flipping the same variable over and over again; at the same time, flips can be immediately reversed with a certain probability if a better choice is not available. Generally, the Novelty algorithm is significantly greedier than WalkSAT, since always one of the two most improving variables from a clause is selected, where WalkSAT may select any variable if no improvement without breaking other clauses can be achieved. Because of this property, it is to be expected that Novelty might get into trouble when applied to formulae with longer clauses, where selecting only among the best two variables in a given clause can lead to situations where the algorithm gets stuck in local minima of the objective function. Also, Novelty is more deterministic than WalkSAT and GWSAT, since its probabilistic decisions

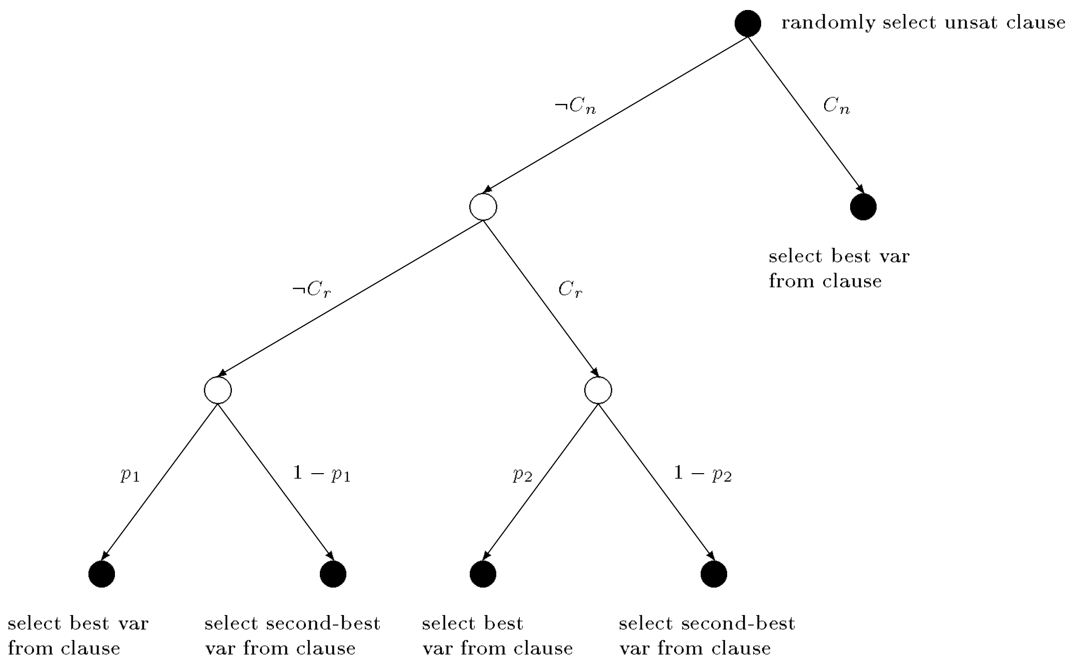


Figure 4.8: Decision tree representation for GLSM state RNOV; condition $C_n \equiv$ "best var does not have minimal age"; condition $C_r \equiv$ "score diff between best and second best var" > 1; "best" and "second-best" relate to GSAT-score of variables; $p_1 = \max\{1 - 2p, 0\}$, $p_2 = \min\{2 - 2p, 1\}$.

are more limited in their scope and take place under more restrictive conditions.[3] On one hand side, this often leads to a significantly improved performance of Novelty when compared to WalkSAT. On the other hand, as we will see later, Novelty occasionally gets stuck in local minima which severely compromises its performance.

4.3.4 R-Novelty

R-Novelty, also introduced in [MSK97], is a variant of Novelty which is based on the intuition that, when deciding between the best and second best variable (w.r.t. the same score function as for Novelty), the actual difference of the respective scores should be taken into account. The exact mechanism for choosing a variable from the selected clause can be seen from the decision tree representation given in Figure 4.8. Note that the R-Novelty heuristic is quite complex – as reported in [MSK97], it was discovered by systematically testing a large number of WalkSAT variants.

[3] Interestingly, different from WalkSAT, the Novelty strategy for variable selection within a clause is deterministic for both $\mathsf{wp} = 0$ and $\mathsf{wp} = 1$.

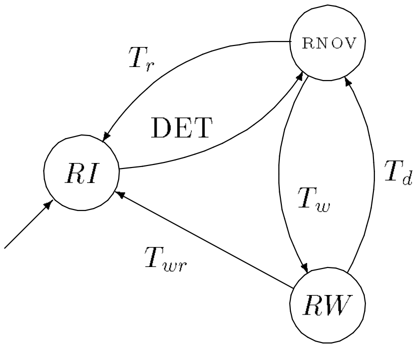


Figure 4.9: GLSM realising R-Novelty; transition types: $T_r \equiv \text{CPROB}(C_r, 1)$, $T_w \equiv \text{CPROB}(C_w \wedge \neg C_r, 1)$, $T_d \equiv \text{CPROB}(\neg C_r, 1)$, $T_{wr} \equiv \text{CPROB}(C_r, 1)$ with conditions $C_r \equiv \mathsf{mcount(maxSteps)}$, $C_w \equiv \mathsf{mcount(100)}$.

R-Novelty's variable selection strategy is even more deterministic than Novelty's; in particular, it is completely deterministic for $\mathsf{wp} \in \{0, 0.5, 1\}$. Since the pure R-Novelty algorithm gets too easily stuck in local minima, a simple loop breaking strategy is used: every 100 steps, a variable is randomly chosen from the selected clause and flipped. Thus, R-Novelty is a hybrid strategy which can be modelled using the 2-state+init GLSM shown in Figure 4.9. However, it is questionable whether one random walk step every 100 steps is sufficient for effectively escaping from local mininima. Despite being more determinstic than the other WalkSAT variants, the R-Novelty algorithm is still very sensitive with respect to the wp parameter. Although there is some indication that R-Novelty shows superior performance on several problem classes [MSK97], it is not clear whether this algorithm generally improves on Novelty.

4.4 The Benchmark Set

The benchmark suite we are using as a basis for the empirical evaluation of SLS algorithms for SAT comprises three different types of problems: test-sets sampled from Random-3-SAT, a well-known random problem distribution; test-sets obtained by encoding instances from a random distribution of hard Graph Colouring instances into SAT; and SAT-encoded instances from two problem domains which are of a certain practical interest, the Blocks World Planning problem, and the All-Interval-Series problem. All these benchmark instances are hard in general and difficult to solve for SLS algorithms. For the SAT-encoded problems, the hardness of the instances is inherent rather than just induced by the encoding scheme that was used for transforming them into SAT. The SAT-encodings used here are mostly very simple and well-known from the literature. In the following, we will introduce the benchmark problems and give some background on them as well as a description of how they were generated. All benchmark instances are available from the SATLIB website (`www.informatik.tu-darmstadt.de/AI/SATLIB`).

4.4.1 Uniform Random-3-SAT

Uniform Random-3-SAT is a family of SAT instance distributions obtained by randomly generating 3-CNF formulae in the following way: For an instance with n variables and k clauses, each of the k clauses is constructed from 3 literals which are randomly drawn from the $2n$ possible literals (the n variables and their negations) such that each possible literal is selected with the same probability of $1/2n$. Clauses are not accepted for the construction of the problem instance if they contain multiple copies of the same literal or if they are tautological (*i.e.*, they contain a variable and its negation). Each choice of n and k thus induces a distribution of Random-3-SAT instances. Uniform Random-3-SAT is the union of these distributions over all n and k. Generally, Random-3-SAT instances can be satisfiable or unsatisfiable. Since in the context of SLS algorithms for SAT it makes no sense to include insoluble instances in a benchmark suite, the insoluble instances have to be filtered out using a complete algorithm. As complete SAT algorithms have usually considerable higher run-times than (incomplete) SLS algorithms, this approach for generating benchmark instances is computationally very expensive, especially when dealing with larger problem sizes. Therefore, both the size of the instances and the number of instances in the test-sets are effectively limited by the complete algorithms used for filtering.

One particularly interesting property of uniform Random-3-SAT is the occurrence of a phase transition phenomenon, *i.e.*, a rapid change in solubility which can be observed when systematically increasing (or decreasing) the number k of clauses for fixed problem size n [MSL92, KS94]. More precisely, for small k, almost all formulae are satisfiable; at some critical $k = k^*$, the probability of generating a satisfiable instance drops sharply to almost zero. Beyond k^*, almost all instances are unsatisfiable. Intuitively, k^* characterises the transition between a region of underconstrained instances which are almost certainly soluble, to overconstrained instances which are mostly insoluble. For Random-3-SAT, this phase transition occurs approximately at $k^* = 4.26n$ for large n; for smaller n, the critical clauses/variable ratio k^*/n is slightly higher [MSL92, CA93, CA96]. Furthermore, for growing n the transition becomes increasingly sharp.

The phase transition would not be very interesting in the context of evaluating SLS algorithms, did not empirical analyses show that problem instances from the phase transition region of uniform Random-3-SAT tend to be particularly hard for both systematic SAT solvers [CKT91] and SLS algorithms [Yok97]. Striving for testing their algorithms on hard problem instances, many researchers used test-sets sampled from the phase transition region of uniform Random-3-SAT (see [GW93b, MSG97, MSK97] for some examples). Although similar phase transition phenomena have been observed for other subclasses of SAT, including uniform Random-k-SAT with $k \geq 4$, these have never gained the popularity of uniform Random-3-SAT. Maybe, one of the reasons for this is the prominent role of 3-SAT as a prototypical and syntactically particularly simple $\mathcal{NP}$-complete problem.

test-set	instances	clause-len	vars	clauses
uf50-218	1,000	3	50	218
uf100-430	1,000	3	100	430
uf200-860	100	3	200	860

Table 4.1: Uniform Random-3-SAT test-sets.

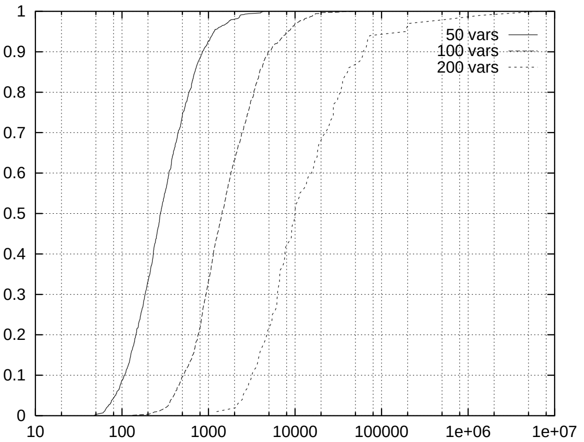


Figure 4.10: Hardness distributions for Random-3-SAT test-sets. The x-axis shows the median search cost (number of local search steps) for WalkSAT per instance.

For evaluating and characterising the behaviour of SLS algorithms, in this and the following chapters we use three test-sets from the phase transition region of uniform Random-3-SAT. The characteristics of these test-sets are shown in Table 4.1. Due to limited computational resources, the test-sets `uf200-860` consists of only 100 instances, while the other test-sets contain 1,000 instances each.

To give an impression of the variability of the hardness of the problem instances across the test-sets, we measured the median number of local search steps (median search cost) for each instance, using WalkSAT with an approximately optimal[4] noise parameter as a reference algorithm. As we will see in Section 4.5, there is a notion of intrinsic hardness to which this measure correlates strongly. Figure 4.10 shows the cumulative distributions of the median search cost per instance (measured in local search steps) across the 50, 100, and 200 variable test-sets. For each problem instance, the median search cost was determined from an RLD obtained by running WalkSAT with an approximately optimal noise parameter (walk probability) of $\mathsf{wp} = 0.55$ for 1,000 tries. By using an extremely high cutoff parameter (*i.e.*, maxSteps) we could ensure that each instance was solved in every single try. This is

[4] As an optimality criterion we used the expected number of steps required for finding a solution.

test-set	mean	median	stddev	stddev/mean
uf50-218	429.56	277	479.71	1.12
uf100-430	2,507.5	1,433	3,580.3	1.43
uf200-860	106,440	10,032	664,100	6.24

Table 4.2: Random-3-SAT test-sets, basic statistics for hardness distributions.

possible, because WalkSAT with the given noise parameter (*i.e.*, wp or tl, resp.) shows *approximately complete* behaviour on all instances.

As can be seen from Figure 4.10, there is a huge variability between the instances of each test-set (see also Table 4.2). In particular, the long tails of these distributions show that a substantial part of the problem instances for each test-set is dramatically harder than the rest of the test-set. For increasing problem sizes, this phenomenon becomes more prominent, indicating that for larger instances there is a considerably higher variability in the median search cost, especially among the hardest instances from each test-set. This can also be seen from the normalised standard deviations as given in Table 4.2.

In some of the empirical studies following later in this work, we use the instances corresponding to the minimum, median, and maximum of these hardness distributions; these are referred to as the "easy", "medium" (med), and "hard" hardness instances from the underlying test-set.

4.4.2 Graph Colouring

The Graph Colouring problem (GCP) is a well-known combinatorial problem from graph theory: Given a graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the set of vertices and $E \subseteq V \times V$ the set of edges connecting the vertices, find a colouring $C : V \mapsto \mathbb{N}$, such that neighbouring vertices always have different colours. There are two variants of this problem: In the optimisation variant, the goal is to find a colouring with a minimal number of colours, whereas in the decision variant, the question is to decide whether for a particular number of colours, a couloring of the given graph exists. The two variants are tightly related, as optimal colourings can be found by solving a series of decision problems, using, for instance, binary search to determine the minimal number of colours. In the context of SAT-encoded Graph Colouring problems, we focus on the decision variant.

The Graph Colouring problem can be easily reformulated as a CSP: The colouring of each vertex of the given graph is represented by a constraint variable; the domain of these variables is $\mathbb{Z}_k = \{0, 1, 2, \ldots, k-1\}$, where k is the number of colours; and for each pair of vertices connected by an edge there is a constraint ensuring that they are coloured differently. This is formalised in the following way:

test-set	instances	vertices	edges	colours	vars	clauses
flat30-60	100	30	60	3	90	300
flat50-115	1,000	50	115	3	150	545
flat100-239	100	100	239	3	300	1,117

Table 4.3: SAT-encoded Graph Colouring test-sets (flat random graphs).

For a given graph $G = (V, E)$ and $k \in \mathbb{N}$, $GCP(G, k)$ is represented by the CSP $(X, \mathcal{D}, \mathcal{C})$, where $X = \{c_0, \ldots, c_{\#V-1}\}$, for all i, $\mathcal{D}_{c_i} = \mathbb{Z}_k$, and $\mathcal{C}$ consists of the constraints[5] $c_i = x \wedge c_j = y \iff \neg E(v_i, v_j) \vee x \neq y \vee i = j \quad (i, j \in \mathbb{Z}_n)$.

This CSP representation can be transformed into the SAT domain by using a straightforward encoding [Ben96]: Each assignment of a value to a single CSP variable is represented by a propositional variable; each constraint is represented by a set of clauses; and two additional sets of clauses ensure that valid SAT assignments assign exactly one value to each CSP variable:

For a given graph $G = (V, E)$ and $k \in \mathbb{N}$, $GCP(G, k)$ is represented as a SAT instance by the CNF formula over the propositional variables $c_{i,x}$ (where $i \in \mathbb{Z}_n$ and $x \in \mathbb{Z}_k$) consisting of the following sets of clauses:

$$
\begin{array}{lll}
(1) & \neg c_{i,x} \vee \neg c_{j,x} & (i, j \in \mathbb{Z}_n; i \neq j; \neg E(v_i, v_j); x \in \mathbb{Z}_k) \\
(2a) & \neg c_{i,x} \vee \neg c_{i,y} & (i \in \mathbb{Z}_n; x, y \in \mathbb{Z}_k; x \neq y) \\
(2b) & c_{i,0} \vee c_{i,1} \vee \ldots \vee c_{i,k-1} & (i \in \mathbb{Z}_n)
\end{array}
$$

We used Joe Culberson's random graph generator[6] for generating three sets of 3-colourable flat graphs (see Table 4.3). The connectivity (edges/vertex) of these graphs is adjusted in such a way that the instances have maximal hardness (in average) for graph colouring algorithms like the Brelaz heuristic [MJPL92].

As for Random-3-SAT, we analysed the median search cost for WalkSAT with approximately optimal noise parameter setting to characterise the variability in hardness for local search across the Graph Colouring test-sets. The basic descriptive statistics of the cumulative hardness distributions are reported in Table 4.4. Although we observe a large variability in the hardness of problem instances across the test-sets, it is somewhat lower for the more structured Graph Colouring instances than for Random-3-SAT (as can be seen by comparing the normalised standard deviations). Also, there is no clear indication that the variability in hardness increases with growing problem size, as in the case of Random-3-SAT.

[5] For clarity's sake, we are using a different notation here than in Definition 1.6; however, both are equivalent and can be easily transformed into each other.

[6] available from `http://web.cs.ualberta.ca/~joe/Coloring/index.html`, Joe Culbersons's Graph Coloring Page.

test-set	mean	median	stddev	stddev/mean
flat30-60	615.52	417.22	674.80	1.10
flat50-115	3,913.76	3,132.93	2,779.63	0.71
flat100-239	44,900.75	31,728.92	39,845.72	0.89

Table 4.4: Graph Colouring test-sets, basic statistics for hardness distributions.

Like in the case of Random-3-SAT, we will sometimes use the single instances corresponding to the minimum, median, and maximum of these hardness distributions; these are referred to as the "easy", "medium" (med), and "hard" hardness instances from the underlying test-set.

4.4.3 Blocks World Planning

The Blocks World is a very well-known problem domain in AI research. The general scenario in Blocks World Planning comprises a number of blocks and a table. The blocks can be piled onto each other, where the downmost block of a pile is always on the table. In our examples of Blocks World Planning, taken from [KS96], there is only one operator which moves the top block of a pile to the top of another pile or onto the table. Given an initial and a goal configuration of blocks, the problem is to find a sequence of operators which, when applied to the initial configuration, leads to the goal situation. Such a sequence is called a (linear) plan. Blocks can only be moved when they are clear, *i.e.*, no other block is piled on top of them, and they can only be moved on top of blocks which are clear or onto the table. If these conditions are satisfied, the move operator always succeeds. As in the case of Graph Colouring, there is an optimisation and a decision variant of the Blocks World Planning problem: In the optimisation variant, the goal is to find a shortest plan, whereas in the decision variant, the question is to decide whether a plan of a given length exists. Again, the two variants are tightly related, as shortest plans can be found by solving a series of decision problems. As all SAT-based approaches to Blocks World Planning, we focus on the decision variant.

A linear encoding strategy is used for translating Blocks World Planning instances into CNF formulae. The encoding is based on the following predicates:

- $clear(x,t)$ - block x is clear at time t;
- $on(x,y,t)$ - block x is directly on top of y at time t;
- $move(x,y,z,t)$ - block x is moved from block y on block z at time t.

clear and *on* are state predicates, *i.e.*, they are used to specify the state of the world, while *move* is an action predicate which is used to describe actions that change the state of the

instance	blocks	steps	vars	clauses
anomaly	3	3	48	261
medium	5	4	116	953
bw_large.a	9	6	459	4,675
huge	9	6	459	7,054
bw_large.b	11	9	1,087	13,772
bw_large.c	15	14	3,016	50,457
bw_large.d	19	18	6,325	131,973

Table 4.5: SAT-encoded Blocks World Planning instances (state-based encoding).

world. The axioms which specify the problem can be grouped into four categories:

- actions imply their preconditions and effects,
- exactly one action can be executed at each time t,
- classical frame conditions, which state that state predicates do not change between time t and $t+1$ if they are not directly affected by the action at time t,
- in $move(x,y,z,t)$, x, y, and z are different.

Note that the last group of actions is redundant, but has been found to be useful for speeding up local search. For a given Blocks World Planning instance, instantiating the predicates listed above gives the propositional variables over which the axioms can then be formulated as CNF clauses (for details, *cf.* [KS96]).

The SAT encoding used for generating the benchmark instances relies critically on two important techniques for reducing the size of the CNF formulae: operator splitting [KMS96] and simple propositional reductions (unit propagation and subsumption). Operator splitting replaces a predicate which takes three or more arguments by a number of binary predicates. This reduces the number of propositional variables for the given problem from $O(kn^3)$ to $O(kn^2)$ where n is the number of blocks and k the number of plan steps. Unit propagation and subsumption, two well-known propositional reduction strategies, are used to simplify the formulae before applying stochastic local search. These reducations can be computed in polynomial time and eliminate a number of propositional variables thus efficiently reducing the search space. Intuitively, by applying these strategies, the initial and goal states are propagated into the planning structure.[7]

Our benchmark set contains seven Blocks World Planning instances taken from Henry Kautz' and Bart Selman's SATPLAN distribution. These instances are described in Table 4.5; despite the reductions mentioned above, they are still very large when compared

[7] Details on the SAT encoding used to generate the benchmark instances can be found in [KS96, KMS96].

to other instances of our benchmark suit. At the time of this writing, `bw_large.c` and `bw_large.d` belong to the hardest problem instances which can be solved by state-of-the-art SAT algorithms in reasonable time.

4.4.4 All-Interval-Series

The All-Interval-Series (AIS) problem is an arithmetic problem which, to our best knowledge, is used here for the first time in the context of evaluating SLS algorithms. It is inspired by a well-known problem occurring in serial musical composition [Col40], which, in its original form, can be stated in the following way:

> Given the twelve standard pitch-classes (`c`, `c#`, `d`, ...), represented by numbers $0, 1, \ldots, 11$, find a series in which each pitch-class occurs exactly once and in which the musical intervals between neighbouring notes cover the full set of intervals from the minor second (1 semitone) to the major seventh (11 semitones), *i.e.*, for each of these intervals, there is a pair of neigbhouring pitch-classes in the series, between which this interval appears.

These musical all-interval series have a certain prominence in serial composition; they can be traced back to Alban Berg and have been extensively studied and used by Ernst Krenek, *e.g.* in his orchestral piece op.170, "Quaestio temporis" (A Question of Time) [Kre74]. The problem of finding such series can be easily formulated as an instance of a more general arithmetic problem in $\mathbb{Z}_n$, the set of integer residues modulo n:

> For given $n \in \mathbb{N}$, find a vector $s = (s_1, \ldots, s_n)$, such that (i) s is a permutation of $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$; and (ii) the interval vector $v = (|s_2 - s_1|, |s_3 - s_2|, \ldots |s_n - s_{n-1}|)$ is a permutation of $\mathbb{Z}_n - \{0\} = \{1, 2, \ldots, n-1\}$. A vector v satisfying these conditions is called an *all-interval series of size n*; the problem of finding such a series is called the *All-Interval-Series Problem of size n* and denoted by $AIS(n)$.

In this form, the problem can be represented as a Constraint Satisfaction Problem in a rather straightforward way. Each element of s and v is represented as a variable; the domains of these variables are $\mathbb{Z}_n$ and $\mathbb{Z}_n - \{0\}$, respectively; and the contraint relations encode the conditions (i) and (ii) from the definition given above. Formally, the corresponding CSP is defined in the following way:

> For a given $n \in \mathbb{N}$, $AIS(n)$ is represented by the Constrained Satisfaction Problem $(X, \mathcal{D}, \mathcal{C})$, where $X = \{s_0, \ldots, s_{n-1}, v_1, \ldots v_{n-1}\}$; for all k, $\mathcal{D}_{s_k} = \mathbb{Z}_n$ and $\mathcal{D}_{v_k} = \mathbb{Z}_n - \{0\}$; and $\mathcal{C}$ consists of the following constraints:[8]

[8] For clarity's sake, we are again using a different notation here than in Definition 1.6; however, both are equivalent and can be easily transformed into each other.

$$
\begin{array}{lll}
(1) & s_i = x \wedge s_k = y \iff x \neq y \vee i = k & (i, k \in \mathbb{Z}_n) \\
(2) & v_i = x \wedge v_k = y \iff x \neq y \vee i = k & (i, k \in \mathbb{Z}_n - \{0\}) \\
(3) & s_{i-1} = x \wedge s_i = y \wedge v_i = z \iff |x - y| = z & (i, k \in \mathbb{Z}_n - \{0\})
\end{array}
$$

From this CSP formulation, $AIS(n)$ can be easily transformed into a SAT instance by using essentially the same encoding as for the Graph Colouring Problem. Again, each assignment of a value to a single CSP variable is represented by a propositional variable; each of the three sets of constraints from the definition above is represented by a set of clauses; and two additional sets of clauses ensure that valid SAT assignments assign exactly one value to each CSP variable:

For a given $n \in \mathbb{N}$, $AIS(n)$ is represented as a SAT instance by the CNF formula over the propositional variables $s_{i,x}, v_{j,y}$ (where $i, x \in \mathbb{Z}_n$ and $j, y \in \mathbb{Z}_n - \{0\}$) consisting of the following sets of clauses:

$$
\begin{array}{lll}
(1) & \neg s_{i,x} \vee \neg s_{k,x} & (i, k, x \in \mathbb{Z}_n; i \neq k) \\
(2) & \neg v_{i,x} \vee \neg v_{k,x} & (i, k, x \in \mathbb{Z}_n - \{0\}; i \neq k) \\
(3) & \neg s_{i-1,x} \vee \neg s_{i,y} \vee v_{i,z} & (x, y \in \mathbb{Z}_n; i, z \in \mathbb{Z}_n - \{0\}; |x - y| = z) \\
(4a) & \neg s_{i,x} \vee \neg s_{i,y} & (i, x, y \in \mathbb{Z}_n; x \neq y) \\
(4b) & s_{i,0} \vee s_{i,1} \vee \ldots \vee s_{i,n-1} & (i \in \mathbb{Z}_n) \\
(5a) & \neg v_{i,x} \vee \neg v_{i,y} & (i, x, y \in \mathbb{Z}_n - \{0\}; x \neq y) \\
(5b) & v_{i,1} \vee v_{i,2} \vee \ldots \vee v_{i,n-1} & (i \in \mathbb{Z}_n - \{0\})
\end{array}
$$

instance	n	vars	clauses
ais6	6	61	581
ais8	8	113	1,520
ais10	10	181	3,151
ais12	12	265	5,666

Table 4.6: SAT-encoded All-Interval-Series instances.

Using this encoding scheme, we generated four SAT instances `ais6`, `ais8`, `ais10`, and `ais12` corresponding to $AIS(n), n = 6, 8, 10, 12$. Table 4.6 shows the numbers of variables and clauses for these CNF formulae. As we will later see, these problem instances are extremely hard to solve for state-of-the-art SLS-based SAT algorithms; therefore we did not include instances with $n > 12$ in our benchmark suite.

4.5 Comparing SLS Algorithms

In this section, we study the performance of the SLS algorithms introduced before applied to our suite of benchmark problems. We use two basic methods: For individual instances, we

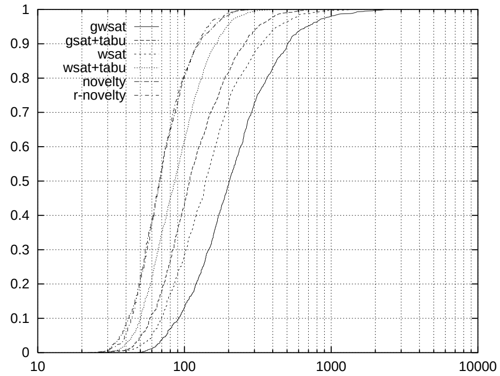


Figure 4.11: Problem instance `uf100-430/easy`, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

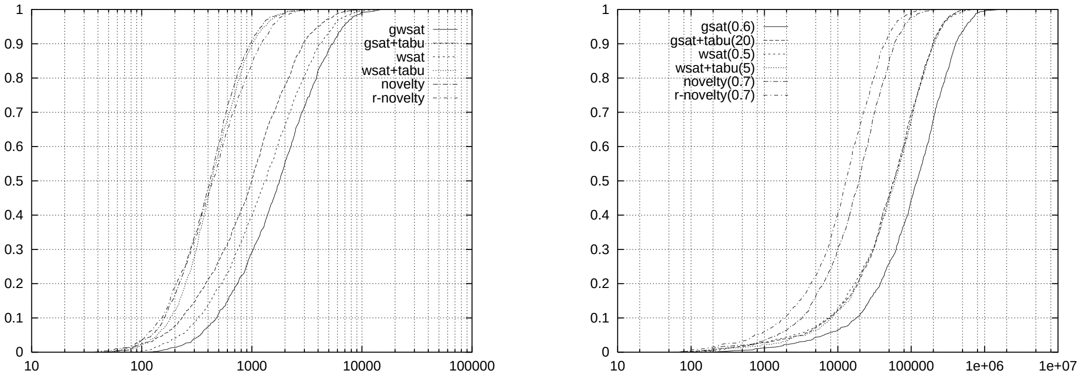


Figure 4.12: Problem instance `uf100-430/med` *(left)* and `uf100-430/hard` *(right)*, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries / instance.

compare the RLDs of the algorithms using approximately optimal noise parameter settings. As an optimality criterion we used the expected number of steps required for finding a solution, as defined in Chapter 2, Section 2.3. For test-sets sampled from random instance distributions, we analyse the correlation between the mean search cost per instance for different algorithms, again using approximately optimal noise settings.

4.5.1 Random-3-SAT

For Uniform Random-3-SAT, we first empirically analysed SLS performance on the hard, medium, and easy problems from the test-set `uf100-430`. For comparing the algorithms' performance, we determined the RLD for each algorithm using approximately optimal noise settings (walk probability / length of tabu list). The RLDs are shown in Figures 4.11 and 4.12; the corresponding descriptive statistics are given in Tables 4.7–4.9, which also indicate the optimal noise settings (in parentheses behind algorithms' names). As can be seen from

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat(0.5)	270.63	240.62	0.89	202	133	313	91	504	2.35	5.54
gsat+tabu(20)	137.78	91.31	0.66	108	78	170	58	255	2.18	4.40
wsat(0.5)	177.86	135.78	0.76	139	93	206	69	336	2.22	4.87
wsat+tabu(5)	97.79	49.04	0.50	86	62	119	50	164	1.92	3.28
novelty(0.9)	77.36	37.29	0.48	68	52	92	41	124	1.77	3.02
r-novelty(0.9)	76.63	34.79	0.45	68	53	90	43	121	1.70	2.81

Table 4.7: Problem instance `uf100-430/easy`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat(0.5)	2,432.03	2,159.87	0.89	1,785	886	3,345	472	5,434	3.78	11.51
gsat+tabu(20)	1,368.77	1,297.45	0.95	995	471	1,804	230	2,903	3.83	12.62
wsat(0.5)	1,877.78	1,776.33	0.95	1,333	644	2,510	322	4,133	3.90	12.84
wsat+tabu(3)	532.96	365.34	0.69	433	283	689	181	999	2.43	5.52
novelty(0.6)	504.77	372.03	0.74	416	247	647	157	962	2.62	6.13
r-novelty(0.7)	580.21	467.04	0.80	440	248	767	153	1,213	3.09	7.93

Table 4.8: Problem instance `uf100-430/med`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat(0.6)	177,468.78	181,843.19	1.02	119,666	46,919	247,047	17,577	400,466	5.27	22.78
gsat+tabu(20)	84,578.80	84,301.05	1.00	57,778	23,993	120,286	7,874	196,493	5.01	24.95
wsat(0.5)	86,773.44	90,538.08	1.04	56,666	22,297	120,583	7,319	198,109	5.41	27.07
wsat+tabu(5)	87,031.08	86,692.41	1.00	60,019	23,207	119,246	7,884	206,822	5.14	26.23
novelty(0.7)	26,995.92	27,165.22	1.01	19,434	8,346	36,972	3,248	58,661	4.43	18.06
r-novelty(0.7)	19,118.53	19,827.00	1.04	12,819	5,669	26,707	1,880	43,911	4.71	23.36

Table 4.9: Problem instance `uf100-430/hard`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

the figures, the RLDs for the different algorithms have all roughly the same shape, and, with one exception (R-Novelty and WalkSAT+tabu on the medium instance), no cross-overs occur. This means that generally, there is a dominance relation between the algorithms when applied to the same instance, *i.e.*, when comparing two algorithms' performance, one is uniformly better or worse than the other (*cf.* Chapter 2, Section 2.2.2). Given this situation, it is admissible to base performance comparisons mainly on descriptive statistics, as given in Tables 4.7–4.9.

As a first and general observation, we note that for all algorithms, the standard deviation of the RLDs is very similar to the mean, *i.e.*, there is a huge variance in the length of different runs of the same algorithm. Interestingly, as can be seen when comparing the stddev/mean values between the easy, medium, and hard instance, the relative variance of the RLDs increases with growing instance hardness; the same tendency can be observed for the percentile ratios. This indicates that for harder problem instances, the variability in the length of the algorithms' runs is even higher. In the semi-log plots of the RLDs (Figures 4.11 and 4.12), this is reflected by the fact that the distribution curves get less steep with increasing problem hardness.

Since the RLDs do not cross over (except in one case, which will be discussed separately), for comparing the algorithms' performance on these three problem instances, we can just compare the mean or median of the corresponding RLDs, as given in Tables 4.7–4.9. Doing this, we find the following situation: For the easy problem instance, Novelty and R-Novelty give the best performance, followed by WalkSAT+tabu, GSAT+tabu, WalkSAT and GWSAT. For the medium instance, Novelty, R-Novelty, and WalkSAT+tabu all give similarly good performance, followed by GSAT+tabu, WalkSAT, and GWSAT. Here, the only instance of crossing RLDs occurs; while R-Novelty shows better performance than WalkSAT+tabu for short runs, for long runs the situation is reversed. For the hard problem instance, R-Novelty shows the best performance, followed by Novelty; next are WalkSAT+tabu, WalkSAT, and GSAT+tabu, the performance of which is roughly identical, and finally GWSAT, which shows a substantially weaker performance.

It may be noted that the differences in performance between the best and the worst algorithm is far more dramatic for the hard instance (a factor of approx. 10 in the median) than for the medium (factor ≈ 4 in the median) and the easy instance (factor ≈ 3 in the median). For the hard instance, the factor between best and worst performance is roughly identical for all percentiles, which is manifested as a horizontal shift of the RLDs in a semi-log plot. For the medium and easy instance, the factor is larger for the higher percentiles; at the same time, there is a positive correlation between the variance of the RLDs and their mean (or median). This indicates that while for easy instances and short runs, the differences between the SLS methods are not too substantial, for longer runs or when applied to hard instances, the performance differences between the algorithms become much more obvious.

To generalise and refine these results, we did a pairwise hardness correlation analysis for the same algorithms across the whole test-set `uf100-430`. As a measure for hardness we chose the expected number of steps required for finding a solution, based on 100 tries with a high

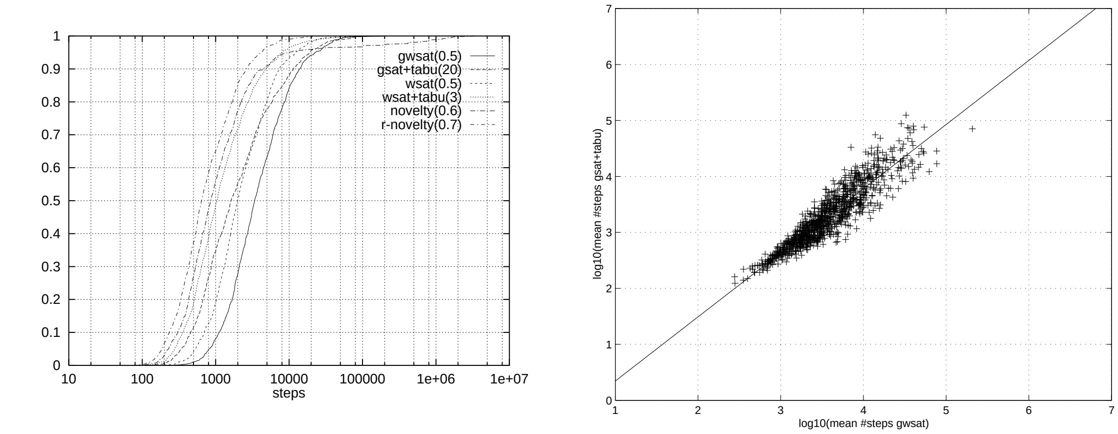


Figure 4.13: *Left:* Hardness distributions across test-set `uf100-430` for various algorithms; x-axis: median number of local search steps / solution. *Right:* Correlation between average local search cost for GWSAT (horizontal) and GSAT+tabu (vertical) for test-set `uf100-430`, using approx. optimal noise.

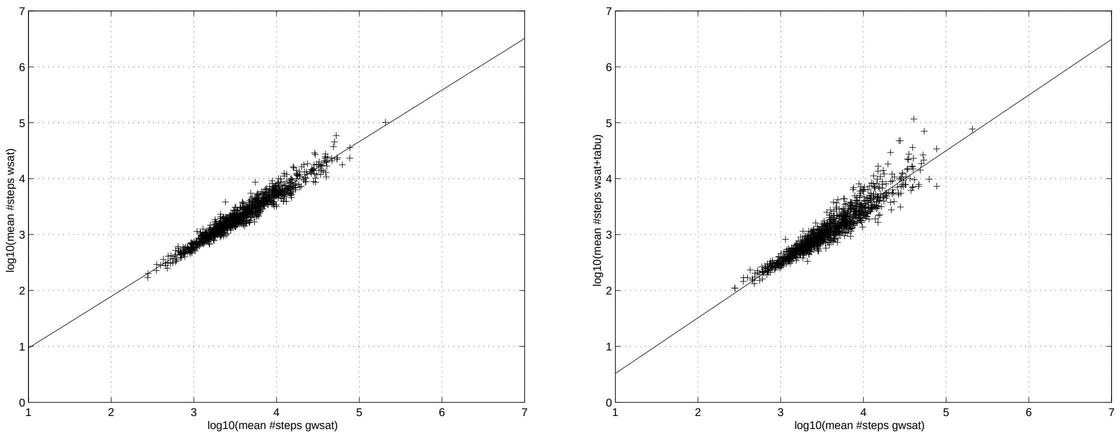


Figure 4.14: *Left:* Correlation between average local search cost for GWSAT (horizontal) and GSAT+tabu (vertical) for test-set `uf100-430`, using approx. optimal noise. *Right:* Same for GWSAT and WalkSAT+tabu.

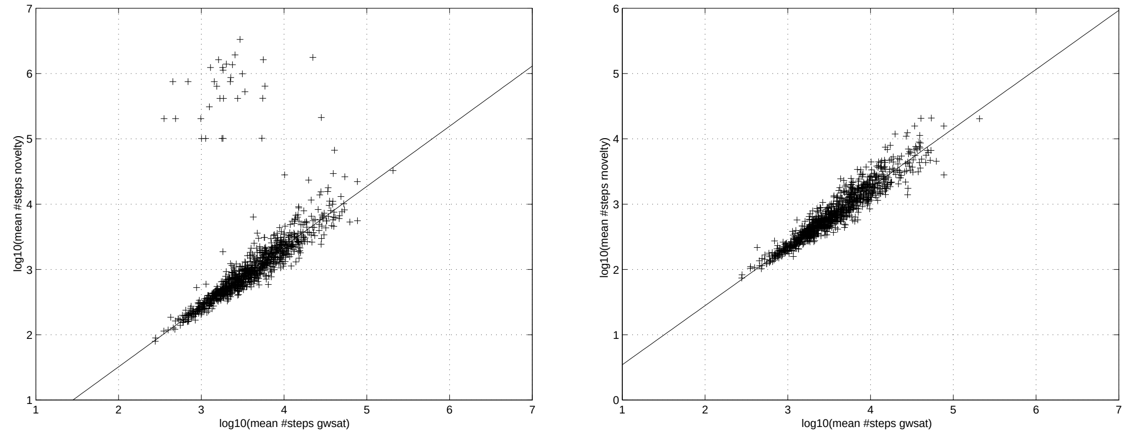


Figure 4.15: *Left:* Correlation between average local search cost for GWSAT (horizontal) and Novelty (vertical) for test-set `uf100-430`, using approx. optimal noise. *Right:* Same for GWSAT and R-Novelty.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	med	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat	6,532.04	10,639.43	1.63	3,398.10	1,844.02	7,040.34	1,099.45	13,880.52	3.82	12.62
gsat+tabu	4,783.97	9,741.31	2.04	1,621.54	757.77	4,260.16	457.23	11,399.68	5.62	24.93
wsat	3,672.59	5,698.34	1.55	1,995.42	1,143.13	4,142.59	701.81	7,296.75	3.62	10.40
wsat+tabu	2,485.44	6,263.64	2.52	1,041.02	565.51	2,215.02	351.74	4,751.42	3.92	13.51
novelty	28,257.51	191,668.71	6.78	851.87	479.14	1,845.30	302.88	4,390.39	3.85	14.50
r-novelty	1,245.90	1,893.65	1.52	640.140	375.91	1,402.15	232.30	2,625.92	3.73	11.30

Table 4.10: Test-set `uf100-430`, basic descriptive statistics of hardness distributions for various algorithms with approx. optimal noise, based on 100 tries / instance; noise settings as for medium instance (*cf.* Table 4.8).

cutoff value of 10^7 steps per try. In our analysis, we determined this hardness measure for each instance from the problem set and analysed the correlation between the hardness for different algorithms. To reduce the overall amount of computation, we chose GWSAT as a reference algorithm, *i.e.*, for each other algorithm we analysed the correlation between its performance and GWSAT's on a per instance basis. Figure 4.13 (left) shows the hardness distributions of `uf100-430` for various algorithms, while Figures 4.13 (right) to 4.15 show the correlation data as log-log plots.

When comparing the hardness distributions for the different algorithms, the most prominent feature is that the curves are roughly parallel in a semi-log plot, which indicates that the differences in search cost for these algorithms can be characterised by a uniform factor across the whole hardness distribution. However, examining the top 10% of the cumulative distribution curves, there are three notable exceptions from that rule: those for GSAT+tabu, WalkSAT+tabu, and — far more dramatically — Novelty have a significantly heavier tail than the curves for the other algorithms. Table 4.10 also reflects this behaviour; comparing

algorithms	r	a	b	o
gwsat *vs* gsat+tabu	0.9171	1.1456	-0.8007	0
gwsat *vs* wsat	0.9725	0.9229	0.0471	0
gwsat *vs* wsat+tabu	0.9464	0.9962	-0.4814	0
gwsat *vs* novelty	0.9431	0.9211	-0.3331	33
gwsat *vs* r-novelty	0.9492	0.9044	-0.3628	0

Table 4.11: Test-set `uf100-430`, pairwise hardness correlation for various algorithms with approx. optimal noise, based on 100 tries / instance; r is the correlation coefficient, a and b are the parameters of the *lms* regression analysis, and o the number of outliers which have been eliminated before the analysis (see text).

the normalised standard deviations and percentile ratios shows that GSAT+tabu, WalkSAT+tabu, and Novelty cause considerably more inter-instance hardness variation. For Novelty, 33 of the 1,000 instances had a solution rate between 75% and 99%, whereas for identical number of tries and cutoff per try all other algorithms solved all instances in each try. This explains the huge standard deviation observed for the Novelty hardness distribution. Furthermore, this observation strongly suggests that Novelty is essentially incomplete for the given test-set, and therefore for uniform Random-3-SAT in general.

However, the primary observation that the hardness distributions are mostly similarly shaped (neglecting the heavy tails for the moment) suggests that the hardness of instances for different algorithms is tightly correlated. This hypothesis is confirmed by the results from a direct correlation analysis (Figures 4.13–4.15). The scatter plots show that in a log-log scale there is a strong linear correlation between the hardness for GWSAT and the other algorithms. This correlation is strongest for WalkSAT (Figure 4.14, left), somewhat noisier for WalkSAT+tabu (Figure 4.14, right) and R-Novelty (Figure 4.15, right), and again a bit weaker for GSAT+tabu (Figure 4.13, right). The correlation for GWSAT and Novelty (Figure 4.15, left) shows a signifcant number of outliers, most (but not all of which) mark instances with a solution rate of less than 100%. Interestingly, these outliers occur across the whole hardness range of GWSAT, so even instances which are extremely easy for GWSAT (and the other algorithms) can be very difficult for Novelty. Nevertheless, except for these outliers, hardness for Novelty and GWSAT is tightly correlated. This indicates that the hardness for any of these algorithms (neglecting the outliers for Novelty) is an intrinsic property of the instances. Note, however, that the hardness correlation gets noisier for harder problems. This holds especially for the more effective variants of WalkSAT, like WalkSAT+tabu or R-Novelty.

Table 4.11 shows the results of the correlation and least-mean-squares (*lms*) linear regression analysis of the logarithm of the hardness (mean search cost per solution); the regression lines shown in the log-log plots (Figures 4.13–4.15) correspond to power functions of the type $y = x^a \cdot exp(b)$. The data confirms our earlier observation that the correlation between

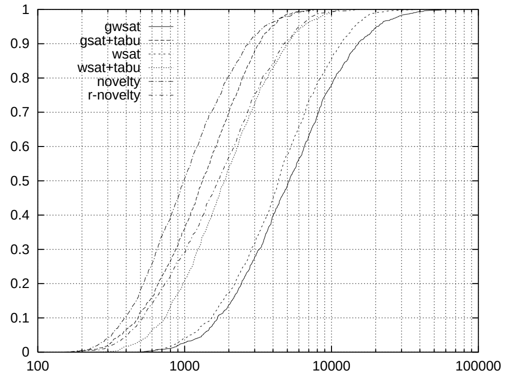


Figure 4.16: Problem `flat100-239/easy`, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	med	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat(0.6)	7,268.85	6,898.08	0.95	5,146	2,788	9,031	1,673	15,168	3.24	9.07
gsat+tabu(20)	1,636.08	1,156.67	0.71	1,320	756	2,242	481	3,215	2.97	6.68
wsat(0.5)	5,602.40	4,358.06	0.78	4,341	2,524	7,398	1,521	11,577	2.93	7.61
wsat+tabu(5)	2,453.86	1,924.93	0.78	1,863	1,118	3,155	734	4,978	2.82	6.78
novelty(0.6)	1,333.68	1,097.39	0.82	980	586	1,733	388	2,704	2.96	6.97
r-novelty(0.6)	2,253.83	1,912.84	0.85	1,687	860	2,958	516	4,721	3.44	9.15

Table 4.12: Problem instance `flat100-239/easy`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

the hardness for different algorithms is very strong; this holds also for Novelty after the outliers (which were discussed above) have been removed. From the regression data we see that the a parameter is almost equal to 1 for WalkSAT+tabu which indicates a constant factor between GWSAT and WalkSAT+tabu hardness across the whole test-set. In other words, regardless of the intrinsic hardness of an instance, the mean local search cost for WalkSAT+tabu is on average by a factor of 1.64 lower than for GWSAT. For GSAT+tabu, in contrast, a is significantly greater than 1, indicating that this algorithm's performance decreases relative to GWSAT's with increasing instance hardness. At the same time, the low value of the b parameter indicates that for easy instances, GSAT+tabu shows significantly better performance than GWSAT. The situation for WalkSAT and R-Novelty is significantly different: here, the a parameters are significantly lower than 1; consequently, for harder problems, their relative performance compared to GWSAT's increases with the hardness of the problem instances.

4.5.2 Graph Colouring

Applying the same analysis as used for Random-3-SAT to the `flat100-329` Graph Colouring

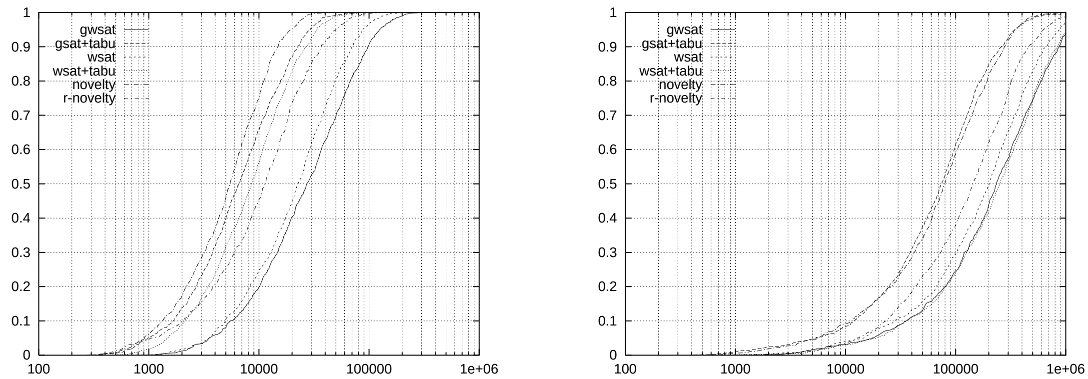


Figure 4.17: Problem `flat100-239/med` *(left)* and `flat100-239/hard` *(right)*, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries / instance.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	med	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat(0.6)	41,784.17	41,629.91	1.00	27,288	12,043	58,159	5,426	97,410	4.83	17.95
gsat+tabu(10)	9,811.36	9,765.54	1.00	6,539	3,203	13,041	1,656	22,443	4.07	13.55
wsat(0.5)	31,260.92	28,586.30	0.91	22,595	10,053	42,567	4,995	70,267	4.23	14.07
wsat+tabu(3)	11,881.07	11,094.59	0.93	8,548	4,160	16,086	2,091	26,590	3.87	12.72
novelty(0.6)	7,070.50	5,928.43	0.84	5,455	2,643	9,951	1,297	14,826	3.77	11.43
r-novelty(0.6)	16,183.90	16,333.39	1.01	11,291	4,769	20,790	1997	36,645	4.36	18.35

Table 4.13: Problem instance `flat100-239/med`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	med	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat(0.6)	306,886.96	249,012.94	0.81	240,302	103,701	451,475	43,870	700,129	4.35	15.96
gsat+tabu(10)	107,872.02	111,690.08	1.04	73,594	31,683	142,838	11,779	251,477	4.51	21.35
wsat(0.5)	254,373.79	222,835.34	0.88	192,788	79,300	375,285	26,948	578,138	4.73	21.45
wsat+tabu(3)	297,778.48	245,045.43	0.82	229,496	97,486	453,820	32,462	674,915	4.66	20.79
novelty(0.6)	116,773.89	117,259.20	1.00	79,307	33,396	159,525	11,273	276,180	4.78	24.50
r-novelty(0.6)	195,965.22	183,408.23	0.94	140,671	54,880	284,093	22,413	433,875	5.18	19.36

Table 4.14: Problem instance `flat100-239/hard`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

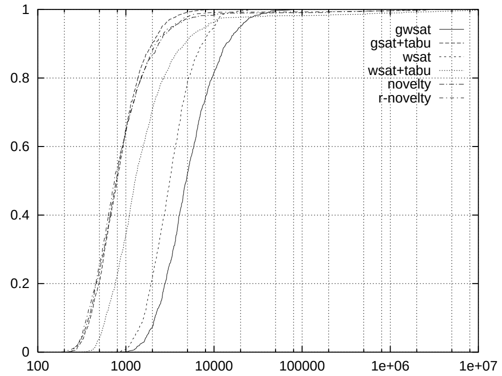


Figure 4.18: Hardness distributions across test-set `flat50-239` for various algorithms; x-axis: median number of local search steps / solution.

test-set, we observe some surprising differences (*cf.* Figures 4.16 and 4.17, and Tables 4.12–4.14). These concern mainly the performance of GSAT+tabu and R-Novelty. The latter algorithm, which dominated the scene for the Random-3-SAT test-set, shows a relatively weak performance on the Graph Colouring domain tested here, where it ranks third (on the easy and hard instance) or fourth (on the medium problem). Novelty is significantly better than R-Novelty on all three test instances; the performance difference is uniformly characterised by a factor of ca. 2 in the number of steps required to find a solution with a given probability.

On the other hand, GSAT+tabu is much stronger when applied to the Graph Colouring test-set than what we measured for Random-3-SAT. Here, for the easy and medium instance, GSAT+tabu is only second to Novelty, which is the best algorithm for these instances; for the hard instance, GSAT+tabu is slightly better than Novelty and thus leads the field. Analogously to what we observed for Random-3-SAT, WalkSAT+tabu's performance is relatively poor on the hard instance; on the `flat100-239` test-set it is actually inferior to WalkSAT which is somewhat surprising given the huge advantage GSAT+tabu realises over GWSAT (for the hard instance, we observe a factor of ca. 3 between the performance of these two algorithms).

Analysing the correlation of average search cost between different SLS algorithms across the whole test-set, like in the case of Random-3-SAT again gives a more detailed picture. For practical reasons (computing time limitations), we used the `flat50-115` test-set for this analysis; however, our experimental experience indicates that these results directly translate to `flat100-239`. The hardness distributions shown in Figure 4.18 (left) confirm and refine the observations from studying the three sample instances. Novelty, R-Novelty, and GSAT+tabu clearly dominate the other algorithms on the major part of the test-set; the hardness distributions of these algorithms appear to be almost identical. However, for Novelty and R-Novelty, as well as for WalkSAT+tabu, the distributions have heavy tails

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat	7,023.25	7,060.91	1.01	4,846.90	3,066.88	8,122.62	2,118.13	13,640.42	2.65	6.44
gsat+tabu	1,040.80	855.10	0.82	787.85	539.67	1,220.34	393.27	1,963.95	2.26	4.99
wsat	3,913.76	2,779.63	0.71	3,132.93	2,097.73	4,648.53	1,574.41	7,287.42	2.22	4.63
wsat+tabu	61,393.42	720,406.37	11.73	1,265.48	835.27	2,252.70	584.23	4,675.93	2.70	8.00
novelty	20,065.04	330,262.72	16.46	776.34	512.90	1,282.32	377.06	2,335.16	2.50	6.19
r-novelty	7,109.64	97,543.64	13.72	739.61	490.34	1,282.53	356.66	2,142.78	2.62	6.01

Table 4.15: Test-set `flat50-115`, basic descriptive statistics of hardness hardness distributions for various algorithms with approx. optimal noise, based on 100 tries / instance; noise settings as for medium instance (*cf.* Table 4.13).

algorithms	r	a	b	o
gwsat *vs* gsat+tabu	0.9021	0.7673	0.0715	0
gwsat *vs* wsat	0.9527	0.7738	0.6353	0
gwsat *vs* wsat+tabu	0.8417	0.8885	-0.1149	27
gwsat *vs* novelty	0.8398	0.8212	-0.1217	9
gwsat *vs* r-novelty	0.8247	0.7848	-0.0205	6

Table 4.16: Test-set `flat50-115`, pairwise hardness correlation for various algorithms with approx. optimal noise, based on 100 tries / instance; r is the correlation coefficient, a and b are the parameters of the *lms* regression analysis, and o the number of outliers which have been eliminated before the analysis (see text).

which indicate that the algorithms get stuck in local minima for a relatively small number of instances. Consequently, their worst-case performance on the given test-set is drastically worse than the other algorithms'. Note that for Novelty, a similar situation was observed in the case of Random-3-SAT — where, however, neither WalkSAT+tabu, nor R-Novelty showed any signs of essential incompleteness.

Studying the hardness correlations between GWSAT and the other algorithms, we again observe a tight correlation between GWSAT and WalkSAT; the correlation between GWSAT and GSAT+tabu is somewhat weaker, while for R-Novelty, Novelty, and WalkSAT+tabu a significant number of outliers occur, which correspond to instances for which the algorithm show stagnation behaviour (*cf.* Table 4.16). Again, as observed for Novelty when applied to the `uf100-430` test-set, the occurrence of the outliers is not correlated with the hardness of the instances for GWSAT or WalkSAT.[9] But even when removing these outliers, the correlation between GWSAT's and Novelty's resp. R-Novelty's and WalkSAT+tabu's per-

[9]Further analysis shows that the instances which are problematic for Novelty tend to be also outliers for WalkSAT+tabu. Also, for the "regular" instances the performance of these algorithms is tightly correlated. These results indicate that the features making a Graph Colouring instance hard for Novelty and WalkSAT+tabu are either the same or highly correlated.

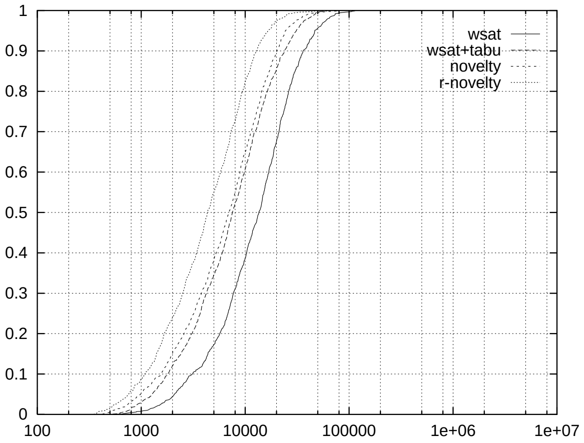


Figure 4.19: Problem instance `bw_large.a`, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

formance is significantly more noisy than for the other algorithms. Nevertheless, neglecting the outliers for Novelty, R-Novelty, and WalkSAT+tabu, the existence of a significant correlation between all these algorithms' local search cost across the test set indicates that, as for Random-3-SAT, the hardness as measured for these local search algorithms is an intrinsic property of the problem instances.

The regression fits show additionaly that both, Novelty's and WalkSAT+tabu's performance scale not as good as WalkSAT's and GSAT+tabu's with increasing hardness (see the a parameters in Table 4.16); as for Random-3-SAT, Novelty scales better than WalkSAT+tabu. For R-Novelty, the scaling is significantly better, but somewhat obscured by the relatively weak correlation. Surprisingly, GSAT+tabu, for which we observed the worst scaling behaviour on Random-3-SAT, shows not only superior performance, but also the best hardness scaling here.

4.5.3 Blocks World Planning

After evaluating the different algorithms on random distributions of hard problem instances, we now turn to single SAT-encoded instances from the Blocksworld Planning domain. Since we are dealing with a small number of single instances, the type of correlation analysis applied to the Random-3-SAT and Graph Colouring test-sets cannot be used in this context. Thus, we are restricted to comparing the optimal RLDs for individual instances, like it was done for the easy, medium, and hard instances from the distributions studied before. Also, after preliminary experiments indicated that the GSAT variants are substantially outperformed by the more efficient WalkSAT variants on these problems, we focussed mainly on algorithms from the WalkSAT family.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
gwsat(0.6)	26,994.18	24,784.20	0.92	18,653	9,296	38,041	5,053	58,679	4.09	11.61
gsat+tabu(10)	10,635.56	10,515.06	0.99	6,977	3,162	14,917	1,263	25,291	4.72	20.02
wsat(0.5)	17,490.27	15,320.75	0.88	13,505	6,822	23,446	3,057	35,981	3.44	11.77
wsat+tabu(3)	10,603.31	9,665.12	0.91	7,563	3,686	14,169	1,780	23,870	3.84	13.41
novelty(0.4)	9,315.07	8,587.17	0.92	6,932	3,202	12,877	1,534	20,202	4.02	13.17
r-novelty(0.6)	6,053.87	5,583.55	0.92	4,317	2,097	8,413	1,085	12,875	4.01	11.87

Table 4.17: Problem instance `bw_large.a`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	$\frac{Q_{75}}{Q_{25}}$
wsat(0.35)	582,384.26	570,626.82	0.98	414,744	166,326	782,842	4.71
wsat+tabu(2)	152,104.03	137,108.76	0.90	107,586	54,611	215,564	3.95
novelty(0.30)	191,715.39	195,045.91	1.02	131,494	53,629	254,779	4.75
r-novelty(0.55)	234,304.18	215,100.55	0.92	166,922	73,673	341,827	4.64

Table 4.18: Problem instance `bw_large.b`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 250 tries.

For the smaller instances (`medium`, `huge`, and `bw_large.a`) we observe a uniform performance pattern: R-Novelty is uniformly better than Novelty, which show slightly improved performance over WalkSAT+tabu, which in turn is significantly better than WalkSAT (see Figure 4.19, Table 4.17). For the larger instances, however, the situation is considerably different: Applied to `bw_large.b`, R-Novelty ranks only third behind WalkSAT+tabu and Novelty, which show a very similar performance (see Table 4.18).

For `bw_large.c`, WalkSAT+tabu is significantly better than Novelty (factor ≈ 2), while Novelty's performance is between WalkSAT's and WalkSAT+tabu's (see Figure 4.19, Table 4.19). The big surprise, however, is that R-Novelty performs extremely poorly: for noise parameter settings up to 0.5, we observed stagnation behaviour with extremely low asymptotic solution probabilities ($< 10\%$). However, if a solution is found, this requires only a relatively small number of steps on average – thus by using restart and a small cutoff value, competitive performance can be achieved. Nevertheless, even using an optimal cutoff value,

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	$\frac{Q_{75}}{Q_{25}}$
wsat(0.2)	14,061,484.00	15,594,687.33	1.11	9,757,075	3,500,013	16,967,151	4.85
wsat+tabu(2)	2,521,760.99	2,118,486.22	0.84	2,004,278	907,250	3,528,813	3.89
novelty(0.2)	6,385,022.16	7,125,793.91	1.12	4,401,794	1,738,668	8,163,114	4.70

Table 4.19: Problem instance `bw_large.c`, basic descriptive statistics of RLDs for various algorithms, approx. optimal noise, based on 250 tries.

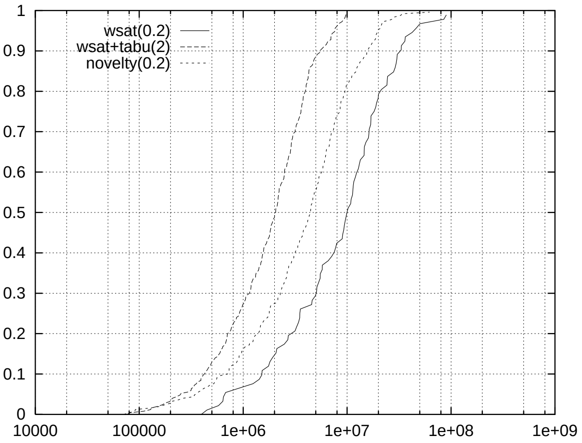


Figure 4.20: Problem instance `bw_large.c`, RLDs for various algorithms, approx. optimal noise, based on 250 tries.

algorithm	$\widehat{p_s}^*$	$\widehat{E_s}$	mean$_c$	stddev$_c$	median$_c$
gwsat(0.4)	1.0	3,029.03	3,029.03	3,027.59	2,096
gsat+tabu(10)	1.0	627.73	627.73	612.79	452
wsat(0.5)	1.0	1,243.47	1,243.47	1,262.55	842
wsat+tabu(10)	0.98	21,940.6	490.16	468.02	343
novelty(0.7)	0.10	$8.9 \cdot 10^6$	62.65	44.60	46
r-novelty(0.8)	1.0	7,122.56	7,122.56	8,099.91	4,595

Table 4.20: Problem instance `ais6`, basic descriptive statistics of conditional RLDs for various algorithms, approx. optimal noise; $\widehat{p_s}^*$ indicates the asymptotic maximal success probability, $\widehat{E_s}$ denotes the expected number of steps for finding a solution (using random restart).

R-Novelty's performance is still inferior to WalkSAT+tabu's (the best estimated factor between the corresponding average local search steps / solution values is 1.75). For higher noise settings and a cutoff after 10^8 steps, we obtained a maximal success probability of less then 15% and the best estimate for the median number of steps / solution is ca. $4 \cdot 10^8$); in this situation, R-Novelty is completely dominated by WalkSAT, WalkSAT+tabu, and Novelty.

4.5.4 All-Interval-Series

Finally, we compare the performance of our set of SLS algorithms for instances from the All-Interval-Series domain. Figures 4.21 and 4.22 show the RLDs for our set of SLS algorithms when applied to instances from the All-Interval-Series domain. All RLDs are based

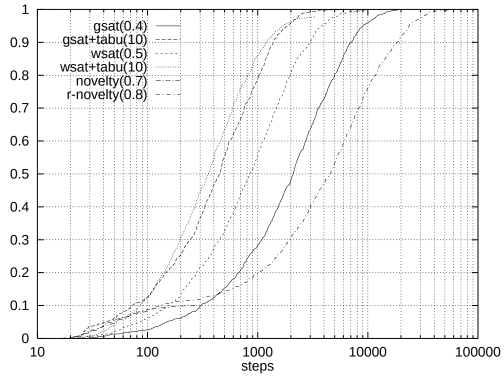


Figure 4.21: Problem instance `ais6`, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

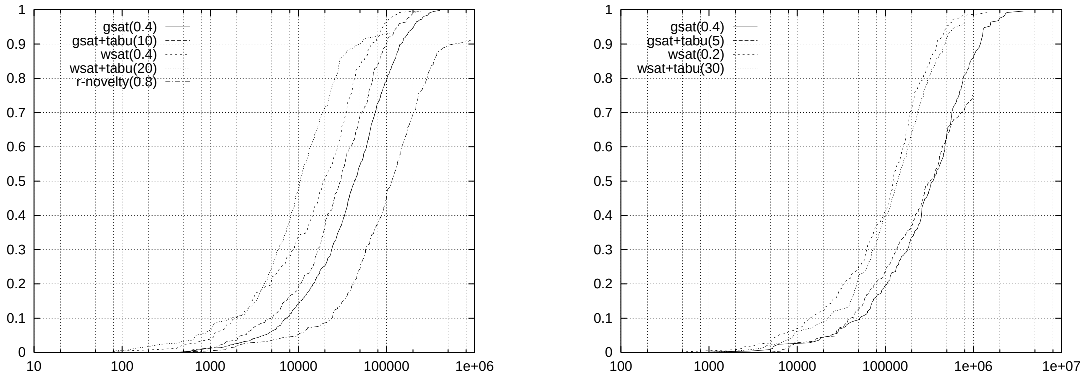


Figure 4.22: Problem instances `ais8` *(left)* and `ais10` *(right)*, RLDs for various algorithms, approx. optimal noise, based on 250 tries / instance.

algorithm	$\widehat{p_s}^*$	$\widehat{E_s}$	mean$_c$	stddev$_c$	median$_c$
gwsat(0.4)	1.0	64,167	64,167	64,661	43,128
gsat+tabu(10)	1.0	43,421	43,421	42,835	29,545
wsat(0.4)	1.0	28,528	28,528	30,232	19,291
wsat+tabu(20)	0.94	82,463	14,086	14,815	9,472
novelty(0.5)	0.005	$2.0 \cdot 10^8$	186.62	153.98	140
r-novelty(0.8)	0.92	231,482	139,779	135,747	99,880

Table 4.21: Problem instance `ais8`, basic descriptive statistics of conditional RLDs for various algorithms, approx. optimal noise; $\widehat{p_s}^*$ indicates the asymptotic maximal success probability, $\widehat{E_s}$ denotes the expected number of steps for finding a solution (using random restart).

algorithm	$\widehat{p_s}^*$	$\widehat{E_s}$	$mean_c$	$stddev_c$	$median_c$
gwsat(0.4)	1.0	494,718	494,718	509,034	336,556
gsat+tabu(5)	0.76	$3.5 \cdot 10^6$	271,000	243,433	203,635
wsat(0.2)	1.0	173,422	173,422	198,887	122,283
wsat+tabu(20)	0.96	548,005	174,561	153,331	130,502

Table 4.22: Problem instance ais10, basic descriptive statistics of conditional RLDs for various algorithms, approx. optimal noise; $\widehat{p_s}^*$ indicates the maximal success probability, $\widehat{E_s}$ denotes the expected number of steps for finding a solution (using random restart).

on 250 or more tries for approx. optimal noise, where the cutoff settings were sufficiently high (ms=10^6–10^7) to guarantee maximal success probabilities. Nevertheless, as shown in Tables 4.20–4.22, stagnation behaviour occurs rather frequently. Only GWSAT and WalkSAT show approximately complete behaviour on all instances; WalkSAT always dominates GWSAT, and the performance difference between the two is similar ($\approx$ 2.5) for all the instances considered here. Interestingly, the approx. optimal noise setting for GWSAT is identical for all instances, while for WalkSAT, as for the Blocks World Planning instances, it decreases with growing problem size.

GSAT+tabu shows approximately complete behaviour for the smaller instances, but seems to become essentially incomplete for ais10 which leads to the poor performance documented in the tables. But as can be seen from the RLDs, by using random restart with an appropriately chosen cutoff, GSAT+tabu will outperform GWSAT. However, the corresponding difference in performance shrinks for the large instances, indicating that GSAT+tabu scales worse than GWSAT. Similar observations hold for WalkSAT+tabu when compared to WalkSAT, but different from GSAT+tabu, WalkSAT+tabu shows stagnation behaviour even when applied to ais6. This indicates that, when applied to All-Interval-Series instances, the tabu search variants scale worse than the random walk versions of the corresponding algorithms; note that we made an analogous observation for Random-3-SAT. Interestingly, in this domain, the optimal tabu-list length (noise setting) for WalkSAT+tabu is significantly higher than for any other domain studied here, while this does not hold for GSAT+tabu.

Novelty and R-Novelty suffer severely from premature stagnation, indicating essentially incomplete behaviour of these algorithms. With growing problem size, their performance deteriorates rapidly, such that, when applied to ais10, Novelty and R-Novelty achieved maximal solution probabilities of less than 0.1% (for arbitrary noise settings, ms=1,000, and 10,000 runs). For this problem instance, even when using random restart with optimal cutoff values, these algorithms are dominated by all other SLS algorithms tested here.

For ais12, due to computing time limitations, we could not measure full RLDs. However, WalkSAT(0.1) and WalkSAT+tabu(40) found solutions for short cutoff values (ms=10,000) from which we can estimate the expected number of steps / solution as $\approx 5.0 \cdot 10^6$ and $2.6 \cdot 10^6$, resp. Generally, the instances from the All-Interval-Series domain appear to be extremely hard for the SLS algorithms studied here, considering the small size of the SAT-encodings.

Especially when compared to the Blocks World Planning instances, where WalkSAT, WalkSAT+tabu, and Novelty could solve instances with several thousand propositional variables within reasonable time, All-Interval-Series instances like `ais12` (265 variables, 5,666 clauses) seem to pose severe inherent difficulties for SLS algorithms.

In summary, our comparative analysis of different SLS algorithms' performance across various problem domains shows that there is no single best algorithm for all domains. However, there is a tendency indicating that Novelty and R-Novelty are best-performing for random, unstructured problems, while WalkSAT+tabu and GSAT+tabu might be superior for relatively hard, large, structured problem instances, like the large Blocks World Planning instances, the hard Graph Colouring instances, and the All-Interval-Series instances. However, all of these algorithms suffer from stagnation behaviour which severely compromises their performance.

4.6 Related Work

There is a growing body of work on local search methods for solving propositional SAT problems. Early work on SLS algorithms for SAT includes the seminal paper on GSAT by Selman, Levesque and Mitchell [SLM92] and Jun Gu's paper on efficient local search methods for SAT [Gu92]; however, stochastic hill-climbing was used before for solving MAXSAT problems [HJ90] and CSPs [MJPL90, MJPL92]. Since then, many other local search algorithms for SAT have been proposed and tested, including methods based on Simulated Annealing [Spe93, SKC93, BAH+94], Genetic Algorithms [Fra94, GV98], Evolution Strategies [Rec73, Sch81], and the Breakout Method [Mor93]. The GSAT algorithm itself has given rise to a large number of variants, some of which, such as GSAT with random walk [SKC93, SKC94], GSAT with clause weighting (GWSAT) [SK93, Fra97a], GSAT with tabu lists [MSG97], and HSAT [GW93b] could be shown to outperform the basic algorithm on a number of problem classes. The WalkSAT algorithm was originally described as a variant of GWSAT, but lately gave rise to a new family of very powerful SLS algorithms for SAT, including WalkSAT+tabu, Novelty, and R-Novelty [MSK97].

The idea to systematically study SLS algorithms for SAT by using algorithmic frameworks which can be instantiated to realise different local search strategies is not new. Gent and Walsh introduced the GenSAT frame and used it for empirically studying the relevance of certain features of GSAT and their influence on to SLS performance [GW93b]; this framework was later on further generalised to cover more recent and successful GSAT variants, such as GWSAT [BAH+94]. By further abstracting this scheme, even SLS algorithms based on Simulated Annealing could be realised within the same formal framework [HHSW94]. Another algorithmic framework is given by the WalkSAT architecture, as described in [MSK97]. Using algorithmic frameworks can be advantageous in the context of implemention. Some frameworks for local search are focussed mainly on this aspect, such as the object oriented scheme proposed in [Fra97b]. However, most such developments, including the GLSM model introduced in this work, are mainly inspired by the idea that using a generic framework helps to clearly understand similarities and differences between algorithms; furthermore, they facilitate comparative studies and the systematic exploration and analysis of whole families of algorithms.

When conducting empirical comparative studies of non-deterministic algorithms, it is extremely important to use identical test-sets of problem instances [HS98a]. As argued in Chapter 2, this is especially relevant when using random problem distributions with large inter-instance variability in hardness, like Random-3-SAT or Random Graph Colouring. While for many problems in AI and Operations Research, standardised benchmark-suites are available and widely used (*e.g.*, TSPLIB [Rei91] or ORLIB [Bea90]), this is not the case for SAT. There are some collections of SAT instances (such as the collections used in the 2nd DIMACS Challenge and the International Competition on Satisfiability Testing), but these are either rendered mostly obsolete by recent advances in algorithm development or they are too small resp. specialised to be widely used as a benchmark. We therefore designed a new set of benchmark problems for the empirical analysis of SAT algorithms in the context of this work.

Some of the SAT sub-classes which comprise our benchmark suite have been used extensively for the study of SAT algorithms. Random-3-SAT is probably one of the most frequently used problem distributions; many of the earlier studies of SLS algorithms as well as newer work use it as a test-bed [SLM92, PW96, MSK97]. Random-3-SAT has also received considerable attention in the context of phase-transition phenomena [MSL92, CA93]. Graph Colouring problems have also been used before for evaluating SLS algorithms [Hog96, SLM92]; while previous work mostly uses single hard instances (such as the ones which can be found in the DIMACS Challenge database), we deliberately chose test-sets sampled from random distributions of Graph Colouring instances obtained by using Joe Culberson's flat graph generator [Cul92]. The SAT-encoding used in this context is the same as in [SLM92]; similar test-sets have been used before in [Ste96].

The Blocks World Planning domain has a long history in AI. In the context of SAT it has been used as a benchmark since GSAT's early days [KS92], but only recently, after advances in SLS algorithms for SAT (WalkSAT) and improved SAT-encodings had made it possible to demonstrate the practicability of the approach [KS96], they became more prominent in the context of evaluating SAT algorithms and studying SAT-encodings [KMS96, MSK97, EMW97]. The instances we used for our benchmark suite are taken from Kautz' and Selman's SATPLAN distribution (see [KMS96]). To our best knowledge, the All-Interval-Series problem has not been used before as a benchmark for SLS algorithms; however, it is well-known in music theory [Col40, Kre74] and, in its abstract form, it is loosely related to other combinatorial algebraic problems, such as the Latin Square Problem [LP84].

There is a number of comparative studies of SLS algorithms for SAT [BAH$^+$94, GW93b, PW96]. However, to our best knowledge as to this writing there is only one other study involving the latest algorithms of the WalkSAT family [MSK97], which is fairly limited in its depth and scope. Furthermore, none of the other studies we are aware of involves analyses of RLDs or performance correlations. While some of our findings confirm or refine earlier results (such as the superiority of WalkSAT when compared to GWSAT [SKC94]), others are quite surprising. This holds in particular for our observations regarding Novelty, R-Novelty, and the tabu-search variants of GSAT and WalkSAT. In particular the stagnation behaviour of Novelty and R-Novelty we observed for some of the algorithms has not been reported (nor expected) before.

4.7 Conclusions

In this chapter, we formalised some of the most prominent SLS algorithms for SAT as instances of the GLSM model developed in Chapter 3. Based on the GLSM representations, we discussed their specific similarities and differences. Furthermore, we introduced a benchmark suite comprising test-sets and single instances from a number of subclasses of SAT, including Random-3-SAT and SAT-encoded problems from different domains such as Graph Colouring, Blocks World Planning, and the All-Interval-Series problem. As argued in Chapter 2, such a benchmark suite is an important basis for comparing stochastic algorithms; testing these algorithms on identical problem instances is crucial when using random problem distributions with large inter-instance variability in hardness, like we observed for Random-3-SAT or Random Graph Colouring. Until now, however, no widely used set of benchmark problems has been available for SAT; consequently, we had to create our own benchmark suite, which is now publicly available at the SATLIB website (`www.informatik.tu-darmstadt.de/AI/SATLIB`).

We used this benchmark suite for an extensive comparative study of the different SLS algorithm's performance. Different from most previous studies, our methodology takes into account the extreme inter-instance variability in instance hardness across test-sets sampled from random instance distributions. Comparing the run-time distributions for different algorithms when applied to single problem instances using approximately optimal noise parameter settings, we could show that complete domination seems to be the rule. This means that when comparing two algorithms on the same instance, one of them shows consistently better performance, independent of the cutoff time. There are exceptions to that rule, in particular for the All-Interval-Series domain, where Novelty and R-Novelty are often better for shorter runs while otherwise being inferior to the other WalkSAT variants.

Generally, the performance differences between the algorithms are not consistent across test-sets or problem domains. While among the algorithms studied here, R-Novelty is clearly the best for Random-3-SAT (particularly on hard instances), it is consistently outperformed by Novelty and GSAT+tabu on the Graph Colouring domain and by WalkSAT+tabu for the larger Blocks World Planning instances, where it increasingly shows stagnation behaviour. Applied to the All-Interval-Series instances, WalkSAT, followed by GWSAT, dominates the scene, since for all other algorithms, including R-Novelty and Novelty, the performance rapidly deteriorates with increasing problem size (due to increasingly drastic stagnation behaviour). Furthermore, there are some performance relations which can be observed across all problem domains from our benchmark set: GWSAT is mostly inferior to all WalkSAT variants (except in the cases, where Novelty or R-Novelty show stagnation behaviour); and GSAT+tabu consistently outperforms WalkSAT, except in the cases where it suffers from premature stagnation.

In a more detailed study, we analysed the correlation between the different algorithm's performance across the Random-3-SAT and Graph Colouring test-sets. As a result, we found that there is a relatively strong linear correlation between the logarithm of the average local search cost for each pair of algorithms. In other words, instances which are hard for one algorithm tend also to be hard for all other algorithms. Consequently it is justified to talk about the intrinsic hardness of problem instances. The existence of the tight correlation in search cost we observed for various SLS algorithms is somewhat surprising, because the underlying concepts are sufficiently different to cause significant differences in performance, as we have seen. This suggests that the local search cost for these algorithms depends on the same structural features of the search space. Later on, in Chapter 6, we try to identify and analyse some of these features.

Another interesting result is the fact that Novelty, R-Novelty, and WalkSAT+tabu suffer from stagnation behaviour on a number of instances (outliers) from the Random-3-SAT and Graph Colouring test-sets; this phenomenon is not correlated to the intrinsic hardness of the corresponding instances. We also observe that, when neglecting these outliers, WalkSAT's, Novelty's, and R-Novelty's performance scales significantly better with instance hardness than GWSAT's and WalkSAT+tabu's. For GWSAT+tabu, which showed the worst scaling behaviour (w.r.t. instance hardness) on Random-3-SAT, we observed not only superior performance, but also the best scaling behaviour when applied to the Graph Colouring domain. When comparing the performance of SLS algorithms for the Blocks World Planning and the All-Interval-Series domain, we find that the former problem class appears to be relatively easy (relative to the size of the formulae), while the latter seems to be very hard. Since in both cases, as well as for the Graph Colouring domain, essentially the same encoding was used, we conjecture that this must be a feature of the original problem rather than an artifact introduced by the SAT encoding.

Summarising the results presented in this chapter, we found that although there are significant performance differences between the algorithms studied here, these are generally not consistent across test-sets and problem domains. Thus, there is no single best algorithm, dominating all others on all domains. There is however a notion of intrinsic hardness of problem instances, rendering the same instances difficult for all algorithms. At the same time, there are systematic differences between algorithms w.r.t. their performance scaling with increasing instance hardness across test-sets. Finally, we have found evidence that some of the most recently proposed algorithms like Novelty and R-Novelty, suffer from essential incompleteness, which severely compromises their performance, especially on hard structured problems.

Chapter 5

Characterising and Improving SLS Behaviour

Building on Chapter 4, here we extend and refine the analysis of SLS behaviour. We perform this analysis for the SLS algorithms for SAT introduced and discussed before; for empirical investigations we use the benchmark suite described in Section 4.4. However, the methodology used here is not restricted to SAT but can be rather easily applied to other problem domains and different SLS algorithms. In our refined analysis of SLS behaviour we first focus on qualitative norms of behaviour like approximate completeness. For the first time we give theoretical results concerning the approximate completeness of some of the most powerful currently known SLS algorithms for SAT. Next, we extend our empirical analysis of SLS behaviour by functionally characterising the algorithm's RTDs for optimal and sub-optimal noise parameter settings. Our analysis reveals a suprisingly regular behaviour of some of the best SLS algorithms for SAT when applied to hard instances from different benchmark domains. From these empirical results we derive a number of theoretically and practically interesting consequences, concerning parameterisation and parallelisation of these algorithms as well as an interpretation of their run-time behaviour. Finally, we show how, based on our characterisation results, some of the best currently known SLS algorithms for SAT can be further improved.

5.1 Characterising SLS Behaviour

In the last chapter we compared different SLS algorithms for SAT on our suite of benchmark problems; now, we analyse the behaviour of these algorithms in more detail. For this analysis we always use the algorithms without a restart mechanism, as the effects of the restart mechanism depend crucially on the run-time behaviour of the pure strategies and can be easily determined when its run-time distribution is known. In this section, we use the functional characterisation of run-time distributions as a central method, *i.e.*, we develop a mathematical model which describes the empirically observed behaviour correctly and accurately.

5.1.1 Asymptotic Run-time Behaviour

Local search algorithms as the ones we study here belong to the class of Las Vegas algorithms. As discussed in an abstract context in Chapter 2, these can be classified into three qualitative norms of behaviour with respect to their asymptotic behaviour for arbitrarily long run-times (see Section 2.1.1).

While all GSAT and WalkSAT variants are *incomplete*, when evaluating these algorithms we already observed that some of them appear to be *essentially incomplete*, while others seem to be *approximately complete*, *i.e.*, with increasing run-time the probability of finding a solution for a soluble instance approaches one. We will use the term *PAC behaviour*[1] to characterise a situation, where a Las Vegas algorithm, applied to a particular problem instance, shows approximately complete behaviour. Algorithms which show PAC behaviour for all soluble instances will be called *PAC algorithms*. Note, that empirical observations can neither prove PAC behaviour for a given algorithm applied to a particular instance (because based on a finite number of runs, it cannot be ruled out that for some very unlikely search trajectories, the algorithms gets stuck and will never find a solution), nor the PAC property of an algorithm. Essential incompleteness cannot be proven empirically, either: although it suffices to find one instance for which the RLD does not converge towards one, all experimental analyses have to be based on finite run-times – consequently, it can never be ruled out empirically that for even longer run-times convergence could be observed. Nevertheless, empirical results can indicate PAC behaviour and essential incompleteness; moreover, the significance of such evidence can be improved by increasing the number and run-time of experiments.

At the time of this writing we are not aware of any theoretical result concerning the approximate completeness or essential incompleteness of any of the SLS algorithms for SAT studied here. Our empirical evaluations (*cf.* Chapter 4) suggest the following hypotheses:

- plain GSAT is essentially incomplete;
- GWSAT is approximately complete for $\mathsf{wp} > 0$;
- GSAT+tabu is essentially incomplete for very small and very large tabu list lengths;
- WalkSAT is approximately complete for noise parameter settings $\mathsf{wp} > 0$;
- WalkSAT+tabu is essentially incomplete for very small and very large tabu list lengths;
- Novelty is essentially incomplete at least for some noise parameter settings;

[1]PAC is the abbreviation for Probabilistically Aproximately Complete; the term is inspired by *PAC learning*, a related concept from machine learning [KM90, Mit97].

- R-Novelty is essentially incomplete at least for some noise parameter settings.

While some of these hypotheses can be proven or refuted theoretically, for others it is not clear how to find a proof. To indicate how theoretical results concerning approximate completeness can be derived, we give proof sketches for some cases here. First we prove the approximate completeness for GSAT with random walk (GWSAT).

Theorem 5.1

GWSAT is approximately complete for all $\mathsf{wp} > 0$.

Proof For a given instance with n variables and k clauses, we show that there exists a $p' > 0$ such that from each non-solution assignment GWSAT can reach a solution with a probability $p \geq p'$: Let a be the current (non-solution) assignment, and s the solution with minimal hamming distance h from a.

Lemma 1: For arbitrary a and s there is always one valid GWSAT step which decreases h by one. In particular, there is always at least one such random walk step.

To see why this holds, assume that no such random walk step exists. Then none of the variables whose values are different in a and s can appear in an unsatisfied clause. But since a is not a solution, there has to be at least one clause c which is violated by a; now the variables appearing in c have the same value in a and s, therefore s also violates c and cannot be a solution. Since this contradicts the assumption, Lemma 1 has to be true.

Using Lemma 1 inductively, one can construct a sequence of h random walk steps from a to s. Next, we derive a lower bound for the probability with which GWSAT will execute this sequence.

Note first that for any assignment a', the number of unsatisfied clauses is always less or equal to k, and the number of literals occurring in unsatisfied clauses is less or equal to $k \cdot l$, where l is the length of the longest clause in the given instance. Therefore, if GWSAT decided to do a random walk step, the probability to select a variable which decreases h is at least $1/(k \cdot l)$. Thus, the overall probability of executing a random walk step which decreases the hamming distance to the nearest solution is at least $\mathsf{wp}/(k \cdot l)$. Since GWSAT's steps depend only on the current assignment (Markov property), a lower bound for the probability of executing the correct h step sequence to reach the solution can be estimated as $[\mathsf{wp}/(k \cdot l)]^h$.

Finally, note that h is always less or equal to n, and therefore the probability of reaching a solution from any given assignment a is at least $p' = [\mathsf{wp}/(k \cdot l)]^n$. Since this probability is independent from the number of steps which have been performed, the probability of finding a solution within t steps is at least

$$\sum_{i=1}^{t} (1-p')^{i-1} \cdot p' = p' \cdot \frac{(1-p')^t - 1}{(1-p') - 1}$$

For $t \to \infty$, this geometric series converges to $p'/p' = 1$, which proves GWSAT's approximate completeness. □

Note that this proof relies critically on the fact that the algorithm can decrease the hamming distance to the nearest solution in each step. Our proof shows that this is guaranteed if arbitrarily long sequences of random walk steps can be performed with a probability $p \geq p' > 0$ which is independent of the number of steps which have been performed in the past. Since our estimates for the probability of finding a solution did not depend on the actual variable selection strategy, the proof generalises easily to each algorithm which satisfies this condition.

The algorithms of the WalkSAT family, however, generally do not allow arbitrarily long sequences of random walk steps. In particular, for WalkSAT the variable selection strategy does not allow a random walk step if the selected clause contains a variable which can be flipped without breaking any currently satisfied clauses. Therefore, approximate completeness cannot be proven using the scheme given above. Actually, although our empirical results suggest that WalkSAT could be approximately complete, a proof seems to be difficult to find. For the other members of the WalkSAT family, we can prove their essential incompleteness using a very simple example instance. The proofs for WalkSAT+tabu, Novelty, and R-Novelty are very similar; therefore we give only the proof for Novelty here, and then discuss the corresponding results for the other algorithms.

Theorem 5.2

Novelty is essentially incomplete for arbitrary noise parameter settings.

Proof Let $F = \bigwedge_{i=1}^{6} c_i$ be the formula consisting of the clauses:

$$\begin{aligned}
c_1 &\equiv \neg x_1 \vee x_2 \\
c_2 &\equiv \neg x_2 \vee x_1 \\
c_3 &\equiv \neg x_1 \vee \neg x_2 \vee \neg y \\
c_4 &\equiv x_1 \vee x_2 \\
c_5 &\equiv \neg z_1 \vee y \\
c_6 &\equiv \neg z_2 \vee y
\end{aligned}$$

F is satisfiable and has exactly one model $(x_1 = x_2 = \top, y = z_1 = z_2 = \bot)$. Now assume that the algorithm's current assignment is $A_1 \equiv (x_1 = x_2 = y = z_1 = z_2 = \top)$. In this situation, all clauses except c_3 are satisfied. Applying Novelty, c_3 will be selected and the variables receive the following scores: $x_1 : 0, x_2 : 0, y : -1$. Since regardless of the noise parameter, Novelty always flips the best or second-best variable, either x_1 or x_2 will be flipped.

Because both cases are symmetric, we assume without loss of generality that x_1 is flipped. This leads to the assignment $A_2 \equiv (x_1 = \bot, x_2 = y = z_1 = z_2 = \top)$ which satisfies all clauses except c_2. The scores for x_1 and x_2 are both 0, and since x_1 is the most recently flipped variable, x_2 will be picked now. The current assignment at this point is $A_3 \equiv (x_1 = x_2 = \bot, y = z_1 = z_2 = \top)$ which satisfies all clauses except c_4. Again, x_1 and x_2 have the same score of 0, but now x_2 is the most recently flipped variable, so x_1 will be flipped. Now, the current assignment is $A_4 \equiv (x_1 = \top, x_2 = \bot, y = z_1 = z_2 = \top)$, which leaves only c_1 unsatisfied. As before, x_1 and x_2 both receive a score of 0, and x_2 will be flipped. But this leads back to the assignment $A_1 \equiv (x_1 = x_2 = y = z_1 = z_2 = \top)$; therefore, Novelty got stuck in a loop.

Therefore, we found a soluble problem instance F and an initial assignment A_1 for which Novelty will never reach a solution, regardless of how long it runs. Consequently, Novelty cannot be approximately complete but has to be essentially incomplete. □

Note that this result does not mean that Novelty will show essentially incomplete behaviour on each given problem instance. However, while for most of our benchmark problems, we observed approximately complete behaviour, for some instances Novelty got stuck in loops (*cf.* Chapter 4). Of course, in our example as well as in other cases, looping behaviour will usually only occur with a certain probability, since many trajectories of the algorithm avoid assignments leading into loops. Using the same example, one can easily prove:

Corollary 5.3 WalkSAT

+tabu and R-Novelty are essentially incomplete for arbitrary tabu-list lengths and noise parameter settings, resp.

Note that for R-Novelty, even the built-in deterministic loop breaking strategy (randomly picking a variable from the selected clause every 100th step) does not prevent the algorithm from getting stuck in loops, since these can be timed such that the loop breaker will never be activated when c_3 is violated — which would be the only way of flipping y and reaching a solution. In the case of WalkSAT+tabu, the same loop will be observed for any tabu list length $\mathsf{tl} > 0$. For $\mathsf{tl} \geq 2$, the reason for this is the fact that when all variables in a clause are tabu, WalkSAT+tabu will not flip any variable at all (*cf.* Section 4.3); for $\mathsf{tl} = 1$, as for Novelty, y can never be flipped when c_3 is selected.

So generally, although algorithms like WalkSAT+tabu, Novelty, and R-Novelty show mostly superior performance compared to GWSAT and WalkSAT, they suffer from essential incompleteness. However, as we will discuss in Section 5.2, based on the proofs outlined here we can overcome this weakness and thus substantially improve the most powerful SLS algorithms for SAT.

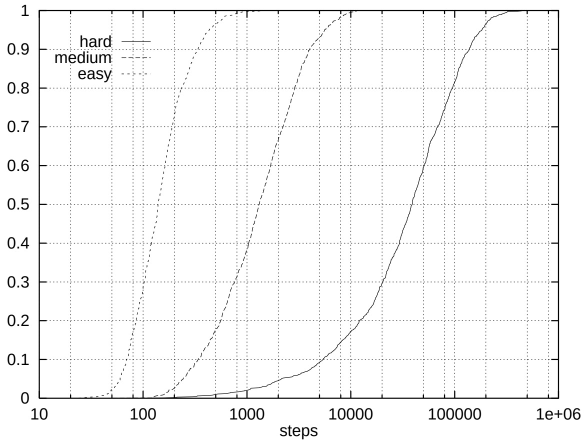


Figure 5.1: RLDs for WalkSAT (approx. optimal noise, `wp` = 0.55) applied to easy, medium, and hard problem instance from Random-3-SAT test-set `uf100-430`.

5.1.2 Functional Approximations for Optimal Noise

In the last section, we characterised the asymptotic behaviour of various modern SLS algorithms for SAT; now, we refine our analysis by functionally characterising their run-time behaviour. For this analysis, we apply the empirical methodology developed in Chapter 2, *i.e.*, for a given algorithm, we approximate the empirical RLDs for individual problem instances with continuous probability distributions; the goodness-of-fit of these approximations is tested using a standard χ^2 test. Our empirical results show that for optimal noise parameter settings, the run-time distributions of the SLS algorithms studied here when applied to hard problem instances can be well aproximated by exponential distributions. We use the term *EPAC behaviour*[2] to characterise a situation, where the RLD for a given Las Vegas algorithm, when applied to a particular problem instance, can be well approximated by an exponential distribution. Informally, we refer to algorithms which show EPAC behaviour for a wide range of hard problem instances as *EPAC algorithms*. In the following sections, we will present the empirical results regarding the EPAC property of several modern SLS algorithms for the different problem domains from our benchmark suite.

Random-3-SAT Figure 5.1 shows the RLDs for WalkSAT (using an approximately optimal noise setting of 0.55, 1,000 tries and cutoff settings high enough to guarantee 100% success rate) when applied to the easy, medium, and hard problem instance from the `uf100-430` test-set. For the hard instance, the RLD can be approximated[3] using the cumulative form of an exponential distribution $ed[m](x) = 1 - 2^{-x/m}$ where m is the median of the distri-

[2] EPAC is the abbreviation for Exponentially Probabilistically Aproximately Complete.
[3] All approximations were done using C. Gramme's "Gnufit" implementation of the Marquart-Levenberg algorithm.

instance	median #steps m	χ^2 for $ed[m]$	passed (α)
uf100-430/easy	132	501.68	no
uf100-430/medium	1,433	69.41	no
uf100-430/hard	61,082	27.51	yes (0.05)

Table 5.1: RLD approximations using exponential distributions $ed[m]$ for WalkSAT (approx. optimal noise) applied to the easy, medium, and hard instance from `uf100-430` test-set.

test-set	acceptance level α	number passed
uf50-218	0.01	11.4%
uf50-218	0.05	7.1%
uf100-430	0.01	16.6%
uf100-430	0.05	10.3%
uf200-860	0.01	30%
uf200-860	0.05	26%

Table 5.2: Fraction of instances passing the χ^2 test for different Random-3-SAT test-sets.

bution and x the number of steps required to find a solution.[4] For testing the goodness of this approximation we use a standard χ^2-test [Roh76]. Basically, for a given empirical RLD this is done by estimating the parameter m and comparing the deviations to the predicted distribution $ed[m]$. The result of this comparison is the χ^2 value, where low χ^2 values indicate a close correspondence between empirical and predicted distribution. Table 5.1 shows the estimated parameters m and the χ^2 values for the easy, medium, and hard instance mentioned before. It can be clearly seen that with increasing median search cost m, the χ^2 values decrease, indicating that the harder the problem, the closer WalkSAT's RLD on this problem approximates an exponential distribution. For the medium and easy problem, this approximation is still reasonably good for the tail of the distribution (*i.e.*, for long runs), while for shorter runs the actual distribution is steeper than an exponential distribution. We conjecture that the deviations are caused by the initial hill-climb phase of the local search procedure (*cf.* [GW93a]).

This observation leads to the following hypothesis: *Applied to hard Random-3-SAT instances from the phase transition region, WalkSAT with optimal noise setting shows EPAC behaviour, i.e., its behaviour can be characterised using exponential distributions.* To test this hypothesis, we apply the methodology outlined above to the entire 50, 100, and 200 variable test-sets. The resulting correlation between median search cost and χ^2 values can

[4] In the statistical literature, the exponential distribution $Exp(\lambda)$ is usually defined by $P(X \leq x) = 1 - e^{-\lambda x}$, which is equivalent to our representation $ed[m]$ using $m = \ln 2/\lambda$. A similar argument applies to the Weibull distribution mentioned later.

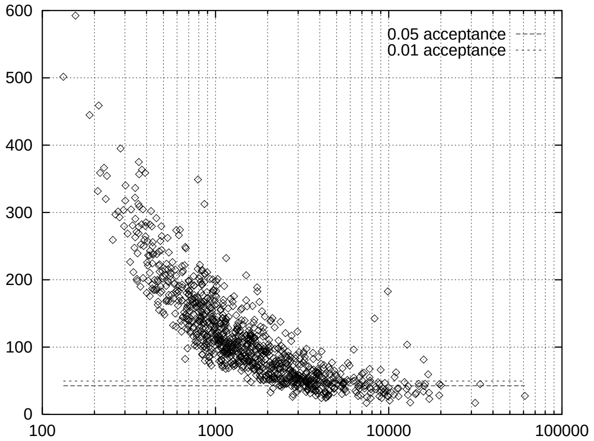


Figure 5.2: Correlation between hardness of problems (horizontal) and χ^2 values (vertical) from testing the RLDs of individual instances versus a best-fit exponential distribution for test-set `uf100-430`. The horizontal lines indicate the acceptance thresholds for the 0.01 and 0.05 acceptance level (one of these lines is hard to see, since it coincides with the $\chi^2 = 50$ line).

be seen from the scatter plots given in Figures 5.2 and 5.3. Obviously, there is a strong negative correlation, indicating that, indeed, for harder problem instances, WalkSAT's behaviour can be more and more accurately characterised by exponential distributions. The figures also indicate two standard acceptance levels[5] for the χ^2 test ($\alpha = 0.01$ and $\alpha = 0.05$). As can be seen from the plots, for high median search cost, almost all instances pass the χ^2 test. Table 5.2 shows the overall percentage of the instances which passed the test for the different acceptance levels. The data suggests that for increasing problem hardness, a relatively higher number of instances pass the test, *i.e.*, the deviations of the RLDs from ideal exponential distributions apparently become less prominent for larger problems.

Table 5.3 reports the results of an analogous analysis for different algorithms when applied to the test-set `uf100-430`. This time, the χ^2 test was restricted to the hardest 25% of the test-set. For WalkSAT+tabu and Novelty, stagnation behaviour was observed for some instances; these were removed from the test-set before approximating and testing the RLDs. As can be seen from Table 5.3, for ca. 50% of the tested instances the RLDs could be successfully approximated using exponential distributions. In the majority of the cases where the approximations failed the χ^2 test, like pointed out earlier for WalkSAT, the reason lies in the deviations caused by the initial search phase. In summary, this shows clearly that the result established for WalkSAT generalises to the other, more powerful WalkSAT variants: Applied to hard Random-3-SAT instances, these show approximately

[5]The acceptance level α specifies the probability with which a statistical test incorrectely rejects the given hypothesis. Therefore, for lower acceptance levels, the test has to be more cautious with rejecting the hypothesis than for higher acceptance levels.

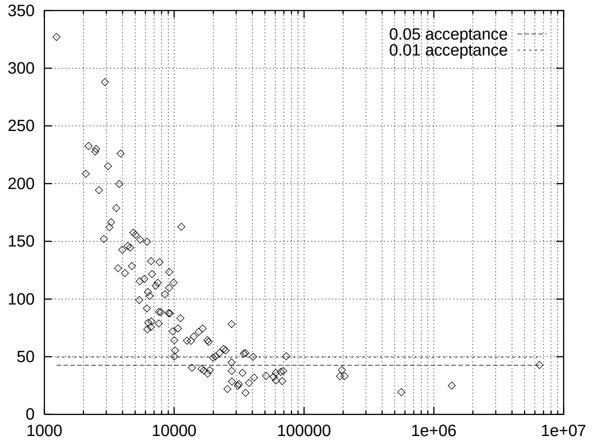


Figure 5.3: Correlation between hardness of problems (horizontal) and χ^2 values (vertical) from testing the RLDs of individual instances versus a best-fit exponential distribution for test-set `uf200-860`. The horizontal lines indicate the acceptance thresholds for the 0.01 and 0.05 acceptance level.

exponential RLDs when using optimal noise settings. However, since these algorithms are essentially incomplete, there are instances for which they show stagnation behaviour and to which, consequently, this characterisation does not apply.

Graph Colouring In further experiments, we applied the same methodology to the `flat100-239` test-set from the graph colouring domain. As for the Random-3-SAT test-sets, we used approximally optimal noise parameter settings. For each algorithm we performed 1,000 tries, using an extremely large maxSteps value (10^7) to ensure maximal success rates for all instances. The results of the χ^2 *vs* median search cost correlation analysis are summarised in Table 5.4. As for Random-3-SAT, we observe a strong negative correlation between the hardness of the problem instances and the χ^2 values, indicating that with increasing median local search cost, the RLDs are more and more accurately approximated by exponential distributions. This holds for WalkSAT as well as for its more powerful, but essentially incomplete variants; for the latter the instances for which essentially incomplete behaviour is observed are not considered in the correlation analysis. Comparing the results for the `flat100-239` test-set with those for `flat50-115` (not reported here) confirm the observation we made for Random-3-SAT: with increasing problem size, the fraction of instances from the test-set for which the approximation using exponential distributions pass the χ^2-test increases considerably. Note also that compared to the `uf100-430` test-set, the acceptance rates are generally higher for the `flat100-239` test-set. This is most probably caused by the fact that the `flat100-239` instances are larger and harder than the `uf100-430` instances.

algorithm	acceptance level α	fraction passed	number removed
wsat(0.55)	0.01	57.6%	0
	0.05	34%	0
wsat+tabu(3)	0.01	49.6%	1
	0.05	36%	1
novelty(0.7)	0.01	57.2%	35
	0.05	40.2%	35
r-novelty(0.7)	0.01	49.6%	0
	0.05	34.8%	0

Table 5.3: Fraction of instances from the hardest 25% of the `uf100-430` Random-3-SAT test-set passing the χ^2 test for different algorithms with approx. optimal noise settings; the last column indicates the number of instances for which essentially incomplete behaviour was observed, these were removed from the test-set.

algorithm	acceptance level α	fraction passed	number removed
wsat(0.5)	0.01	52%	0
	0.05	40%	0
wsat+tabu(3)	0.01	64%	1
	0.05	48%	1
novelty(0.6)	0.01	84%	1
	0.05	76%	1
r-novelty(0.6)	0.01	100%	1
	0.05	100%	1

Table 5.4: Fraction of instances from the hardest 25% of the `flat100-239` Graph Colouring test-set passing the χ^2 test for different algorithms with approx. optimal noise settings; the last column indicates the number of instances for which essentially incomplete behaviour was observed, these were removed from the test-set.

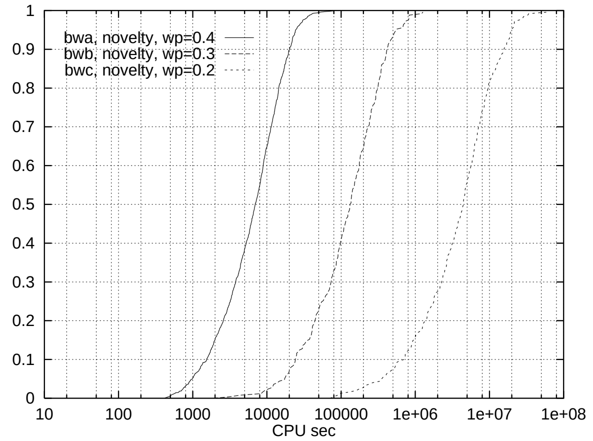


Figure 5.4: RLDs for Novelty with approx. optimal noise setting on SAT-encoded blocksworld planning problems.

instance	median #steps m	χ^2 for $ed[m]$	ν	passed (α)
medium	847	162.07	29	no
huge	$14,905$	73.93	29	no
bw_large.a	$6,839$	60.37	29	no
bw_large.b	$119,680$	11.40	16	yes (0.05)
bw_large.c	$4.27 \cdot 10^6$	8.71	16	yes (0.05)
bw_large.d	$8.51 \cdot 10^6$	10.14	10	yes (0.05)

Table 5.5: RLD approximations using exponential distributions $ed[m]$ for Novelty (approx. optimal noise) applied to Blocks World Planning instances; last column indicates whether the approximation passed the χ^2 test.

Blocks World Planning Fig. 5.4 shows the RLDs for Novelty when applied to SAT-encoded problem instances from the blocks-world planning domain. We used the optimal noise parameters from Section 4.5 and maxSteps settings high enough (10^7 and 10^8, resp.) to ensure 100% success rate. For the smaller instances (up to `bw_large.a`) we used 1,000 runs, for `bw_large.b` and `bw_large.c` 250, and for `bw_large.d` 100 runs to approximate the actual RLDs as described before. The estimates for the parameter m and the corresponding χ^2 values for the approximation by exponential distributions are shown in Table 5.5. The critical χ^2 values for a standard $\alpha = 0.05$ acceptance level are 42.6 for 1,000 tries ($\nu = 29$) and 26.3 for 250 tries ($\nu = 16$). This means, that only for the smaller instances the approximation does not pass the test. As for the easy Random-3-SAT instances, this can be explained by the initial hill-climb phase of local search.

Table 5.6 reports the results of approximating the RLDs of various SLS algorithms applied

algorithm	median #steps m	χ^2 for $ed[m]$	passed (α)
wsat(0.35)	119,680	11.40	yes (0.05)
wsat+tabu(2)	110,416	27.50	yes (0.01)
novelty(0.3)	133,453	14.47	yes (0.05)
rnovelty(0.55)	170,734	12.68	yes (0.05)

Table 5.6: RLD approximations using exponential distributions $ed[m]$ for various algorithms applied to Blocks World Planning instance `bw_large.b` (using approx. optimal noise); the last column indicates whether the approximation passed the χ^2 test ($\nu = 16$).

instance	algorithm	median #steps m	χ^2 for $ed[m]$	ν	passed (α)
ais6	gwsat(0.5)	1,898.1	33.34	29	yes (0.05)
ais6	wsat(0.5)	857.78	28.31	29	yes (0.05)
ais8	gwsat(0.4)	44,187.5	35.72	29	yes (0.05)
ais8	wsat(0.4)	19,439.9	23.01	16	yes (0.05)
ais10	gwsat(0.4)	343,096	25.51	16	yes (0.05)
ais10	wsat(0.2)	118,237	19.29	16	yes (0.05)

Table 5.7: RLD approximations using exponential distributions $ed[m]$ for GWSAT and WalkSAT applied to All-Interval-Series instances (using approx. optimal noise); the last column indicates whether the approximation passed the χ^2 test.

to instance `bw_large.b` with exponential distributions. The RLD data is based on 250 runs of each algorithm; the critical χ^2 value for a standard $\alpha = 0.05$ acceptance level is 26.3. The approximations for all algorithms pass the test; only WalkSAT+tabu does not pass for the $\alpha = 0.05$ acceptance level, but for the $\alpha = 0.01$ acceptance level. The corresponding results of the larger Blocks World Planning instances are very similar. This confirms that our result concerning approximately exponential RLDs generalises to WalkSAT+tabu, Novelty, and R-Novelty also when applied to hard instances of the Blocks World Planning domain.

All-Interval-Series For the All-Interval-Series domain, we already know from Section 4.5 that only GWSAT and WalkSAT show approximately complete behaviour; all other algorithms suffer from stagnation behaviour. Therefore, we study only RLD characterisations for GWSAT and WalkSAT here. As can be seen from the approximation and test data reported in the Table 5.7, the exponential RLD approximations for both algorithms and all three instances passed the χ^2 test for a standard acceptance level of $\alpha = 0.05$. Interestingly, for this problem domain, the effect of the initial search phase seems to be much less prominent than for the other problem classes studied here.

In summary, the results presented in this section show that some of the most powerful SLS

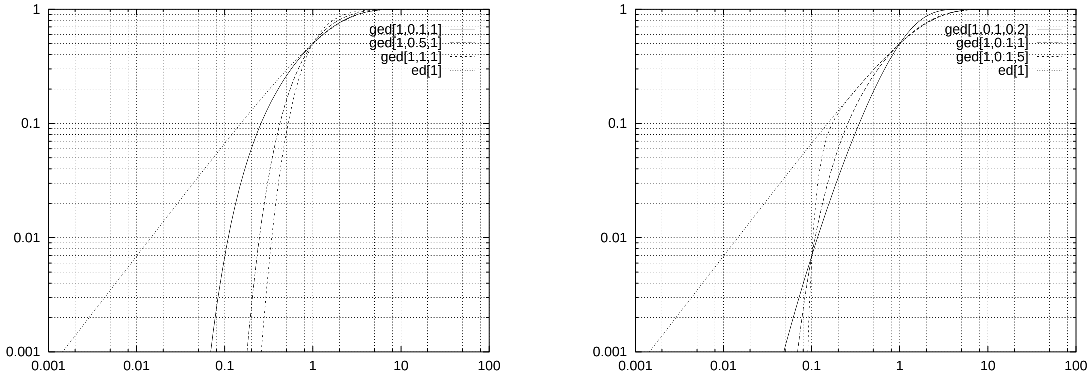


Figure 5.5: The new, generalised distribution class $ged[m, \gamma, \delta]$ for different γ and δ values; note the difference to the standard exponential distribution.

algorithms for SAT show a very regular behaviour when applied to hard instances from a broad range of both problem domains: Using optimal noise parameter settings, the RLDs for these algorithms are approximately exponentially distributed. Furthermore, the quality of the corresponding functional approximations monotonically increases with the hardness of problem instances.

5.1.3 Modelling the Initial Search Phase

As pointed out before, the RLD approximations using exponential distributions show a systematic deviation at the left end of the RLD graphs which corresponds to the behaviour during the initial search phase. To study this phenomenon in more detail, we developed a new probability distribution which facilitates modelling the run-time behaviour during the initial search phase in great detail. This new distribution is based on the Weibull distribution[6] $wd[m, \beta](x) = 1 - 2^{-(x/m)^{\beta}}$, a well-known generalisation of the exponential distribution; the underlying intuition is that during the initial search phase, the performance of an SLS algorithm (*i.e.*, its probability of finding a solution within a fixed number of steps) is lower than later in the search process. This can be modeled by a Weibull distribution with a dynamically changing β parameter (which controls the steepness of the cumulative distribution curve in a semi-log plot).

Our new model is defined by the following cumulative distribution function:[7]

$$ged[m, \gamma, \delta](x) = wd[m, 1 + (\gamma/x)^{\delta}](x) = 1 - 2^{-(x/m)^{1+(\gamma/x)^{\delta}}}$$

Although the full defining term looks quite complicated, the definition reflects exactly the idea of a Weibull distribution with dynamically changing β parameter as given above.

[6] Weibull distributions are used in reliability theory to model failure rates in systems which are subject to aging phenomena [KGC77].

[7] We denote it *ged*, because it is a generalised form of the exponential distribution, as discussed later.

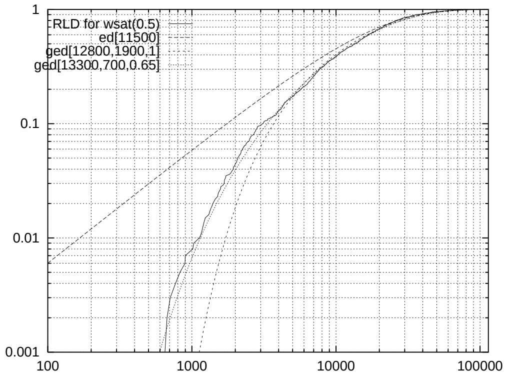


Figure 5.6: RLD for WalkSAT (approx. optimal noise) applied to Blocks World instance `bw_large.a` and three functional approximations.

distribution	χ^2	ν	passed (α)
$ed[11800]$	93.56	29	no
$ged[12800, 1400, 1]$	66.70	28	no
$ged[12800, 600, 0.6]$	37.53	27	yes (0.05)

Table 5.8: RLD approximations for WalkSAT (approx. optimal noise) applied to `bw_large.a`, using different distribution classes; parameters have been manually determined, the last column indicates whether the approximation passed the χ^2 test. The ν values specify the degrees of freedom for the used distribution class; they are reported here for technical reasons only.

More precisely, the new distribution is obtained from a Weibull distribution by introducing a hyperbolically decaying β parameter. Like for the exponential and Weibull distribution, m in the formula above is the median of the distribution. The two remaining parameters intuitively correspond to the length of the initial search phase (γ) and to its impact on the overall behaviour (δ). High γ values indicate a long initial search phase, while high δ values are used to model a strong influence of the initial search phase. Figure 5.5 shows the new generalised distribution for various parameter values.

Note that the exponential distribution is a special case of this new class of distributions, since $ged[m, 0, \delta] = ed[m]$, while generally, Weibull distributions cannot be represented within this family. As can be easily seen from the definition, $ged[m, \gamma, \delta](x)$ asymptotically approaches an exponential distribution $ed[m](x)$ for large x, regardless of the values for γ and δ. For small and decreasing x, however, the relative difference between $ged[m, \gamma, \delta](x)$ and $ed[m](x)$ increases monotonically, which is qualitatively exactly what we observed for the empirical RLDs of SLS algorithms.

problem instance	χ^2 for $ed[m]$	χ^2 for $ged[m,\gamma,1]$	χ^2 for $ged[m,\gamma,\delta]$	ν
medium	(198.47)	40.68	33.57	29–27
huge	(69.58)	31.96	20.74	29–27
bw_large.a	(74.21)	35.57	25.31	29–27
bw_large.b	27.50	16.51	16.46	16–14
bw_large.c	21.43	21.43	21.43	16–14

Table 5.9: Blocks World Planning instances, functional approximation of RLDs for WalkSAT+tabu (approx. optimal noise) with different distribution classes; χ^2 values in parentheses indicate that the approximation did not pass the test for $\alpha = 0.01$.

Because of computation time limitations and for brevity's sake we exemplify the RLD analysis based on the new distribution only for a selection of algorithms and benchmark problems. Figure 5.6 shows the RLD of WalkSAT (approx. optimal noise = 0.5) when applied to blocks world instance `bw_large.a` and three successively more precise functional approximations (these approximations have been manually determined). While the approximation using an exponential distribution fits very well for the upper part of the empirical RLD, it does not adequately model the algorithm's behaviour for shorter runs and consequently does not pass the χ^2 test (*cf.* Table 5.8). The new, generalised distribution gives a significantly better approximation even when fixing the δ parameter to one. While this approximation is often sufficient for characterising the initial search phase, in this example, it still does not pass the χ^2 test. However, using the δ parameter as an additional degree of freedom, we can model the given RLD so precisely that the corresponding approximation passes the χ^2 test.

As shown in Table 5.9, using *ged* approximations we can characterise the behaviour of WalkSAT+tabu (approx. optimal noise) when applied to the Blocks World instances such that all approximations pass the χ^2 test. Interestingly, we can achieve this even when fixing δ to one. Analogous results can be shown for the other SLS algorithms. The advantages of the *ged* approximation over *ed* approximations are reciprocal to the size of the instances. This is consistent with our earlier observation that for smaller and easier instances the initial search phase is more prominent. On the other hand, for large and/or very difficult problem instances, like `bw_large.c`, by using *ged* approximations no improvement of the approximation can be achieved. In particular, this can be observed for the All-Interval-Series instances; here, *e.g.*, when characterising WalkSAT's behaviour using using exponential approximations, these generally pass the χ^2 test for all, even the smallest, instances. While using *ged* approximations, minor improvements can be achieved for the smallest instance, for the other instances these do not realise any advantage over standard exponential approximations.

Next, we used *ged* approximations of SLS behaviour for Random-3-SAT and Graph Colouring test-sets. Table 5.10 shows the results of approximating RLDs of R-Novelty applied to the `uf100-430` test-sets with different distribution classes. While using *ed* approximations, 12.9% of the instances passed the χ^2 test for the $\alpha = 0.01$ acceptance level; by using *ged*'s

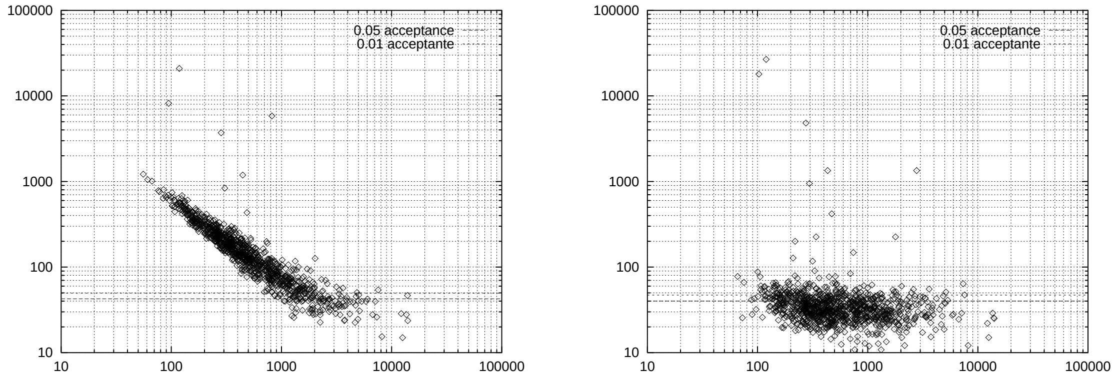


Figure 5.7: Correlation between hardness of problems (horizontal) and χ^2 values (vertical) from testing the RLDs of individual instances versus a best-fit *ed (left)* and *ged* approximation *(right)* for R-Novelty (approx. optimal noise = 0.7) applied to Random-3-SAT test-set `uf100-430`. The dashed horizontal lines indicate the acceptance thresholds for the 0.01 and 0.05 acceptance level.

distribution class	acceptance level α	fraction passed
$ed[m]$	0.01	12.9%
$ed[m]$	0.05	8.7%
$ged[m, \gamma, 1]$	0.01	66.2%
$ged[m, \gamma, 1]$	0.05	52.9%
$ged[m, \gamma, \delta]$	0.01	85.0%
$ged[m, \gamma, \delta]$	0.05	70.0%

Table 5.10: Random-3-SAT, test-set `uf100-430`, functional approximation of RLDs for R-Novelty (approx. optimal noise = 0.7) with different distribution classes.

distribution class	acceptance level α	fraction passed
$ed[m]$	0.01	2.5%
$ed[m]$	0.05	1.5%
$ged[m, \gamma, 1]$	0.01	63.4%
$ged[m, \gamma, 1]$	0.05	50.5%
$ged[m, \gamma, \delta]$	0.01	78.0%
$ged[m, \gamma, \delta]$	0.05	67.6%

Table 5.11: Graph Colouring, test-set `flat50-115`, functional approximation of RLDs for Novelty (approx. optimal noise = 0.6) with different distribution classes; essentially incomplete behaviour was observed for 9 of 1,000 instances, these were removed from the test-set.

with $\delta = 1$, the acceptance rate could be increased to 66.2%; approximating the RLDs with unrestricted *ged*'s, 85.0% of the instances passed the test. At the same time, as can be seen in Figure 5.7, by using the new, generalised distribution, the negative correlation between the hardness of instances and the goodness of functional RLD approximations which is very prominent for exponential approximations is no longer observed. This indicates that, indeed, *ged* approximations are an adequate model for SLS behaviour over the whole range of instance hardness.

Applying the same correlation analysis to Novelty and Graph Colouring test-set `flat50-115` yields analogous results (*cf.* Table 5.11). Note that since these problem instances are very easy for Novelty, the effect of the initial search phase on the RLDs is very strong; consequently, the acceptance rates of the χ^2 test, when applied to *ed* approximations, are very low. Here, by using *ged* approximations which adequately model the initial search phase an even more drastic improvement can be achieved: while for *ed* approximations, only 2.5% of the instances passed the χ^2 test for the $\alpha = 0.01$ acceptance level, this acceptance rate is increased to 78.0% when using *ged*'s.

Nevertheless, one might ask why for a small fraction of problem instances even *ged* approximations are not good enough to pass the χ^2 test. Figure 5.8 shows the RLD corresponding to the most extreme outlier from Fig. 5.7 with a χ^2 value of 23,753.20 for the best *ged* approximation. This RLD is multi-modal and shows a premature stagnation behaviour of the algorithm between 200 and 1,000 steps; then, however, the perfomance improves again and the RLD converges to one. This example seems to be typical for the instances in which the given RLD cannot be adequately characterised by *ged* approximations. The intuitive interpretation of this behaviour is that initially the algorithm behaves in a regular way (which can be characterised by a *ged* approximation), but then gets stuck in some sort of "trap". It will, however, eventually escape from this trap and then continue its search in a regular way. These traps probably correspond to attractive non-solution areas of the search space (*e.g.*, complex local minima areas) which are hard to escape for the algorithm. As also shown in Fig. 5.8, a combination of two *ged* functions reflecting this interpretation

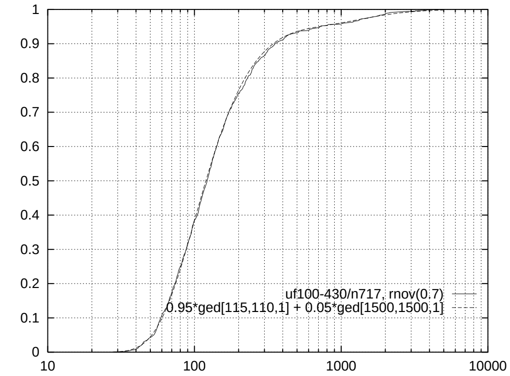


Figure 5.8: RLD for R-Novelty (approx. optimal noise = 0.7) applied to instance `uf100-430/n717`; note, how using a combination of two overlayed *ged* approximations, the observed behaviour can be almost perfectly modeled.

algorithm	median of RLD	m	χ^2 for $ed[m]$	passed (α)
wsat(0.5)	13,505	13,094.9	98.41	no
wsat(0.6)	15,339	16,605.5	76.17	no
wsat(0.7)	27,488	27,485.9	42.13	yes (0.5)
wsat(0.8)	72,571	74,289.8	20.78	yes (0.5)

Table 5.12: $ed[m]$ approximations of RLDs for WalkSAT applied to Blocks World Planning instance `bw_large.a`, using optimal and larger-than-optimal noise parameter settings.

gives an almost perfect approximation of the observed RLD. This strongly suggests that simple combinations of *ged* approximations can be used to adequately model and explain the irregular behaviour observed for a small fraction of problem instances from our test-sets.

Results analogous to the ones presented here could be obtained for other WalkSAT algorithms when applied to various test-sets and instances from our benchmark suite. In summary, our analysis shows that using the new probability distribution type *ged* developed here, we can model the run-time behaviour of state-of-the-art SLS algorithm for SAT in great detail. In particular, this extended model captures the SLS behaviour during the initial search phase very accurately.

5.1.4 Functional Approximations for Non-optimal Noise

While up to this point, we always concentrated on approximately optimal noise parameter settings, in the following, we present some results regarding SLS behaviour for non-optimal noise. For larger-than-optimal noise settings, we observe essentially the same EPAC be-

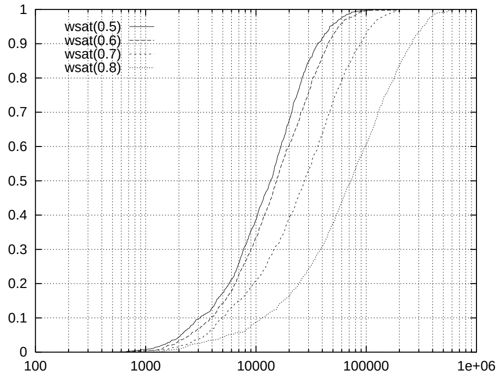


Figure 5.9: RLDs for WalkSAT applied to Blocks World Planning instance `bw_large.a`, using different noise parameter settings (0.5 is approx. optimal).

haviour as for optimal noise, but run-times compared to optimal noise are uniformly higher for all success probabilities. This can be easily seen in the semi-log plots of the corresponding RLDs, where for larger-than-optimal noise, the RLD curves have the same shape while being shifted to the right. At the same time, when increasing the noise, the effects of the initial search phase become less prominent — consequently, the approximations using exponential distributions are usually better for higher noise. This can be seen in Figure 5.9, showing the RLDs for WalkSAT with different noise settings when applied to Blocks World Planning instance `bw_large.a` (RLD data is based on 1,000 runs). The results for a χ^2 test, using *ed* approximations, are reported in Table 5.12. For optimal noise, the *ed* approximation does not pass the test, because the deviation caused by the initial search phase is too strong. While this effect decreases for a slightly increased noise setting (as can be seen from the lower χ^2 value), the *ed* approximation still does not pass the test. When further increasing the noise setting, the initial search phase becomes less prominent and the *ed* approximation passes the χ^2 test. It should be noted, however, that when using *ged* approximations (*cf.* Section 5.1.3), the initial search phase can be correctly modeled in all three cases in such a way that the corresponding RLD approximations pass the χ^2 test. Note also the increasing median run-times, indicating the decrease of performance for larger-than-optimal noise settings.

As a second example, we report the results of analysing the effects of increasing the noise parameter for R-Novelty, applied to the Graph Colouring test-set `flat50-115`. Table 5.13 shows the fraction of instances from this test-set, for which a best-fit *ed* approximation passed the χ^2 test. As can be seen from this data, the fraction of the hardest 25% of the test-set which passed the test increases for larger-than-optimal noise, as the effects of the initial search phase become less prominent. At the same time the median hardness (mean number of steps / solution) increases considerably. Analogous results could be obtained for different SLS algorithms and other problem domains from our benchmark suite.

While for larger-than-optimal noise, the SLS performance decreases but the algorithms still

algorithm	median hardness	acceptance level α	fraction passed
r-novelty(0.6)	739.61	0.01	19.0%
		0.05	13.3%
r-novelty(0.7)	832.92	0.01	23.8%
		0.05	14.9%
r-novelty(0.8)	1,043.29	0.01	40.7%
		0.05	24.2%

Table 5.13: Fraction of instances from the hardest 25% of the `flat50-115` Graph Colouring test-set passing the χ^2 test for Novelty with optimal and larger-than-optimal noise settings; instances, for which essentially incomplete behaviour was observed, were removed from the test-set.

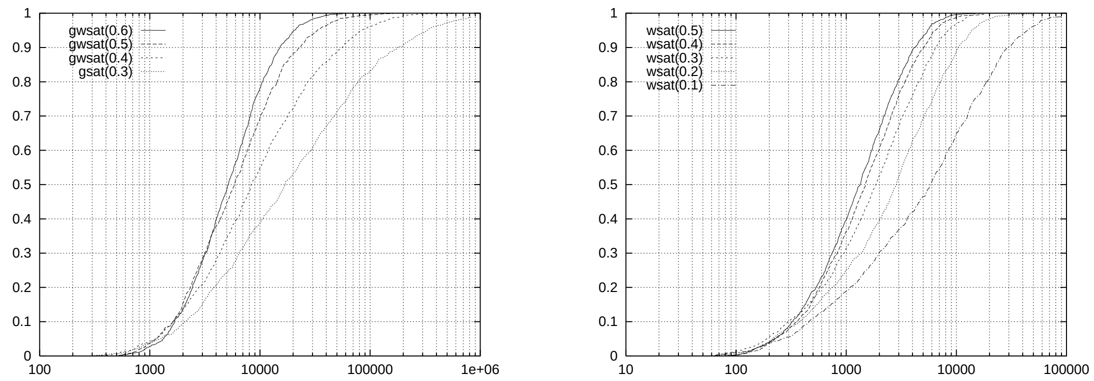


Figure 5.10: *Left:* RLDs for GWSAT applied to easy instance from Graph Colouring test-set `flat100-239`, using different noise parameter settings (0.6 is approx. optimal); *right:* same for WalkSAT applied to medium instance from Random-3-SAT test-set `uf100-430` (approx. optimal noise = 0.5).

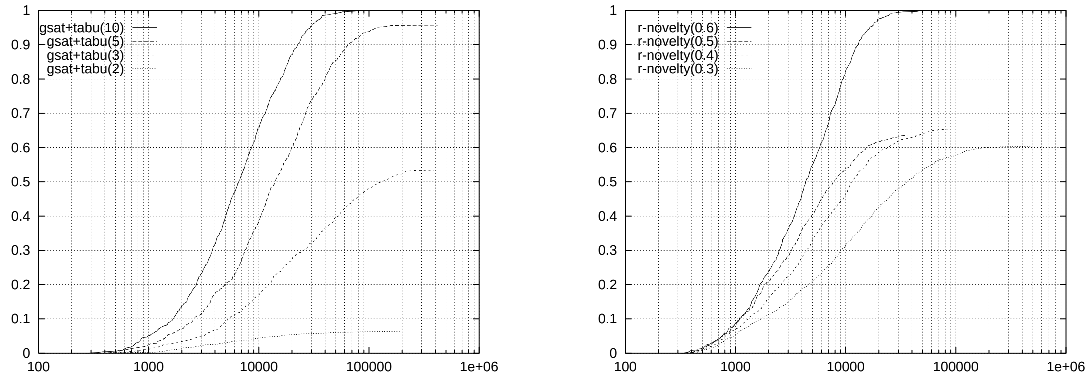


Figure 5.11: *Left:* RLDs for GSAT+tabu applied to medium instance from Graph Colouring test-set `flat100-239`, using different noise parameter settings (10 is approx. optimal); *right:* same for R-Novelty applied to Blocks World Planning instance `bw_large.a` (approx. optimal noise = 0.6).

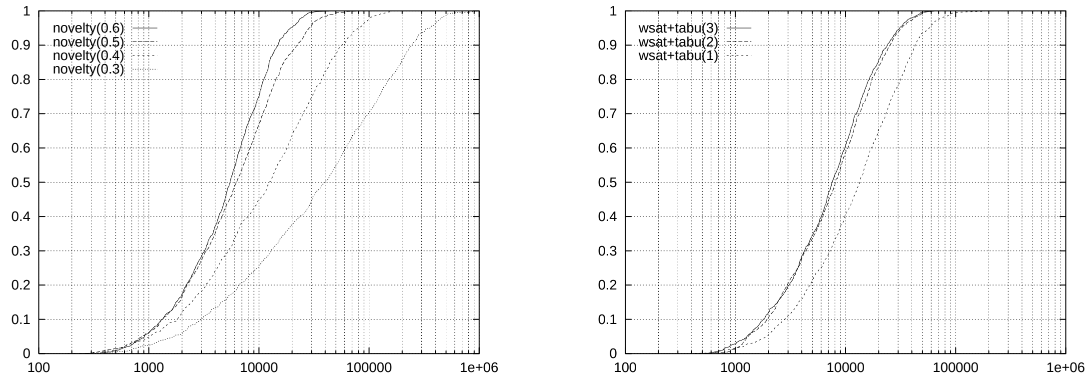


Figure 5.12: *Left:* RLDs for Novelty applied to medium instance from Graph Colouring test-set `flat100-239`, using different noise parameter settings (0.6 is approx. optimal); *right:* same for WalkSAT+tabu applied to Blocks World Planning instance `bw_large.a` (approx. optimal noise = 3).

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}
gwsat(0.2)	595,388.77	580,061.58	0.97	413,064	168,375	797,305	66,640	1,431,559
gwsat(0.3)	69,778.76	73,354.57	1.05	47,511	20,531	96,839	7,134	158,442
gwsat(0.4)	64,167.11	64,661.00	1.01	43,128	19,079	85,901	7,150	143,875
gwsat(0.5)	79,258.66	82,593.77	1.04	50,887	22,172	109,799	8,387	181,464
gwsat(0.6)	169,139.57	170,347.50	1.01	115,024	49,558	229,213	17,247	389,196

Table 5.14: $ed[m]$ approximations of RLDs for GWSAT applied to All-Interval-Series instance `ais8`, using approx. optimal (=0.4) and non-optimal noise parameter settings.

show EPAC behaviour, for lower-than-optimal noise parameter settings we find a completely different situation. While for all algorithms, the performance decreases non-uniformly for different solution probabilities, some of them remain approximately complete, while others show essentially incomplete behaviour. We demonstrate this here by giving a small number of examples, which are nevertheless typical for our overall observations. As shown in Figure 5.10, GSAT and WalkSAT show a similar behaviour for smaller-than-optimal noise: While for short runs, the performance even improves, for longer runs and the corresponding higher success probabilities, the performance deteriorates considerably. Consequently, the RLDs are less steep than exponential distributions and suffer from an increasingly long and heavy tail. Both GSAT+tabu and R-Novelty show a similar, but additionally essentially incomplete behaviour; as can be seen from Figure 5.11, the maximal success probabilities for both algorithms are decreasing with the noise parameter. For Novelty, no increasingly essential incomplete behaviour is observed, but the performance also decays non-uniformly as for GSAT and WalkSAT (*cf.* Figure 5.12). In comparison, WalkSAT+tabu seems to be relatively mildly affected; the main reason for this might be that typically, both the clause-length and the optimal tabu-list-length are rather small.

Summarising these results, typically the detrimental effects of non-optimal noise parameter settings are much worse for lower-than-optimal noise settings than for larger-than-optimal noise. This is particularly the case for the higher percentiles of the corresponding RLDs (*cf.* Table 5.14); additionally, for essentially incomplete SLS algorithms, stagnation behaviour occurs more frequently with decreasing noise settings. However, the behaviour for very short runs is usually not affected by lower-than-optimal noise and sometimes, even performance improvements can be observed for the early phases of local search (this is consistent with the intuition that greedier local search behaviour should pay off during the initial search phase, where gradient descent dominates the search.)

5.1.5 Consequences and Interpretations

The empirical results presented in the previous sections have a number of theoretically and practically interesting consequences.

Random restart For all of the SLS algorithms studied here, the standard implementations include a restart mechanism which allows to restart the local search process after a fixed maximal number maxSteps of steps, if at that point no solution has been found. Restarting the local search is the most simple method for escaping from local minima, and for essentially incomplete algorithms like plain GSAT, using restart leads to significantly improved performance. For EPAC algorithms, however, restart is not effective, since in the case of exponential run-time distributions the probability of finding a solution within a fixed time interval is independent of the run-time spent before. These SLS algorithms are essentially memoryless, as for a given total time t, restarting at time $t' < t$ does not significantly influence the probability of finding a solution in time t.[8] Based on the results reported before, many modern SLS algorithms for SAT show EPAC behaviour for optimal and larger-than optimal noise settings when applied to our benchmark problems. Consequently, for these algorithms random restart is mostly ineffective. However, due to the effects of the initial search phase, this does not hold for relatively easy problem instances and short run-times. However, in practice the ineffectivity of random restart for hard instances can be easily verified over a broad range of maxSteps settings (this will be exemplified in Section 5.2.2 in a slightly different context). As we have seen in Section 5.1.4, the EPAC property is usually lost when using lower-than-optimal noise parameter; in this case, random restart is advantageous for appropriately chosen cutoff times (*cf.* Chapter 2).

Search space reduction One surprising consequence of our characterisation results is based on the well-known fact that blind guessing (*i.e.*, repeatedly testing a randomly selected variable assignment), one of the most naïve search methods, is also characterised by exponential RLDs. Therefore, our results suggest that the behaviour of modern, powerful SLS algorithms for optimal and larger-than-optimal noise settings can be interpreted as blind guessing in a significantly reduced search space. More precisely, these SLS algorithms, when applied to a problem instance with search space S and search space size $|S|$, behave exactly like blind guessing in a "virtual search space" S'. Assuming that S' has the same (known) number of solutions as S, the size $|S'|$ of the virtual search space can be easily calculated from the empirical RLD of the given SLS algorithm A and the original search space size $|S|$. For a given problem instance, $|S'|$ gives an indication for the effectiveness of algorithm A: the smaller $|S'|$, the more effective is A. This can be easily generalised to test-sets of problems or problem distributions by combining the $|S'|$ values for the individual instances. Note that this interpretation holds for all situations where an SLS algorithm applied to a given problem instance or problem class shows EPAC behaviour, *i.e.*, it exhibits exponential RLDs.

This result immediately raises the following question: How is the virtual search space related to the actual search space? For now, we have no conclusive answer to this question. But ideally, we would like to be able to identify structural features of the original search space which can be shown to be tightly correlated with the virtual search space size. One would

[8] Because the exponential distribution is memoryless, it is often used in reliability theory to describe components that are not subject to aging phenomena like transistors [BP81].

assume that size and number of plateaus (including those consisting of solutions) in the original search space should be important factors. However, preliminary studies have shown that these features alone are not sufficient for explaining the observed search space reduction phenomenon.

5.2 Improving Existing SLS Algorithms

After characterising the behaviour of several state-of-the-art SLS algorithms for SAT, in this section we show how based on these results the algorithms can be further improved. In the following, we discuss three ways of improving SLS algorithms based on the empirical analyses given before: using hybrid GLSM extensions, multiple independent tries, and algorithm portfolios. While the first approach is used to improve the behaviour of specific SLS algorithms, the other two approaches are very general and can be exploited for parallel as well as sequential time-sliced processing.

5.2.1 Hybrid GLSM Extensions

As discussed in Section 5.1.1, some of the most powerful SLS algorithms for SAT, like WalkSAT+tabu, Novelty, and R-Novelty suffer from essential incompleteness. In practice, this can be easily overcome by introducing random restarts. Obviously, if for a given cutoff time t the probability of finding a solution for a given problem instance is $p > 0$, the probability of finding a solution within n independent runs is $p' = 1-(1-p)^n$ which converges to 1 for $n \to \infty$. Therefore, by introducing random restart after a fixed cutoff time, any Las Vegas algorithm can be made approximately complete. Moreover, the modified algorithm will have an approximately exponential run-time distribution for large numbers of restarts.

In the context of the GLSM model, random restart is realised by connecting at least one local search state with the initialisation state by a deterministic transition with condition $\mathsf{mcount}(\mathsf{maxSteps})$. This is realised in the implementations of all the SLS algorithms for SAT discussed in the context of this work (*cf.* Sections 4.2 and 4.3). In this sense, all the actual implementations correspond to hybrid GLSM extensions obtained from the pure local search strategy by adding restart.

Unfortunately, in general it is extremely difficult to find good cutoff times *a priori* (*a posteriori* they can be easily derived from the RLD data, as detailed before). Today, to our best knowledge, there exist no theoretical results on how to determine good cutoffs. The only empirical results we are aware of, are for GSAT when applied to hard Random-3-SAT problem distributions [GW93a]; but since these results are based on potentially problematic methodology, their significance is unclear (*cf.* Section 2.3). While using inappropriately chosen cutoffs generally still eliminates essential incompleteness, in practice, it leads to extremely poor performance.

Another way of overcoming essential incompleteness of a SLS algorithm is to extend it with a random walk state. This extension can again be interpreted in the GLSM framework by adding a random walk state and connecting it to the local search state(s) via unconditional probabilistic transitions. As an example, consider GSAT and GWSAT; this also demonstrates that adding random walk induces approximate completeness, as proven for GWSAT in Section 5.1.1. As mentioned there, the proof for GWSAT's approximate completeness easily generalises for all algorithms which guarantee that arbitrarily long sequences of random walk steps can be performed with a probability $p > 0$ which is independent of the number of steps that have been performed in the past. Thus, any GLSM G satisfying the following conditions will provably correspond to a PAC algorithm:

1. G has a random walk state RW;
2. RW has a probabilistic, unconditional self-transition;
3. for any point in time, there is a positive, bounded probability for G being in state RW, *i.e.*, $\forall t : \exists p' > 0 : P(G \text{ is in state } RW \text{ at time } t) \geq p'$.

As a convention, we assume that the random walk state has a bounded, positive probability of decreasing the hamming distance to the nearest solution in each step, as is the case for the random walk state used by GWSAT (*cf.* Chapter 4, Section 4.2 for formal definition). While these conditions are sufficient for guaranteeing approximate completeness, they can be further weakened. But because of the simplicity of the algorithms we are dealing with here, the conditions as given above are fully adequate for our purposes. More specifically, since all GSAT and WalkSAT variants discussed here are 1-state+restart GLSMs, we extend them with random walk using the generic scheme shown in Figure 5.13, where LS is any local search state and RW the random walk state as defined for GWSAT. It can be easily verified that this scheme satisfies the conditions for approximate completeness as given above; therefore, any instantiation will show approximately complete behaviour.

Random walk apparently has an advantage over random restart, since at least in GWSAT, it is more robust w.r.t. the additionally introduced parameter wp than random restart is w.r.t. maxSteps (the cutoff time). One reason for this empirically observed phenomenon is related to the inherent randomness of the random walk sequences; random restarts occur after a fixed cutoff time, whereas random walk sequences are probabilistically variable in their length and frequency of occurrence. Furthermore, when using random restart, the search process is re-initialised; consequently, a new try cannot benefit from the search effort spent in previous tries (unless information is carried from one run to the next, which is not the case for the algorithms considered here). The amount of perturbation introduced by a random walk sequence, however, probabilistically depends on the length of the sequence such that small pertubations are much more likely to occur.

We use the random walk extension here to improve the behaviour of Novelty and R-Novelty. For R-Novelty, we replace the deterministic loop breaking strategy, which is essentially equivalent to performing one random walk step each 100 steps, by a random walk extension as described above; for Novelty, we newly introduce an equivalent extension. We call

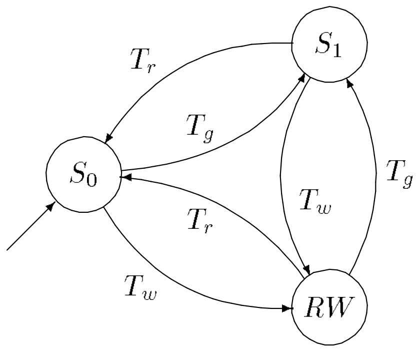


Figure 5.13: Random walk extension of a 1-state+restart GLSM; transition types: $T_r \equiv CPROB(C,1)$ with condition $C \equiv \mathsf{mcount}(\mathsf{maxSteps}); T_w \equiv CPROB(\neg C, \mathsf{wp}); T_g \equiv CPROB(\neg C, 1-\mathsf{wp})$.

these modified algorithms R-Novelty$^+$ and Novelty$^+$; their approximate completeness (PAC property) follows immediately using the argument given above.

Although these modified algorithms have theoretically considerable advantages, *a priori*, it is not clear how their performance compares with the original versions in practice. Table 5.15 shows the basic descriptive statistics for the mean local search cost (hardness distribution) across two Random-3-SAT and Graph Colouring test-sets (the data is based on 100 runs / instance, using $\mathsf{maxSteps} = 10^6$.). As can be seen when comparing the statistics for the original and modified versions of Novelty and R-Novelty, the latter show dramatically reduced means and standard deviations. The percentiles, however, are not too different, although for the Random-3-SAT test-set, the higher percentiles are also significantly lower. This indicates that the essentially incomplete behaviour which causes the high means and standard deviations of these hardness distributions is efficiently eliminated by the random walk extension, while for instances which do not suffer from essentially incomplete behaviour, the performance remains mainly unaffected. This interpretation is confirmed by the RLD statistics for Novelty and R-Novelty and their respective modified versions when applied to Blocks World Planning and All-Interval-Series instances (*cf.* Tables 5.16 and 5.17, all RLD data is based on 250 or more runs of each algorithm). Note that, as discussed earlier, even when using random restart, essentially incomplete behaviour makes an algorithm's performance extremely dependent on appropriately chosen cutoff parameters. This effect is especially drastic for extremely low asymptotic success probabilities, as observed for Novelty and R-Novelty when applied to large and hard Blocks World Planning and All-Interval-Series instances.

In summary, these results indicate that R-Novelty$^+$ and Novelty$^+$ show significantly improved performance over the original algorithms where these suffered from essentially incomplete behaviour, while in the remaining cases the performance is approximately identical. Thus, extending Novelty and R-Novelty with random walk generally improves their

test-set	algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}
uf100-430	novelty(0.6)	28,257.51	191,668.71	6.78	851.87	479.14	1,845.30	302.88	4,390.39
	novelty+(0.6)	1,570.39	2,802.01	1.78	801.75	467.04	1,663.20	288.72	3,049.70
flat50-115	r-novelty(0.6)	7,109.64	97,543.64	13.72	739.61	490.34	1,282.53	356.66	2,142.78
	r-novelty+(0.6)	1,084.01	1,053.71	0.97	747.37	486.65	1,245.42	358.74	2,167.39

Table 5.15: Performance comparison for Novelty and R-Novelty (approx. optimal noise) *vs* Novelty$^+$ and R-Novelty$^+$ for Random-3-SAT and Graph Colouring test-sets. The reported basic descriptive statistics refer to the hardness distributions (mean number of steps / solution) over the test-sets.

instance	algorithm	$\widehat{p_s}^*$	maxSteps	$\widehat{E_s}$	mean$_c$	stddev$_c$	median$_c$
bw_large.c	r-novelty(0.3)	0.028	10^8	$3.47 \cdot 10^9$	169,810.86	157,752.57	32,334
	r-novelty+(0.3)	1.0	10^8	$8.09 \cdot 10^6$	8,086,468.24	8,414,928.96	5,292,830
ais6	novelty(0.7)	0.101	10^6	$8.90 \cdot 10^6$	62.65	44.60	46
	novelty+(0.7)	1.0	10^6	$9.94 \cdot 10^3$	9,944.67	10,828.49	6,909
ais8	r-rnovelty(0.8)	0.916	10^7	$1.06 \cdot 10^6$	139,779.01	135,746.86	99,880
	r-novelty+(0.8)	1.0	10^7	$1.51 \cdot 10^5$	151,051.11	152,669.01	112,413
ais10	r-rnovelty(0.7)	0.012	10^8	$8.23 \cdot 10^9$	7,603.33	1,667.32	7,313
	r-novelty+(0.7)	1.0	10^8	$2.41 \cdot 10^6$	2,410,863	2,240,534	1,697,646

Table 5.16: Performance comparison for Novelty and R-Novelty (approx. optimal noise) *vs* Novelty$^+$ and R-Novelty$^+$ for Blocks World Planning and All-Interval-Series instances; $\widehat{p_s}^*$ indicates the asymptotic maximal success probability, $\widehat{E_s}$ denotes the expected number of steps for finding a solution (using random restart); the reported basic descriptive statistics refer to the corresponding *conditional* RLDs.

instance	algorithm	mean	stddev	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}
bw_large.a	novelty(0.4)	9,315.07	8,587.17	6,932	3,202	12,877	1,534	20,202
	novelty+(0.4)	9,598.46	8,852.51	6,730	3,253	13,198	1,611	20,522
bw_large.b	r-novelty(0.55)	234,305.18	215,100.55	166,922	73,673	341,827	21,524	538,266
	r-novelty+(0.55)	223,493.79	251,273.29	141,353	53,223	315,074	22,499	496,085

Table 5.17: Performance comparison for Novelty and R-Novelty (approx. optimal noise) *vs* Novelty$^+$ and R-Novelty$^+$ for Blocks World Planning instances when no essentially incomplete behaviour is observed.

performance. However, it should be noted that these algorithms, as well as WalkSAT+tabu, can be also improved in a different, more specific way. Re-examining the proof for Novelty's essential incompleteness (*cf.* Section 5.1.1), one might have noted that the looping behaviour for the given example is caused by the fact that Novelty always picks between the best and second-best variable. What is needed to break the loop in the example given in the proof, is a mechanism which allows y to be flipped when c_3 is selected. But in this situation, y's score will always be lower than x_1's and x_2's. Therefore, to generally prevent this situation, a picking mechanism is needed which allows variables with arbitrarily bad scores to be picked once in a while. Similar considerations hold for R-Novelty. Thus, by modifying the picking mechanism according to this requirement, the essential incompleteness of both Novelty and R-Novelty can be removed. Note that adding a random walk state is equivalent to one specific modification of this kind. Other possibilities include weighting the probabilistic choice of the variable to be flipped by the score. However, it is not clear whether such variants show a best-case performance comparable to the original algorithms'.

For WalkSAT+tabu, the main reason for essential incompleteness when using long tabu-lists is the fact that, if all variables in a clause are tabu, none of them is flipped. This problem can be easily overcome by allowing a uniform or score-weighted random variable selection in this situation. Note, however, that — as opposed to the random walk extension discussed above — in general this will be insufficient to guarantee approximate completeness.

5.2.2 Parallelisation Based on Multiple Independent Tries

Randomised algorithms lend themselves to a straightforward parallelisation approach by performing independent runs of the same algorithm in parallel. In the given context, this is equivalent to using a homogenous cooperative GLSM model without communication as discussed in Chapter 3.1, Section 3.8. If the base algorithm is EPAC, this approach is particularly effective: Based on a well-known result from the statistical literature [Roh76], if for a given algorithm the probability of finding a solution in t time units is exponentially distributed with parameter m, *i.e.*, the RLD can be characterised by $ed[m]$, then the probability of finding a solution in p independent runs of time t is distributed along $ed[m/p]$. Consequently, if we run such an algorithm once for time t, we get exactly the same success probability as when running the algorithm p times for time t/p. This means that using multiple independent runs of an EPAC algorithm, an optimal speedup can be achieved for arbitrary numbers of processors.

In practice, however, SLS algorithms are not perfectly EPAC due to the systematic deviations during the initial search phase (*cf.* Section 5.1.3). Furthermore, usually a setup time (for initialising data structures *etc.*) has to be taken into account. Therefore, for very short runs (*i.e.*, very high numbers of processors), the speedup will generally be less than optimal. More precisely, for a given lower bound s' on the speedup, a minimal cutoff time t' and, subsequently, a maximal number of processors p' can be derived from the RLD data such that for $p < p'$, a speedup $s \geq s'$ will be realised. However, if the RLDs converge against exponential distributions and the initial search phase and setup time are rather short compared

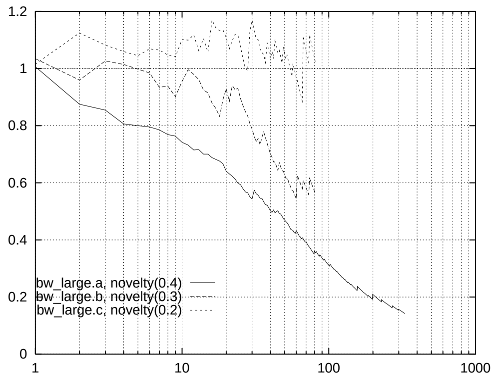


Figure 5.14: Efficiency of multiple independent try parallelisation (vertical) *vs* number of processors (horizontal) for Novelty (approx. optimal noise settings) applied to Blocks World Planning instances. An efficiency value of 1 indicates optimal speedup.

to the time required for solving typical problem instances, this effect becomes almost negligible. Based on the characterisations from the previous sections, this situation is given for most modern SLS algorithm for SAT when applied to hard problem instances from various domains. Furthermore, it holds for optimal as well as larger-than-optimal noise parameter settings (see also [HS98a]). Thus, parallelisation using independent tries will generally yield almost optimal speedup.

To illustrate this for a concrete example, we analysed the parallelisation efficiency for Novelty applied to several Blocks World instances based on the given RLDs (*cf.* Figure 5.4, page 121). Figure 5.14 shows the predicted parallelisation efficiency E as a function of the number of processors p; $E(p)$ is defined as the speedup $S = T_1/T_p$ divided by p, where T_1 is the sequential and T_p the parallel computation time, using p processors. For Las Vegas algorithms like Novelty, it is reasonable to define the computation times such that T_1 is the cutoff time required to guarantee a solution probability of at least p_s, or technically: $T_1 = min(\{t' \mid P(RT \leq t') \geq p_s\})$. Given T_1 and the empirical RTD or RLD data, we can easily calculate T_p:

$$T_p = min(\{t' \mid \widehat{P}(RT \leq t') \geq 1 - (1 - p_s)^{1/p}\})$$

In our example, we chose $p_2 = 0.95$, *i.e.*, we want to guarantee a success probability of 95%. The curves were cut off at the right end to guarantee that the reported results were based on not less than ten runs of the algorithms; the underlying RLDs were based on 1,000 tries for the smallest instance and on 250 tries for the larger instances. As can be seen from the efficiency graphs, we observe an efficiency of one (corresponding to optimal speedup) up to a certain maximal number p_{max} of processors. This p_{max} value depends directly on the relative length of the initial search phase (*cf.* Sections 5.1.2 and 5.1.3) and, consequently, increases monotonically with instance hardness. In our example, for the

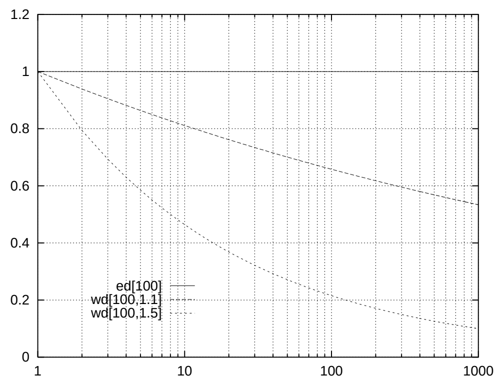


Figure 5.15: Efficiency of multiple independent try parallelisation (vertical) *vs* number of processors (horizontal) if the underlying (hypothetical) algorithm's run-time behaviour can be characterised by a Weibull distribution. An efficiency value of 1 indicates optimal speedup.

smallest instance, `bw_large.a`, the efficiency drops almost immediately when parallelising, while for the largest instance, `bw_large.c`, optimal speedup can still be achieved with ca. 100 processors. Furthermore, p_{max} is positively correlated with the desired success probability p_s such that for tighter success guarantees, the maximal number of processors such that optimal efficiency is still achieved, is higher.

However, in cases, where the RLDs are generally steeper than an exponential distribution, using independent tries will always lead to a suboptimal speedup (*cf.* Chapter 2); again, using RLD data, it is possible to predict the speedup as a function of the number of processors. This is shown for the example of Weibull distributions $wd[m, \beta]$ with $\beta \geq 1$ as RLDs in Figure 5.15; note that while for $\beta = 1$ (exponential distribution) optimal efficiency is observed, with increasing β (*i.e.*, steepness of the distribution) the efficiency decays faster for large number of processors. However, leaving aside the effects of the initial search phase for EPAC algorithms (as discussed above), we have never observed this type of behaviour in our experiments with SLS algorithms for SAT.

For RLDs which are partially less steep than an exponential distribution, optimal cutoff times can be derived as detailed in Chapter 2, Section 2.2.2. This situation can be observed particularly for smaller-than-optimal noise parameter settings (*cf.* Section 5.1.4); here, the optimal cutoff implies an optimal number of processors for which a maximal speedup can be obtained when compared to the sequential case $p = 1$. This is exemplified in Figure 5.16, showing the RLD for GWSAT with a suboptimal noise setting when applied to a small Blocks World Planning instance. The "hump" in the multi-modal RLD indicates the existence of an optimal cutoff which corresponds to the optimal number of processors for multiple independent tries parallelisation represented by the maximum in the efficiency graph. Note that the efficiency compared to the sequential algorithm without restart is

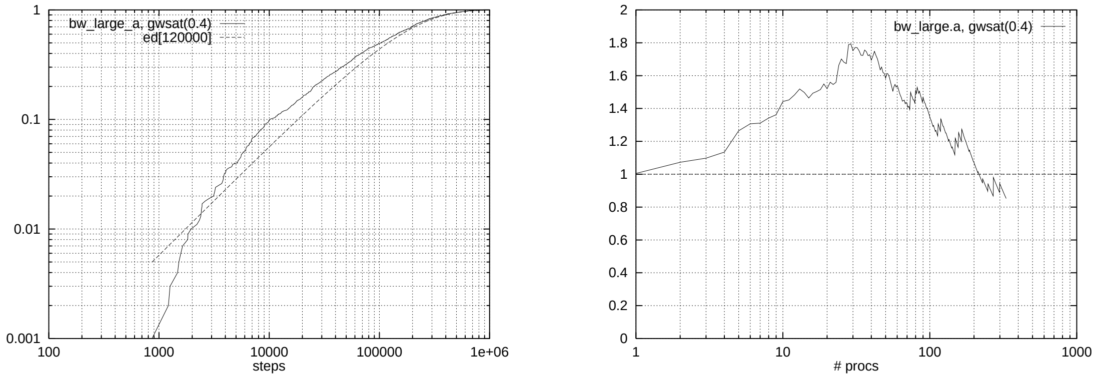


Figure 5.16: GWSAT with suboptimal noise setting (`wp` $= 0.4$) applied to Blocks World Planning instance `bw_large.a`. *Left:* RLD and corresponding asymptotic exponential distribution; *right:* parallelisation efficiency *vs* number of processors ($p_s = 0.95$).

greater than one, *i.e.* super-optimal, over a wide range of parallelisation.

Generally, using multiple independent tries is an extremely attractive model of parallel processing, since it involves basically no communication overhead and can be easily implemented and run on almost any parallel hardware platform, from networks of standard workstations to specialised MIMD machines with thousands of processors. Therefore, these results are not only relevant for the application of SLS algorithms to hard combinatorial problems like SAT to time-critical tasks (like robot control or online scheduling), but also for the distributed solving of very large and hard problem instances.

5.2.3 Parallelisation Based on Algorithm Portfolios

For SLS algorithms, algorithm portfolios are equivalent to the heterogeneous cooperative GLSM model without communication, introduced in Chapter 3. Intuitively, the idea is to execute a set of algorithms ("portfolio") in parallel, applying each individual algorithm to the same problem instance. Note that, like multiple independent tries parallelisation, this scheme can be easily realised on a single-processor system by using processor multiplexing to interleave the execution of the individual algorithms. Multiple independent tries parallelisation can be interpreted as a special case of an algorithm portfolio, where all individual algorithms are identical. Portfolios comprising different algorithms, or differently parameterised instances of an algorithm, only realise advantages over this simpler case if

1. no single individual algorithm dominates all the others, and
2. there are situations in which it is not possible to decide *a priori* which algorithm will be most effective.

If condition 1 is violated, *i.e.*, there is a completely dominant individual algorithm, obviously replacing all other elements of the portfolio with this algorithm will improve the overall

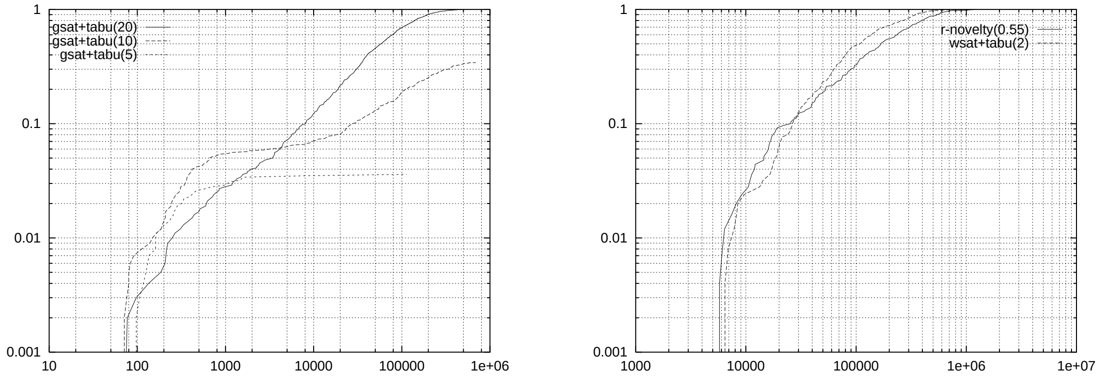


Figure 5.17: *Left:* Crossing RLDs for GSAT+tabu, applied to Random-3-SAT instance `uf100/hard`, using different tabu-list lengths; *right:* crossing RLDs for R-Novelty and WalkSAT+tabu (approx. optimal noise settings) when applied to Blocks World Planning instance `bw_large.b`.

performance. Condition 2 is considerably weaker. Of course, based on an RLD analysis, it is always possible to decide *a posteriori* which algorithm of a given portfolio is best for each problem instance and run-time interval. But only if this information is *a priori* available, it can be used to optimise the portfolio. Since generally, complete domination between algorithm seems to be rather the exception than the rule for hard combinatorial problems, the ability to successfully establish or approximate this *a priori* knowledge would be a great improvement for many problem classes. At least for SAT, the problem of finding methods for successfully predicting dominance relations between different algorithms, or even algorithm classes (such as complete search *vs* incomplete local search) is essentially unsolved. Therefore combining individual algorithms which show superior performance in certain situations or for certain domains into portfolios appears to be a promising method for solving hard SAT problems.

For the SLS algorithms considered here, we have observed different phenomena suggesting that the portfolio approach will be beneficial. First, while for a given problem instance, strict domination between different algorithms and parameterisations seems to be the rule, we occasionally observed crossing RLDs. Within our benchmark suite, such examples could be found for all algorithms presented here. Figure 5.17 shows two different cases for crossing RLDs: On the left-hand side, we see how GSAT+tabu applied to a hard Random-3-SAT instance exhibits crossing RLDs for different tabu-list lengths. Clearly, the shorter tabu-lists give superior performance for shorter run-times, but suffer from stagnation for longer runs. This phenomenon can also be observed for GSAT+tabu when applied to the Graph Colouring domain. The right-hand side of Figure 5.17 shows an example for crossing RLDs between different algorithms using optimal noise settings. Here, R-Novelty is superior to WalkSAT+tabu for short runs, while for long run-times, WalkSAT+tabu is significantly better. Again, a similar situation can be observed for other problem domains from our benchmark suite.

Apart from crossing RLDs, we also observed that for different problem domains, different SLS algorithms show the best performance. The results reported in Chapter 4 suggest that Novelty and R-Novelty are best-performing for random, unstructured problems, while WalkSAT+tabu and GSAT+tabu might be superior for relatively hard, large, structured problem instances, like the large Blocks World Planning instances, the hard Graph Colouring instances, and the All-Interval-Series instances. Thus, to optimise performance over a broad range of domains, or in a situation where the relative performance of the different algorithms is unknown, combining several of the best-performing algorithms mentioned above into a mixed portfolio is advantageous. Furthermore, such a portfolio can also include promising complete algorithms, like Satz [LA97] or Satz-Rand [GSK98], which outperform the best currently known SLS algorithms for certain highly structured problem domains, like Quasigroup Completion, or All-Interval-Series.

Given the current state of knowledge regarding the performance of SAT algorithms across different domains, we conjecture that for solving SAT instances from a broad range of problem domains, or instances the structure of which is largely unknown, heterogeneous portfolios comprising powerful algorithms, like Novelty$^+$, R-Novelty$^+$, WalkSAT+tabu, and GSAT+tabu, as well as state-of-the-art complete algorithms, like Satz-Rand, will prove to be the best and most robust solution methods. Such portfolios can probably be further enhanced by including different parameterisations (*e.g.*, noise settings) for the different algorithms; and in a last step, performance might be even further improved by automatically optimising the portfolios within single runs or over a number of runs, in the spirit of the evolutionary cooperative GLSM models discussed in Chapter 3.

5.3 Related Work

Although SLS algorithms for SAT, in particular GSAT and WalkSAT variants, have been intensively studied in the past, there is only a small number of studies which attempt to characterise SLS behaviour. One of these investigates the behaviour of GSAT and differentiates its behaviour into an initial hill-climbing and subsequent plateau phases [GW93a]. These phases are mathematically modeled; however, the characterisation is based on averaged local search trajectories on hard Random-3-SAT instances rather than on RLDs. A more recent study identifies invariants in local search behaviour [MSK97] for various algorithms of the WalkSAT family. Again, the characterisation is based on local search trajectories and no mathematical model is specified. However, it is empirically observed and conjectured that optimal behaviour of WalkSAT algorithms seems to occur for close-to-minimal mean to variance ratio of the objective function as sampled along the search trajectory. This is shown for single problem instances from various domains, including hard Random-3-SAT and Blocks World Planning.

Our approach of functionally approximating the run-time behaviour of SLS algorithms is partly related to the methodology used in [FRV97]. This concerns primarily the use of continuous probability distributions (which can also be found in [Hoo96b, HS96]) for approximating the behaviour of discrete algorithms and the χ^2-test for testing the goodness-of-fit of these approximations. But while they study the search cost distribution of finding a satisfying assignment on an ensemble of instances for *complete* procedures for SAT, we characterise the behaviour of SLS algorithms applied to single problem instances based on run-time distributions. Gomes and Selman give a functional characterisation of the run-time behaviour of randomised complete search procedures for SAT applied to single problem instances from the *Quasigroup Completion Problem* [GSC97]. While they observe heavy-tailed behaviour which can be exploited using rapid random restarts, we did not observe heavy-tailed RLDs for SLS algorithms (using reasonably good parameter settings) when applied to any of our benchmark instances.

To our best knowledge, the approximative completeness of SLS algorithms for SAT has not been theoretically studied before. Thus, the results presented in Section 5.1.1 are novel and original. The same holds for the functional charactisations of SLS behaviour developed in Sections 5.1.2 and 5.1.3. Our main characterisation result, the EPAC behaviour of modern SLS algorithms (using optimal or larger-than optimal noise settings) when applied to hard problem instances from various domains, explains earlier observations regarding the effectiveness of random restart for GWSAT and WalkSAT [GW95, PW96] (*cf.* Section 5.1.5).

The improvements of SLS algorithms for SAT we discuss in Section 5.2 are inspired by previous work: Our study of multiple independent try parallelisation is conceptually related to Hogg's and Williams' work on the potential of parallel processing for *complete* Graph Colouring algorithms [HW94a]. Our discussion of algorithm portfolios is mainly based on ideas presented by Carla Gomes and Bart Selman in the context of complete algorithms for Quasigroup Completion Problems [GS97a]. Finally, the random walk extension of Novelty and R-Novelty is inspired by earlier work on GSAT variants [SKC93] and builds on McAllester's, Selman's, and Kautz's work on the WalkSAT family [MSK97]. Interestingly, combining our essential incompleteness result for WalkSAT+tabu with the ideas of random walk extension, sheds a new light on earlier work which claimed that adding random walk to tabu search algorithms for SAT does not improve SLS performance [SSS97]. Our results for SAT-encoded Graph Colouring instances show that essential incompleteness occurs in practice when using optimal tabu-list length (*cf.* Section 4.5); but the fact that essential incompleteness can generally be overcome by adding random walk (*cf.* Section 5.2) indicates that extending WalkSAT+tabu with random walk will improve the overall behaviour of the algorithm. The fact that in [SSS97] evidence for a contradicting conclusion could be observed is most probably a consequence of the problematic methodology their result is based on (*cf.* Chapter 2, Section 2.3).

5.4 Conclusions

In this chapter we presented three types of results regarding the run-time behaviour of modern SLS algorithms for SAT: theoretical results characterising their asymptotic behaviour, functional characterisations of their actual behaviour, and a number of strategies to improve existing algorithms.

Regarding the asymptotic behaviour of SLS algorithms, we proved a series of novel results, establishing the approximative completeness (PAC property) of GWSAT (for $\mathsf{wp} > 0$) as well as the essential incompleteness of WalkSAT+tabu (for all $\mathsf{tl} > 0$), Novelty, and R-Novelty (for arbitrary wp). Although for the latter algorithms, which are among the most powerful SAT algorithms known today, essential incompleteness does not imply that stagnation of the local search will occur for each problem instance and each run of the algorithm, stagnation behaviour can be observed in practice (*cf.* Chapter 4) and severely affects the otherwise superior performance of these algorithms.

However, essential incompleteness can be easily overcome by extending the given algorithms with random walk, as shown in Section 5.2.1. We introduced this type of extension using the GLSM framework where it is easy to see under which conditions the resulting hybrid GLSM algorithm will be approximately complete. Of course, approximate completeness is also achieved by using a random restart mechanism; however, since in practice good cutoff values for the local search process are not known, and poorly chosen cutoffs lead to extremely poor performance, using random restart alone is substantially inferior to a random walk extension. To demonstrate the practical effectiveness of the random walk extension, we modified R-Novelty and Novelty accordingly and showed empirically that the extended variants achieve superior performance. To our best knowledge, these R-Novelty and Novelty variants are the most powerful currently known SLS algorithms for SAT. However, for certain problems, such as the All-Interval-Series instances, these algorithms, as all other SLS algorithms we are aware of, are substantially outperformed by state-of-the-art systematic SAT algorithms like Satz [LA97] or Satz-Rand [GSK98].

Regarding the functional characterisation of SLS behaviour, we empirically studied the run-time behaviour of WalkSAT, WalkSAT+tabu, Novelty, and R-Novelty. We could show that, using optimal noise parameter settings, the RLDs of these algorithms when applied to hard problem instances from various domains, can be approximated by exponential distributions (EPAC property). The same phenomenon is observed for larger-than-optimal noise settings, while for smaller-than-optimal noise, qualitatively different behaviour occurs. We further introduced a refined mathematical model based on a new distribution type which is asymptotically exponential, but allows to model the effects of the initial search phase. As we have shown, this extended model allows a precise characterisation of SLS behaviour for a vast majority of the problem instances from our benchmark suite. Thus, for the first time, we can model the behaviour of some of the most powerful and prominent SLS algorithms for SAT in great detail.

Our characterisation result has a number of interesting implications. As a direct consequence, random restart is ineffective for the algorithms studied here, when using optimal noise settings. Even worse, when the restart occurs during the initial search phase, *i.e.*, the cutoff occurs too early, the performance will be negatively effected. A second consequence of exponential RTDs is a new interpretation of the local search process as random picking in a drastically reduced, "virtual" search space. The size of this reduced search space can be calculated from the RTD data and, when compared to the actual search space size, gives an indication of the effectivity of the SLS algorithm. This novel interpretation of SLS behaviour raises the question, whether it is possible to identify structural features of the actual search spaces which directly correspond to the reduced search space. The analysis of search space structure is addressed in Chapter 6; however, so far we could not identify any features which are strongly correlated to the virtual search space size.

Finally, regarding the improvement of existing SLS algorithms for SAT, besides the hybrid GLSM extensions mentioned above, we discussed two forms of parallelisation. Multiple independent tries parallelisation, which is equivalent to using homogeneous cooperative GLSM models without communication (*cf.* Chapter 3), is particularly attractive for EPAC algorithms, since in this case optimal speedup and efficiency are obtained. Moreover, since no communication is involved, this model is simple to implement and extremely scalable. The second approach uses portfolios of different SLS algorithms; it can be easily modeled using heterogeneous cooperative GLSMs without communication. This model shows improved robustness and performance over the homogeneous model, when there is no clear domination relation between a set of individual algorithms, *i.e.*, when different algorithms show superior performance in different situations. As we have seen in Chapter 4, this is the case for modern SAT algorithms based on stochastic local search. Therefore, using the portfolio approach appears to be a promising way for improving robustness and performance of complex SAT algorithms.

Both parallelisation approaches are of particular interest in the context of time-critical application scenarios; however, using processor multiplexing, they can generally be also realised on single-processor systems. As a direct consequence of the EPAC behaviour observed for modern SLS algorithms, multiple independent tries will not improve performance on a single-processor system. For algorithm portfolios, the situation is different: based on the observations on our benchmark suite, improvements can be expected even for the single-processor case.

Summing up our results, we have shown how, based on a careful empirical analysis, SLS behaviour can be mathematically modeled in great detail by functionally approximating RTDs. Doing this, we find that the behaviour of some of the most powerful and popular SLS algorithms for SAT exhibit a surprising regular behaviour over a broad range of problem domains. Based on our characterisation of SLS behaviour, we developed and discussed a number of strategies for improving these algorithms. We expect (and have some preliminary evidence as of this writing) that our results on comparing, characterising, and improving SLS behaviour will generalise to SLS algorithms for problem domains like CSP or other hard combinatorial problems.

Chapter 6

Search Space Analysis

Stochastic local search algorithms are among the most powerful methods for solving hard combinatorial problems like SAT or CSP. Thus, there is a considerable interest in improving the algorithms as well as in the understanding of the factors which influence their performance. The behaviour of local search algorithms crucially depends on structural aspects of the underlying search space. Recently, a growing number of researchers have been investigating the nature of this dependency for the propositional satisfiability (SAT) and constraint satisfaction problems (CSP) [CFG+96, Yok97, FCS97]. The main motivation for this kind of work lies in the assumption that knowledge about structural aspects of search spaces and their correlation to the performance of SLS algorithms can ultimately be exploited to improve the algorithms as well as SAT or CSP encodings of problems from other domains. Furthermore, this kind of knowledge can be used for evaluating SLS algorithms and encoding strategies.

In this chapter, we explore several approaches for search space analysis. After giving some motivation and background on the topic, we first analyse the dependence of local search performance on the number of solutions, an issue which has been investigated before [CFG+96]. Our study, however, avoids some weaknesses of this earlier work while extending both the scope and the depth of the analysis significantly, *e.g.*, by studying the observed phenomena for more powerful SLS algorithms and in dependence of problem size. Then, we introduce some novel approaches for search space analysis, based on analysing the trajectories of SLS algorithms. Our results indicate that the most dominant factor for local search performance is given by the number of solutions. However, other factors, like solution clustering, ruggedness of the search space, and local minima branching have a measurable influence on the behaviour of SLS algorithms.

test-set	instances	clause-len	vars	clauses
uf10-49	1,000	4	10	49
uf20-91	1,000	3	20	91
uf50-163	1,000	3	50	163
uf50-273	1,000	3	50	237

Table 6.1: Additional Uniform Random-3-SAT test-sets which are used for search space structure analysis.

6.1 Motivation and Background

Compared to the development of SLS algorithms for SAT, search space analysis is a rather young research subject. The methods we use for analysing search space structure and its impact on SLS behaviour are to some extent based on the approaches taken in previous work [CFG+96, Yok97, FCS97]. However, we find a number of weaknesses occurring in some or all of these studies:

- GSAT, or simplified versions of this algorithm are used as a reference local search procedure. While GSAT has the advantage of conceptual simplicity, its performance is considerably inferior to modern SLS algorithms like GSAT with random walk or WalkSAT, which in particular have the ability to escape from local minima without using a restart mechanism.
- The empirical methodology used to characterise SLS behaviour is often problematic. For example, [CFG+96] use only one maxSteps setting for all experiments, which is claimed to be optimal for the middle of the phase transition. While there is some doubt about the validity of this optimal value itself [HS98a], it is at least not clear, whether this value is also optimal for other clauses/variable ratios. On the other hand it is known that GSAT's performance critically depends on good maxSteps settings [GW95].
- Most of the studies are mainly based on the empirical analysis of Random-3-SAT instances, while other problem classes, in particular SAT-encoded problems from other domains, are somewhat neglected. Furthermore, the dependence of the results on the problem size is usually not investigated. This seems to be particularly problematic, since the importance of taking scaling properties into account is well-known in the field [Fuk97].

In our investigation presented here, we try to avoid these weaknesses. Instead of GSAT, we use GWSAT with approximately optimal noise settings for our experiments. While still being conceptionally very simple, the influence of maxSteps on the performance of this algorithm is minimal (*cf.* Chapter 5, [HS96, HS98a]). For our experiments we used instances and test-sets from the benchmark suite introduced in Chapter 4. Additionally, we generated two test-sets of very small Random-3-SAT instances (from the phase transition region) as well as two test-sets from the under- and over-constrained regions of Uniform Random-3-SAT (*cf.* Table 6.1). For these latter test-sets we chose the number of clauses such that the clauses/variable ratios are 25% below and above the phase transition point. Furthermore, for comparative analyses involving problems from other domains, we used instances from various other Uniform Random-3-SAT test-sets; this will be described in more detail in Section 6.4. All additional test-sets were generated as described in Chapter 4, *i.e.*, by filtering randomly generated Random-3-SAT formulae with a complete SAT algorithm.

If not indicated otherwise, the local search cost reported for individual problem instances was determined by running GWSAT (with approx. optimal noise setting) for at least 100 tries and estimating the expected local search cost per solution as defined in Chapter 2, taking into account the success-rate as well as the length of successful tries. Generally, maxSteps was chosen high enough ($\geq 10^6$) to ensure success probabilities close to one; this way, the error in extrapolating the local search cost is relatively small. As we have shown before, there is a strong correlation between GWSAT's performance and that of other modern SLS algorithms (*cf.* Chapter 4), therefore most of our results regarding local search cost will equally apply to other SLS algorithms for SAT, such as WalkSAT or Novelty.

When analysing search space structure and its impact on SLS performance, one of the most important techniques is the investigation and characterisation of correlations between structural features of the search space and local search cost. For our Random-3-SAT and Graph Colouring test-sets, we characterise these correlations establishing functional models using least-mean-squares fitting and regression analysis techniques known from literature [CFG+96].

6.2 The Number of Solutions

Intuitively, for a given problem instance, its number of solutions should have a considerable effect on local search performance: One would expect that instances with a high solution density, *i.e.*, a large number of solutions, are much easier to solve for SLS algorithms than instances with very few solutions. In previous work [CFG+96], it was shown that this intuition is correct regarding stochastic local search algorithms for SAT and CSP which are based on hill-climbing, such a GSAT. In this section, we first present the results of analysing the distribution of the number of solutions for different Random-3-SAT test-sets. Based on these results, we then discuss the correlation between local search cost and the number of solutions, refining and extending the methodology and results of [CFG+96].

Figure 6.1 (left) shows the empirical cumulative distributions of the number of solutions for the different test-sets at the phase transition for Random-3-SAT [CA96]. The coarse granularity which can be observed at the left end of the curves, especially for $n = 20$, is due to the fact that the number of solutions is a discrete measure and that for small problem sizes many instances have a very small number of solutions. The graphs show clearly

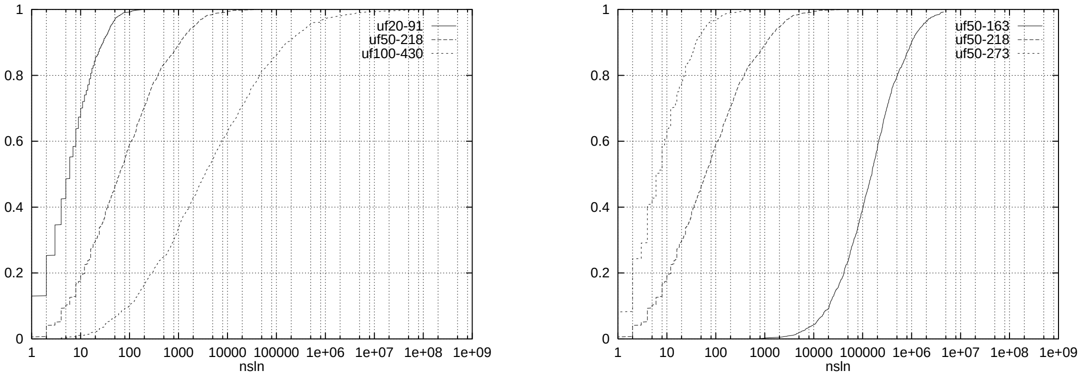


Figure 6.1: Distributions of the number of solutions for Random-3-SAT test-sets with n variables, k clauses (1,000 instances per test-set); *left:* different problem sizes at phase transition; *right:* for $n = 50$ across phase transition.

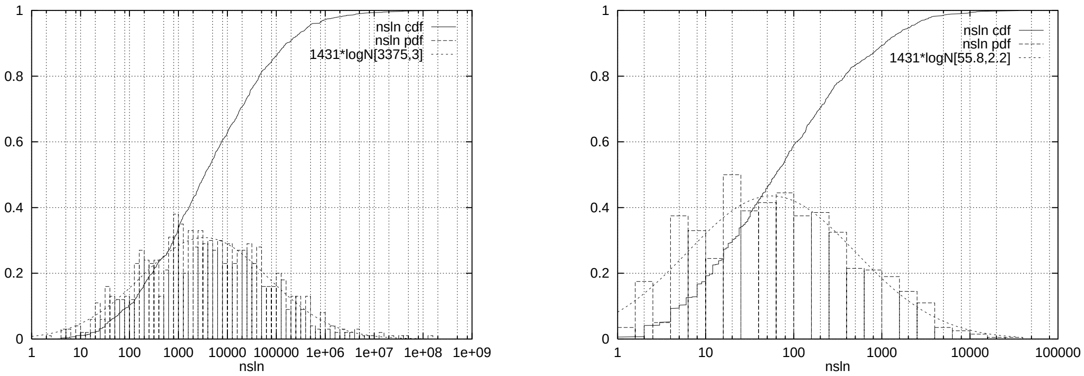


Figure 6.2: Number of solutions distributions for test-sets `uf100-430` *(left)* and `uf50-218` *(right)* at the phase transition (solid curves); the corresponding empirical probability density functions (bar diagrams) can be approximated by log-normal distributions (dashed curves).

that for increasing problem size, the distributions become progressively less steep, *i.e.*, the relative variance in the number of solutions increases significantly for larger problems. Note the difference between this scaling behaviour and the observations for the corresponding hardness distributions, where increasing the problem size affects mainly the tail.

At the first glance, the general shape of the distributions of local search cost and the number of solutions seems to be roughly similar (compare Figure 6.1 and Figure 4.13, page 95). But closer analysis of the empirical probability density functions of the number of solutions distributions indicates that (in contrast to the local search cost distributions) these can be reasonably well approximated using log-normal distributions (Figure 6.2). This is an effect of the random generation process, where each point in the search space (*i.e.*, each assignment) is uniformly affected by each independently generated clause. Note the absence of long and heavy tails, as observed for the corresponding hardness distributions, for which we could not find a functional approximation.

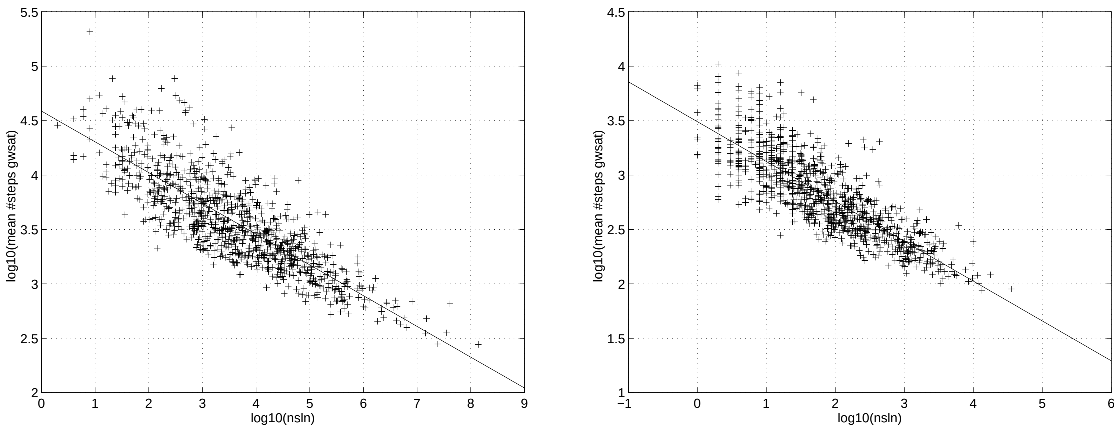


Figure 6.3: Correlation between average local search cost (vertical) and number of solutions (horizontal) for Random-3-SAT test-sets from the phase transition region with $n = 100$ *(left)* and $n = 50$ *(right)* and regression lines from *lms* regression fits.

test-set	a	b	r
uf20-91	-0.494	2.309	-0.82
uf50-218	-0.366	3.491	-0.83
uf100-430	-0.283	4.586	-0.83

test-set	a	b	r
uf50-163	-0.282	3.436	-0.90
uf50-218	-0.366	3.491	-0.83
uf50-273	-0.222	2.827	-0.51

Table 6.2: Data from regression analysis of the correlation between number of solutions and local search cost for Random-3-SAT test-sets; a, b are the parameters of the *lms* linear fit $ax+b$, r is the correlation coefficient; *left:* for different problem sizes at the phase transition, *right:* across the phase transition.

Combining the data from hardness and number of solutions analysis, we can now analyse the correlation between the logarithm of both values. Figures 6.3 depicts this correlation for various problem sizes at the phase transitions. The graphs show scatter plots of the data and the corresponding *lms* regression fits. For all problem sizes, there is a strong negative correlation between the number of solutions and the average local search cost per solution. Also note that while for a high number of solutions, the variability in local search cost is very small, it increases considerably for lower numbers of solutions. Again, the coarse granularity which can be observed at the left side of the graphs for the smaller problem size is caused by the large number of problem instances with few solutions.

As can be seen from Table 6.2 (left), we measure a stronger correlation between the number of solutions and average local search cost than reported in [CFG$^+$96] ($r = -0.83$ *vs* -0.77). At the same time, for $n = 100$ we find a significantly smaller gradient a for the regression fit (-0.283 *vs* -0.44 in [CFG$^+$96]). Remarkably, a decreases strictly monotonically with growing problem size n, meaning that for larger problems the functional dependency of the local search cost from the number of solutions gets weaker. The main reason for this stems from the fact that with growing problem size, the relative variance of the number

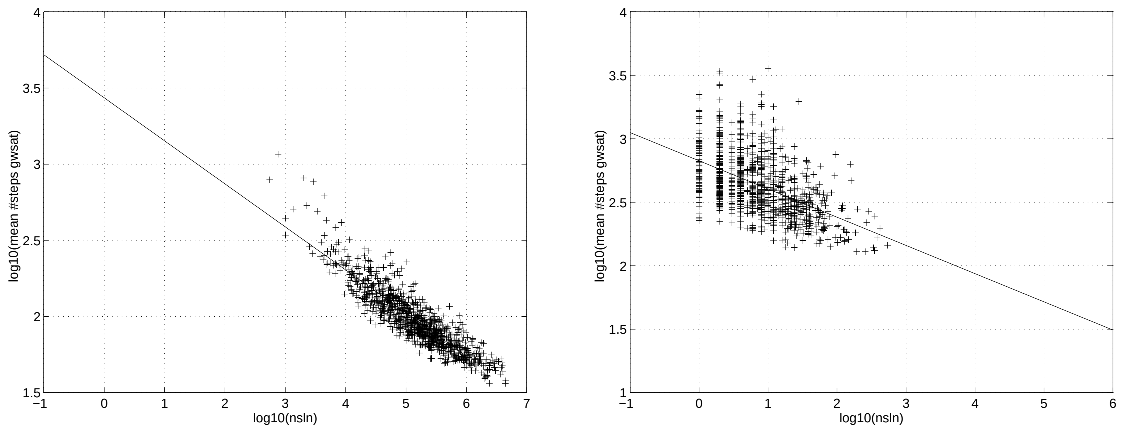


Figure 6.4: Correlation between local search cost (horizonal) and number of solutions (vertical) for Random-3-SAT test-sets with $n = 50$ from the underconstrained region *(left)* and from the overconstrained region *(right)*.

of solution distribution increases significantly faster (and more uniformly across the whole distribution) than that of the corresponding hardness distribution. The dependency between $\log a$ and $\log n$ appears to be roughly linear. More specifically, a very good *lms* fit can be obtained using the parametric function $a = \exp(u \cdot n^v)$, where the optimal parameters are determined as $u = 1.53$ and $v = -0.37$. Not surprisingly, the average local search cost for problem instances with exactly one solution (as estimated by the b parameter of the linear *lms* fit) also increases strictly monotonically with n. Interestingly, using the same fitting approach as above, the dependency between b and n can be approximated by $b = \exp(0.61n^{0.44})$. This confirms earlier observations that the search cost of GSAT-like algorithms might grow slower than a simple exponential [GW93b, GMPW97]. Our results indicate furthermore that the same holds for the extrapolated local search cost for problem instances with exactly one solution.

Applying the same correlation analysis across the phase transition, *i.e.*, for different clauses per variable ratios, reveals some interesting facts. First, the results in Table 6.2 (right) show that the correlation coefficient r is clearly dependent on the clauses per variable ratio (cvr): for increasing cvr, r decreases monotonically. This can be also seen in Fig. 6.4, which shows considerably more noise in the scatter plot for the overconstrained than for the underconstrained test-set. One explanation for this phenomenon could be based on the simple fact that the number of solutions tends to decrease with a growing constrainedness, assuming that for a smaller number of solutions there is more variability in the average local search cost. This hypothesis is not only consistent with the results for $n = 50$ across the phase transition, but also with the fact that within all our test-sets, we found the variability in the number of solutions to be decreasing with growing number of solutions.

Another interesting observation can be made from Table 6.2 (right): The regression gradient a is maximal at the phase transition and significantly lower above and below the critical

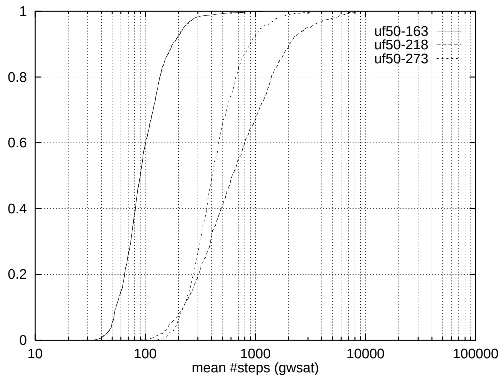


Figure 6.5: Hardness distributions (mean number of local search steps/solution using GWSAT, approx. optimal noise) for Random-3-SAT test-sets across the phase transition.

cvr. Consequently, the number of solutions has the most significant impact on local search cost at the phase transition. This result is a direct contradiction to the result for GSAT (without random walk) reported in [CFG+96], but consistent with their results for various CSP algorithms on Random-CSP problems. The most probable reason for this disagreement is given by the fact that they are using basic GSAT in combination with a rather crude methodology for evaluating the algorithm's behaviour. Note, that the extreme behaviour at the phase transition is also reflected in the hardness and number of solutions distributions (*cf.* Figures 6.5 and 6.1, right). We will come back to this issue in Section 6.3.

One last observation on the data presented in Table 6.2 (right) is the fact that the b parameters of the *lms* linear fits are roughly similar for test-sets `uf50-163` and `uf50-218`, while for `uf50-273` it is considerably lower. Since the b parameter gives an estimate for the average search cost on problem instances with exactly one solution, this observation implies that single solution instances tend to be harder when taken from the underconstrained region than when being overconstrained. One possible explanation for this phenomenon is given by the observation that underconstrained problems tend to have larger plateau regions [Yok97] which are probably more difficult to search for SLS algorithms like GWSAT or WalkSAT [FCS97].

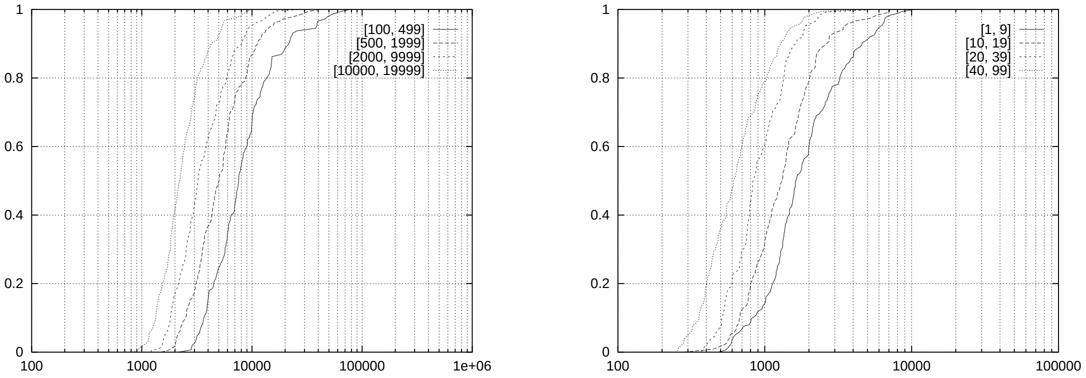


Figure 6.6: Hardness distributions for different strata $[l_i, u_i]$ of Random-3-SAT test-sets with $n = 100$ *(left)* and $n = 50$ *(right)* at the phase transition.

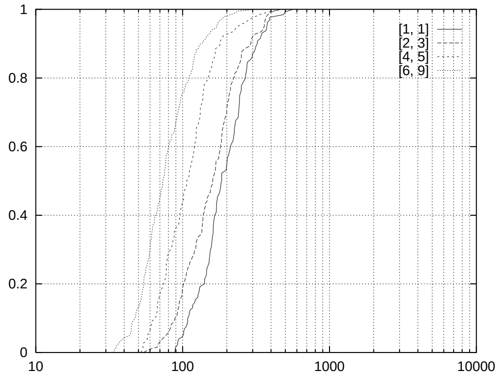


Figure 6.7: Same as Figure 6.6, but for $n = 20$.

6.3 Stratified Random-3-SAT

Although our results show a stronger correlation between the number of solutions and local search cost than reported in [CFG+96], we still observe a considerable variability in the local search cost for instances from the same test-set with a similar or equal number of solutions. To abstract from the predominant influence of the number of solutions on problem hardness, we study this phenomenon in more detail by stratifying our test-sets (defined by purely syntactic properties of the formulae) according to the number of solutions (a semantic property). More precisely, we measure hardness distributions (using the same method and parameters as in Chapter 4, Section 4.5), for the strata s_i obtained from the test-sets by restricting the number of solutions to an interval $[l_i, u_i]$.

For various problem sizes at phase transition, the results of this analysis are shown in Fig. 6.6 and 6.7. Surprisingly, in a semi-log plot, the shapes of these distributions are very similar for the different strata. For larger numbers of solutions, the corresponding curves are shifted to the left and they become slightly steeper, indicating that problems with many

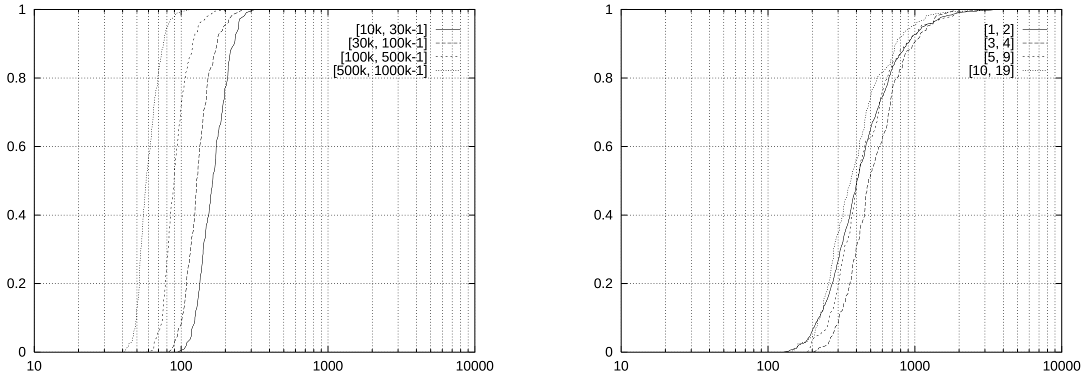


Figure 6.8: Hardness distributions for different strata of Random-3-SAT test-sets with $n = 50$ from the underconstrained region *(left)* and from the overconstrained region *(right)*.

solutions are uniformly easier and exhibit a smaller relative variance of instance hardness. Note also that the overall shape of the distributions for the strata is very similar to that of the overall hardness distributions (*cf.* Figure 6.5). However, the variance of the overall hardness distribution is significantly higher than for the strata; closer examination also reveals slightly heavier tails for the former.

Applying the same analysis across the phase transition, we observe an interesting difference (*cf.* Figure 6.8). While for the overall hardness distribution, minimal steepness (*i.e.*, maximal relative variance) can be observed at the phase transition (*cf.* Figure 6.5), the steepness of the corresponding distributions for the strata appears to be monotonically decreasing with growing constrainedness (*cf.* Figures 6.6–6.8). This means that across the phase transition, the shape of the overall distribution depends not only on the shape of the distributions for the strata, but also on the distribution of the number of solutions. In particular, the fact that the regression gradient for the correlation between local search cost and number of solutions is minimal at the phase transition (*cf.* Section 6.2) appears to be caused by the shape of the number of solutions distributions (*cf.* Figure 6.1, right).

Generally, the results reported in this section show that the noise observed in the correlation between local search cost and the number of solutions is mainly caused by some factor which is not related to the number of solutions. In particular, clustering of the solutions is most likely not the feature responsible for the variation in hardness within the strata: If clustering, *i.e.*, the distribution of the solutions within the search space, would have a major impact on local search cost, we should observe an increased solution cost variability for problems with higher numbers of solutions (where more pronounced differences in clustering can occur). But there is no significant difference between the hardness distributions on problems with only one solution (where clustering is irrelevant) and a higher number of solutions.

6.4 Sampling the Objective Function

Up to this point, our analysis was mainly concerned with the number of solutions and their distribution in the search space. But of course, other aspects of the search space topology play an important role in the context of SLS behaviour. One such factor is the ruggedness or smoothness of the objective function across the search space. It is intuitively clear that a mainly flat objective function which is only locally punctuated by the solutions should be much harder to deal with for any local search algorithm than one in which the solutions are surrounded by sloping basins. Unfortunately, the topological property corresponding to this intuitive notion of ruggedness is very difficult to measure directly.

We therefore use a simple indirect method for analysing the influence of objective function topology on SLS performance. This method essentially correlates the variance of the objective function, which is an indication of the ruggedness of the search space, to the average solution cost. For small problem instances ($n \leq 30$), we can use exhaustive search to determine the exact variance. For larger n, however, this becomes impractical; consequently, we use random sampling to estimate the objective function's variance. The intuition underlying this method is that flat, feature-less search spaces characterised by a low variance of the objective function should pose greater difficulties for SLS algorithms than rugged search spaces with a high objective function variance.

Random-3-SAT We performed this analysis for the Random-3-SAT test-sets for variable problem size and constrainedness. Because of computation time limitations, we reduced each test-set to 50 instances by selecting each 20th instance from the original test-set after ordering it according to the average local search cost for GWSAT (approx. optimal noise). We then measured the standard deviation of the objective function (number of unsatisfied clauses) for each remaining instance; for $n = 20$, the measurement is based on a systematic sample across the full search space (obtained by exhaustive search), for $n = 50$ and 100, it is based on a sample of 10^5 randomly picked assignments.

Preliminary results on single instances of the `uf50-218` test-set suggested a negative correlation between the standard deviation of the number of unsatisfied clauses (*sdnclu*) and the average local search cost. The result of a correlation analysis across the whole test-set confirms this hypothesis. Figure 6.9 shows the correlation between *sdnclu* and the average local search cost (for GWSAT). The negative correlation, although quite noisy, is clearly observable (*cf.* Table 6.3 for correlation coefficient and parameters of the linear *lms* fit).

This result is consistent with our initial intuition that a smaller standard deviation indicates a more uniform search space structure, possibly providing less guidance for local search. Another explanation would suggest that for hard Random-3-SAT instances, a smaller variance of the objective function coincides with a lower number of solutions. This is consistent with our observation that the values of the objective function are approximately normally distributed:[1] in this situation, a smaller variance naturally implies smaller tail probabilities

[1] This is not very surprising, since for a given Random-3-SAT instance, clauses are generated randomly

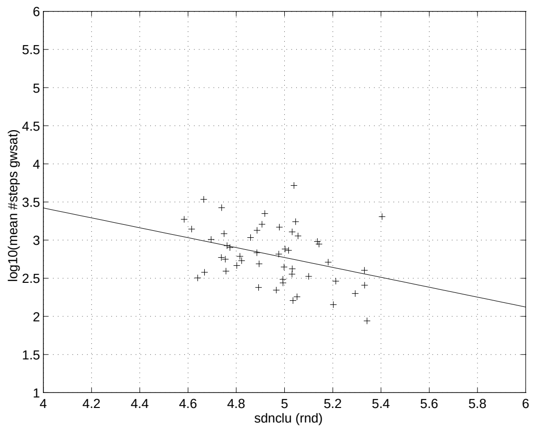


Figure 6.9: Correlation between *sdnclu* values (horizontal) and mean local search cost (vertical) for reduced test-set `uf50-218 (50)`.

test-set	r	a	b
uf20-91 (50)	-0.31	-0.42	3.28
uf50-218 (50)	-0.35	-0.65	6.02
uf100-430 (50)	-0.44	-0.97	10.31

test-set	r	a	b
uf50-163 (50)	-0.52	-0.56	4.37
uf50-218 (50)	-0.35	-0.65	6.02
uf50-273 (50)	-0.02	-0.02	2.75

Table 6.3: Correlation between *sdnclu* and average local search cost; a, b are the parameters of the linear fit $ax + b$, r is the correlation coefficient; *left:* for different problem sizes at phase transition, *right:* across phase transition

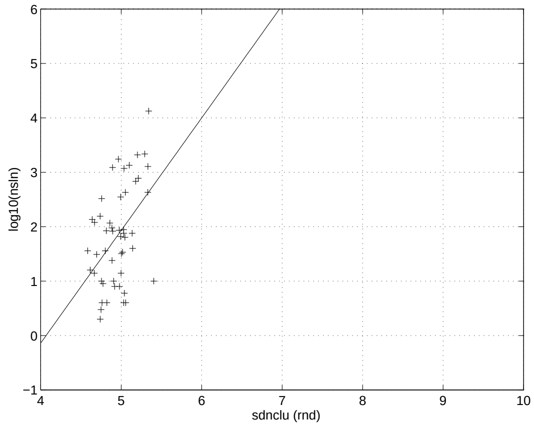


Figure 6.10: Correlation between *sdnclu* values (vertical) and number of solutions (horizontal) for reduced test-set `uf50-218 (50)`.

of this distribution and, consequently, a smaller number of solutions.

While both explanations could be correct, it is a priori not clear, whether one or the other plays a more important role in the given context. To further investigate this issue, we analysed the correlation between the *sdnclu* value and the number of solutions across the same test-set. The results of this analysis is shown in Figure 6.10: a positive correlation can be clearly observed (correlation coefficient $r = 0.46$) and a *lms* linear fit of the correlation data gives a gradient of 2.07. Note that this correlation is slightly stronger than the one between *sdnclu* and the average solution cost reported before. This suggests the following interpretation: For hard Random-3-SAT instances, a uniform search space structure, besides other factors, tends to come along with a small number of solutions, which itself has a major impact on the local search cost for finding a solution. Compared to this effect, the more uniform search space structure seems to play a minor role. This was verified by analysing the correlation between *sdnclu* and average solution cost for strata of the unreduced test-set `uf50-218` with equal or similar numbers of solutions. Here, a negative correlation between *sdnclu* and the average solution cost could not be observed. Thus, the *sdnclu* / solution cost correlation can apparently not be used to refine the analysis from Section 6.2.

Next, we studied the observed correlation for various problem sizes at the phase transition as well as across the phase transition. The results are shown in Figures 6.11–6.12 and in Table 6.3. While for the underconstrained test-set, the correlation between *sdnclu* and average solution cost is quite strong, it is getting weaker as the number of clauses increases. For the overconstrained test-set, no significant correlation could be observed. Note that this

---

and the influence of each clause on a given assignment is mostly independent of other clauses. Note, however, that by restricting the test-sets to satisfiable formulae, strict probablistic independence between the impact of clauses on the search space is lost.

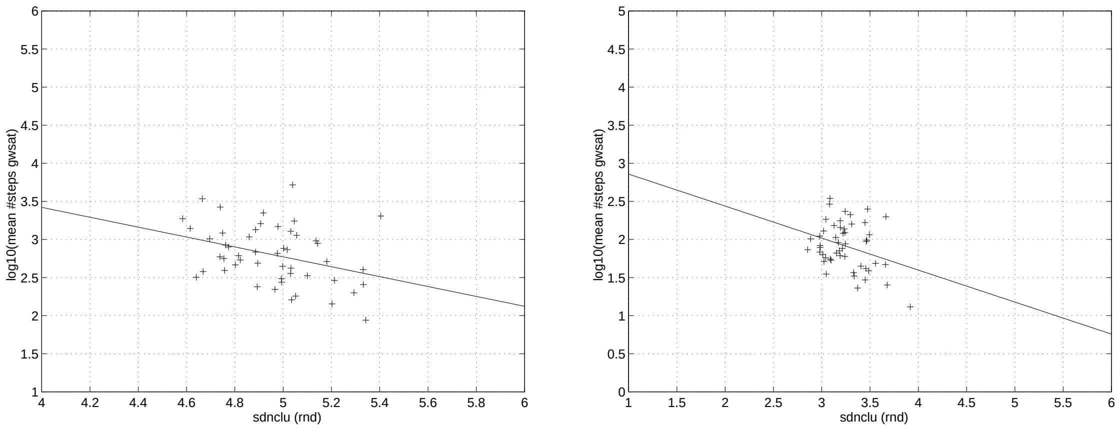


Figure 6.11: Correlation between *sdnclu* values (horizontal) and mean local search cost (vertical) for reduced test-sets `uf100-430 (50)` (*left*) and `uf20-91 (50)` (*right*).

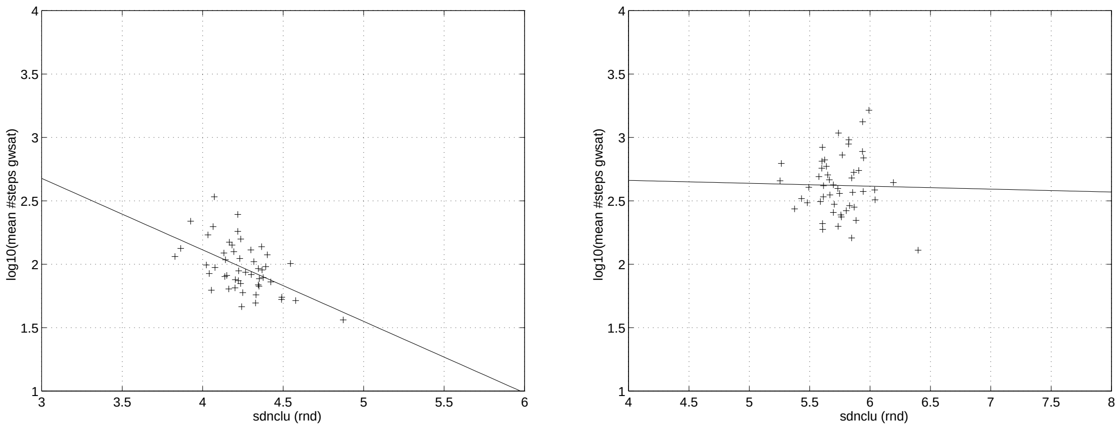


Figure 6.12: Correlation between *sdnclu* values (horizontal) and mean local search cost (vertical) for reduced test-sets `uf50-163 (50)` (*left*) and `uf50-273 (50)` (*right*).

instance	variables	clauses	solutions	avg *lsc*	*sdnclu*
flat30-60/n19	90	300	432	1,775.8	205.84
uf90-388/med	90	388	404	2,492.1	45.00
flat50-115/n4	150	545	5,892	4,539.2	421.20
uf150-645/med	150	645	5,884	10,231.9	70.05

Table 6.4: Comparison of the average local search cost and *sdnclu* values for Graph Colouring and Random-3-SAT instances.

result is analogous to our observations concerning the correlation between the number of solutions and average solution cost (see Section 6.2). Therefore, this result is consistent with the explanation for the *sdnclu* / solution cost correlation proposed above. The same holds with respect to varying problem size; here, we observe that the correlation gets stronger with increasing problem size. This is particularly interesting as, despite the huge differences in search space size between $n = 20$ and $n = 100$, our observations were obtained with a fixed sample size per instance. This is an indication that the observed correlations can be established for quite small relative sample sizes. At the same time, increasing the sample size for small problems does not significantly improve the observed correlation.

Graph Colouring Next, we applied the same analysis to problem instances from the Graph Colouring domain. Because of computational limitations we had to restrict the investigation to individual instances for different problem sizes. Our analysis is based on comparing problem instances with identical search space size (number of variables) and, to abstract from the dominant influence of this feature, a closely matching number of solutions. To achieve this, we newly generated two hard Random-3-SAT test-sets with 90 variables / 338 clauses and 150 variables / 645 clauses, consisting of 100 satisfiable instances (unforced, filtered generation) each. From these test-sets, we picked the median instances w.r.t. average local search cost for GWSAT (approx. optimal noise) and compared them with the instances from the `flat30-60` and `flat50-115` Graph Colouring test-sets which had the closest matching number of solutions, resp.

For these problem pairs, we compared the average local search cost for GWSAT (approx. optimal noise) and the *sdnclu* value. The results of this analysis are reported in Table 6.4; avg *lsc* refers to the average number of steps per solution when using GWSAT with approximately optimal noise settings. First, we observe that the more structured Graph Colouring instances tend to be easier to solve than corresponding Random-3-SAT instances with identical search space size and similar number of solutions. This effect apparently becomes more prominent with increasing problem size. Furthermore, the *sdnclu* values for the Graph Colouring instances are significantly higher than those for the Random-3-SAT instances. This confirms our initial intuition that the standard deviation of the objective function values should be negatively correlated with the hardness of an instance. Apparently, this effect, which we could not clearly detect within hard Random-3-SAT test-sets, can be observed when comparing structured and random instances.

instance	variables	clauses	solutions	avg lsc	$sdnclu$
bwp/anomaly	48	261	1	944.2	166.57
uf48-210/n49	48	210	1	1,509.5	24.84
bwp/medium	116	953	2	1,787.9	1,157.83
uf116-499/n92	116	499	16	12,484.5	57.31

Table 6.5: Comparison of the average local search cost and $sdnclu$ values for Blocks World Planning and Random-3-SAT instances.

instance	variables	clauses	solutions	avg lsc	$sdnclu$
ais6	61	581	24	2,766.42	84.12
uf61-263/n75	61	263	24	1,695.37	28.98
ais8	113	1,520	40	64,167.11	3,711.34
uf113-486/n17:	113	486	40	58,267.11	51.74

Table 6.6: Comparison of the average local search cost and $sdnclu$ values for All-Interval-Series and Random-3-SAT instances.

Blocks World Planning Applying the same methodology to small instances from the Blocks World Planning domain essentially confirms the observations from the previous section. The results are reported in Table 6.5; just as for the graph colouring domain, the Blocks World Planning instances are significantly more difficult for SLS algorithms than Random-3-SAT instances with identical search space size and a similar number of solutions. Again, with growing problem size, the difference in local search cost becomes more pronounced. At the same time, when compared with the corresponding Random-3-SAT instances, the Blocks World Planning instances show significantly higher average $sdnclu$ values which also appear to grow faster with increasing problem size.

Note that because the Blocks World Planning instances have only one resp. two solutions, when taking into account the correlation between the number of solutions and average local search cost for Random-3-SAT, the corresponding Random-3-SAT instances are amongst the hardest of the respective test-sets. Therefore, we could not extend our analysis to the larger Blocks World Planning instances, because the corresponding Random-3-SAT problems are beyond the reach of even the most powerful SLS algorithms, given our computational resources and the type of analysis performed.

All-Interval-Series Performing the same analysis for the two smaller All-Interval-Series instances shows that for those, also much higher $sdnclu$ values than for corresponding Random-3-SAT instances can be observed (*cf.* Table 6.6). However, the All-Interval-Series instances are more difficult to solve. This is not consistent with our observations for the instances from the Graph Colouring and Blocks World Planning domains and indicates that there are other factors which render the All-Interval-Series instances so hard for stochastic local search.

In summary, in this section we found some evidence that the standard deviation of the objective function (*sdnclu*), which intuitively corresponds to the ruggedness of the search space, is much lower for Random-3-SAT than for any of the more structured problem instances. The *sdnclu* value might also be correlated to the average local search cost for GWSAT. However, this correlation is somewhat noisy and does not account for the differences in hardness between Random-3-SAT instances with an identical or similar number of solutions or the relative hardness of All-Interval-Series instances when compared to instances from other structured problem domains. On the other hand, for Graph Colouring and Blocks World Planning instances, which are significantly easier to solve than Random-3-SAT instances with identical search space size and an almost identical number of solutions, we consistently observed significantly higher *sdnclu* values.

6.5 State Types and their Distribution

To further refine the analysis, in this section we classify search space states (or positions) according to the topology of their local neighbourhood and analyse the distributions of these state types. A similar approach has been followed in [Yok97]; it is based on the obvious fact that SLS algorithms are mainly guided by the local neighbourhood of the search space states along their trajectories. We classify the *state types* in the following way:

Definition 6.1 (Search Space State Types)

Let S be a search space, $N \subseteq S \times S$ a neighbourhood relation, and $f : S \mapsto \mathbb{N}$ an objective function on S. For a search space state $s \in S$, we define the following functions which determine the number of upwards, sidewards, and downward steps from s:

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

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

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

□

The intuition behind these state types is that SLMIN and SLMAX states are strict local minima and strict local maxima, resp., LMIN and LMAX are (non-strict) local minima and local maxima which may occur on plateau regions; IPLAT states occur in the interior of plateaus, BPLAT characterises plateau border states, and SLOPE states correspond to sloping regions of the search space, where any direct step either increases or decreases the value of the objective function.

Obviously, for any given search space S, neighbourhood relation N, and objective function f, the classes of search space states induced by these predicates form a complete partition of S, *i.e.*, every search space state falls into exactly one of these types. Note also that these types can be weakly ordered according to the restrictiveness of their defining predicates when assuming that defining equalities are more restrictive than inequalities; in this respect, SLMIN, SLMAX, and IPLAT are most restricted, followed by LMIN, LMAX, and SLOPE, while BPLAT is least restricted. This ordering can be further refined by additionally assuming that conditions on the number of sideward steps are more restrictive than those on the number upwards or downward steps; the underlying intuition is that in the latter case are many options (i.e., objective function values), while for sideward steps, the objective function value is fixed. Thus, type SLOPE is more constrained than LMIN and LMAX, and IPLAT is more restrictive than SLMIN and SLMAX. For random search space structures, we would therefore expect a distribution of the state types according to this ordering, *i.e.*, the more constrained a state type is, the more seldomly it should occur.

However, it is not clear whether and to what extent the search spaces induced by the SAT instances from our benchmark suite, assuming the standard GSAT neighbourhood relation and objective function, exhibit random structure. Even for the Random-3-SAT instance distributions considered here, due to the restricted number of clauses (compared to the search space size, the number of clauses is logarithmic), the search space structure cannot expected to be truly random.[2] Therefore, to get a clearer picture of how local features of the search space topology affect SLS behaviour, we take an empirical approach to investigating state type distributions.

Before we report the results of our empirical analysis of state type distributions, we define the important notion of a *plateau region.*

Definition 6.2 (Plateau Region)

Let S be a search space, $N : S \mapsto S$ a neighbourhood relation, and $f : S \mapsto \mathbb{N}$ an objective function on S. A set $S' \subseteq S$ of search space states is a region, if it is connected by N, *i.e.*, for each pair of states $s', s'' \in S'$, a connecting path $s' = s_0, s_1, \ldots, s_k = s''$ exists, for which all s_i are in S' and all s_i, s_{i+1} are direct neighbours w.r.t. N. The *border* of a region S' is defined as the set of states within S' which have at least one neighbour state which does not belong to S'.

[2] One potential way for generating truly random search space structure is by means of random canonic normal form formulae; this approach would, however, require a number of clauses which is exponential larger than the number of variables.

instance	avg *lsc*	SLMIN	LMIN	BPLAT	IPLAT	SLOPE	LMAX	SLMAX
uf10-49/easy	5.48	0.10%	1.46%	87.21%	0%	10.35%	0.78%	0.10%
uf10-49/medium	24.59	0.10%	3.42%	83.50%	0%	11.82%	0.90%	0.20%
uf10-49/hard	244.15	0.20%	3.22%	86.52%	0%	7.51%	2.44%	0.10%
uf20-91/easy	13.05	0%	0.11%	99.27%	0%	0.59%	0.04%	$< 0.01\%$
uf20-91/medium	83.25	$< 0.01\%$	0.13%	99.40%	0%	0.31%	0.06%	$< 0.01\%$
uf20-91/hard	563.94	$< 0.01\%$	0.16%	99.23%	0%	0.56%	0.05%	$< 0.01\%$

Table 6.7: Distribution of search space state types for instances from Random-3-SAT test-sets `uf10-49` and `uf20-91`, based on complete exhaustive sampling.

Now, a plateau region is defined as a region S' for which

(i) all states $s' \in S'$ have the same objective function value, *i.e.*, $\exists l \in \mathbb{N} : \forall s' \in S' : f(s') = l$, and

(ii) no state s' from the border of S' has a neighbour state with the same level as s', *i.e.*, $\neg\exists s' \in S', s'' \in S - S' : N(s', s'') \wedge f(s') = f(s'')$. □

Obviously, the combined size of all plateau regions in S' is given by the sum of the number of IPLAT, BPLAT, LMIN, and LMAX states. In the following, we will use the term local minima region to refer to plateau regions containing LMIN states. Note that this definition covers the notions of both, local minima and benches, as defined in [FCS97]. As we will argue later (*cf.* Section 6.6), this approach makes sense in the light of our findings regarding the topology of local minima regions, considering that we are mainly concerned with SLS algorithms which can effectively escape local minima, such as GWSAT.

Random-3-SAT In a first experiment, we determined the complete state type distribution for the easy, medium, and hard instance of test-set `uf20-91` (the search space of these instances is small enough for exhaustive sampling). Table 6.7 shows the results of this analysis; *avg lsc* refers to the mean local search cost for GWSAT (measured in steps per solution, using approximately optimal noise, 100 tries per instance); the percentages for the state types specify the fraction of the sample matching this state type. Entries reading $< 0.01\%$ and $> 99.99\%$ indicate that the corresponding values are in the open intervals $(0\%, 0.01\%)$ and $(99.99\%, 100\%)$, respectively. The results are consistent with the ordering based on the restrictiveness of the state types discussed above: BPLAT states are predominant, followed by SLOPE, LMIN, and LMAX states; SLMIN and SLMAX states occur very rarely, and no IPLAT states were found for any of the instances analysed here or later. First of all, this suggests that the search spaces of Random-3-SAT instances show indeed structural properties (regarding state type distributions) similar to enirely random search spaces. But while random search spaces can be expected to contain equal numbers of LMIN and LMAX *resp.* SLMIN and SLMAX states, this is apparently not the case for Random-3-SAT search spaces. Here, LMIN states occur more more frequently than LMAX

instance	avg lsc	SLMIN	LMIN	BPLAT	IPLAT	SLOPE	LMAX	SLMAX
uf50-218/medium	615.25	0%	47.29%	52.71%	0%	$< 0.01\%$	0%	0%
uf100-430/medium	3,410.45	0%	43.89%	56.11%	0%	0%	0%	0%
uf150-645/medium	10,231.89	0%	41.95%	58.05%	0%	0%	0%	0%

Table 6.8: Distribution of search space state types for medium hardness problem instance from various Random-3-SAT test-sets, based on sampling along GWSAT trajectories (wp = 0.5).

states. This is most probably an effect of the CNF generation mechanism for Random-3-SAT instances; since each added three-literal CNF clause "lifts" the objective function for one eighth of the search space states by one, while the remaining states remain unaffected, local maxima are more likely to be eliminated when more and more clauses are added.

Our results also suggest that for Random-3-SAT at the phase transition, instance hardness might be correlated to the number of LMIN states, which conforms to the intuition that local minima states impede local search (*cf.* [Yok97]). But more interestingly, since we do not observe any IPLAT states, for algorithms using random walk (like GWSAT), there is always a direct escape route from any local minimum state. Thus, the common analogy relating local minima regions to basins is apparently not correct for Random-3-SAT instances from the phase transition, as each local minimum state has at least one neighbour which is not part of its plateau region.

We do not include analogous results for larger Random-3-SAT instances, since for these, exhaustive sampling is impractical (due to their huge search spaces) and random sampling finds almost exclusively BPLAT (and, very seldomly, SLOPE) states. Instead, in the following, we turn to sampling the state distributions from the trajectory of GWSAT using a noise setting of 0.5.[3] For obtaining these samples, we used 100 tries of the algorithm with a maximum of 1,000 steps each. If within one of these short tries a solution was found, this try was aborted as usual, to prevent solution states from being over-represented in the sample.[4] Therefore, the actual sample sizes vary; however, we made sure that each sample contained at least 50,000 search space states.

The results of this analysis, reported in Table 6.8, show clearly that GWSAT is strongly attracted to local minima states (as is to be expected, given its close relation to gradient descent). They also suggest that neither SLMIN, nor IPLAT or SLOPE states play a significant role in SLS behaviour on Random-3-SAT instances. This is consistent with our earlier observation for small instances, that these state types are extremely rare or do not occur at all.

[3] We chose this noise setting, because it is close to optimal for almost all the domains from our benchmark suite, and we prefer to use identical parameter settings for all problem domains to enhance comparability of our results across problem domains.

[4] GWSAT will, once it has found a solution, return to that solution over and over again with a very high probability, unless the search is restarted.

instance	avg lsc	SLMIN	LMIN	BPLAT	SLOPE
flat30/n19	1,775.8	0.44%	72.86%	26.58%	0.12%
uf90-388/medium	2,492.1	0%	49.41%	50.59%	0%
flat50/n4	4,539.2	1.30%	61.13%	37.10%	0.46%
uf150-645/medium	10,231.9	0%	41.95%	58.05%	0%
bwp/anomaly	944.2	18.68%	16.01%	54.94%	10.37%
uf48-210/n49	1,509.5	0%	46.01%	53.97%	0.02%
bwp/medium	1,787.9	6.56%	33.40%	56.31%	3.73%
uf116-499/n92	12,484.5	0%	44.77%	55.23%	0%
ais6	2,766.42	8.35%	40.78%	44.33%	6.55%
uf61-263/n75	1,695.37	0.03%	50.51%	49.46%	0%
ais8	64,147.11	4.73%	41.31%	48.35%	5.61%
uf113-486/n17	58,267.11	0%	42.14%	57.86%	0%

Table 6.9: Distribution of search space state types for instances from the Random-3-SAT (`uf-*`), Graph Colouring (`flat-*`), Blocks World Planning (`bwp-*`), and All-Interval-Series (`ais-*`) domains; based on sampling along GWSAT trajectories (`wp` = 0.5).

Graph Colouring, Blocks World Planning, and All-Interval-Series Applying an analogous analysis to the other problem domains from our benchmark suite, we find essentially the same situation (*cf.* Table 6.9). To abstract from the predominant influence of the number of solutions on local search behaviour, we used the same pairs of instances as in Section 6.4. IPLAT, MAX, and SLMAX states were not observed for any of the instances and are therefore not listed in the table. The results show several differences between the problem domains. First, while for Random-3-SAT, SLMIN states are almost never encountered, they regularly occur for the other problem domains. The same holds for SLOPE states. For the Blocks World Planning and All-Interval-Series instances, both observations can be intuitively explained by the fact that these have a significantly higher clauses per variable ratio than Random-3-SAT instances from the phase transition (*cf.* Chapter 4). Adding clauses tends to break up plateau regions [Yok97], which eventually converts some LMIN states into SLMIN states. Using a similar argument, it is easy to see that at the same time, BPLAT states are converted into SLOPE states. However, this explanation does not account for the corresponding observation regarding Graph Colouring instances, as these have a slightly lower clauses per variable ratio than Random-3-SAT at phase transition. As the All-Interval-Series instances are harder than corresponding Random-3-SAT instances, while Graph Colouring and Blocks World Planning instances are significantly easier, our results give no indication that the more frequent occurrence of SLMIN or SLOPE states for the SAT-encoded instances has any disadvantages, nor advantages w.r.t. SLS performance.

Second, the ratio between BPLAT and LMIN states is different for most of the domains: While for the Graph Colouring instances, more LMIN states are encountered, Blocks World Planning instances show more BPLAT states. For Random-3-SAT as well as All-Interval-Series, the ratio is more even, with a slight dominance on the side of the BPLAT states. However, since both, the Graph Colouring and the Blocks World Planning instances are significantly easier for GWSAT than the corresponding Random-3-SAT instances, this does not suggest any correlation between the hardness of a problem instance and the BPLAT to

LMIN ratio along GWSAT trajectories. Intuitively, this means that neither LMIN states nor BPLAT states induce significantly more difficulties for SLS algorithms.

Summarising these results, we found that IPLAT states do not occur for any of the instances analysed here, regardless of whether the search space is randomly sampled, or whether the samples are based on GWSAT trajectories. Furthermore, SLOPE as well as SLMIN states occur very seldomly for Random-3-SAT instances and only occasionally for SAT-encoded instances from any of the other domains when sampling GWSAT trajectories. Instead, the state type distributions thus obtained are dominated by BPLAT states, which mainly represent the gradient descent and plateau escape behaviour of GWSAT, and LMIN states, which characterise the plateau search phases of GWSAT's search. Somewhat surprisingly, our results regarding the ratio of BPLAT and LMIN states along GWSAT's trajectories when applied to instances from various problem domains, do not give clear evidence for the plateau search phase having a significantly larger or smaller impact on SLS performance than the gradient descent and plateau escape phases.

Thus, analysing state type distributions did not provide a simple feature that could account for the observed differences in SLS performance between various problem instances and domains. Consequently, we have to consider other, less localised features in our attempt to gain a better understanding of SLS behaviour. To this end, the fact that BLMIN states are never observed, provides an interesting starting point.

6.6 Local Minima Branching

Obviously, local minima regions play an important role for local search and it is a well-known fact that the number of local minima states and regions has a major impact on SLS performance [Yok97, FCS97]. But of course, not only the number of local minima states in the search space matters, but also their distribution. In earlier work, the metaphor of plateaus consisting of local minima was often used in the context of characterising SLS behaviour or search space structure. The number and size of these plateaus, as well as the number of their exits have been studied for Random-3-SAT [FCS97], while their geometry has not been addressed. This is somewhat surprising, since it is intuitively clear that, under the standard assumption that SLS algorithms spend most of their search effort in plateau regions, notions like the diameter, or the length of escape routes from a given plateau, should be extremely relevant. The topology of local minima regions is particularly interesting given the proven effectiveness of random walk techniques in the context of modern SLS algorithms, as, assuming a plateau-like, compact structure, random walk steps are only effective for escaping from a plateau if they are executed at its border. From Section 6.5 we know that interior plateau states are non-existent, or extremely rare, both for Random-3-SAT and SAT-encoded problems. Therefore, *every* state in a plateau is a border state — which partly explains the effectiveness of simple escape strategies.

In the following, we study the geometry of local minima regions for classes of random and

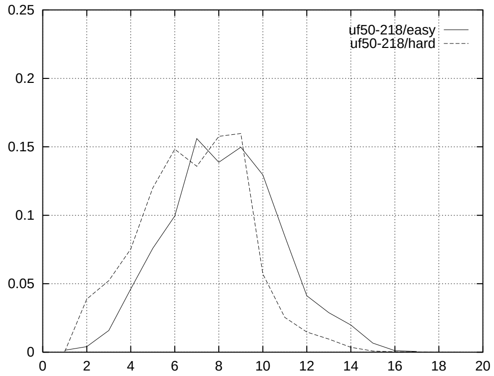


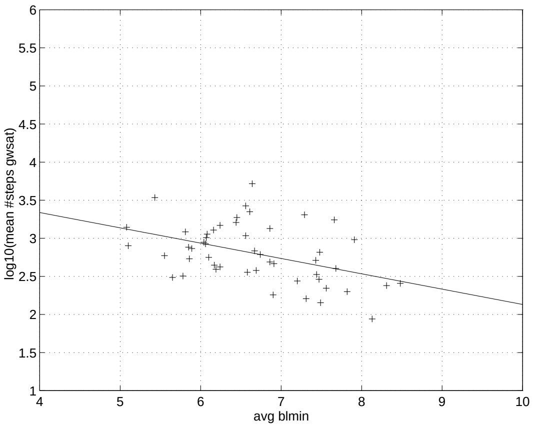
Figure 6.13: Empirical *blmin* probability distributions for easy and hard Random-3-SAT instances from `uf50-218` test-set.

structured SAT problems in more detail. Our analysis is based on measuring the branching of local minima states, where the *branching factor of a local minimum state* (*blmin*) is defined as the number of its direct neighbours with identical objective function value.[5] Thus, *blmin* is a positive integer between 0 and n (the number of variables for the given instance). It is quite obvious to assume that the branching, *i.e.*, the number of direct escapes from a given plateau state, should have an important impact on the effectiveness of escape mechanisms like random walk. But measuring *blmin* is not as simple as one might think; given the low relative proportion of local minima states when compared to the total size of the search space (*cf.* Section 6.5), random sampling is not an option. Therefore, we sample along trajectories of GWSAT (for approx. optimal noise); this has the additional effect of concentrating the analysis to the regions of the search space which are more relevant for SLS behaviour. If not indicated otherwise, our analyses are based on a sample obtained from 100 GWSAT tries with a maximal number of 1,000 steps per try. Thus, our samples usually contain 100,000 search space states; since in some of the tries, GWSAT finds a solution, the actual size of the samples might be slightly lower in some instances.[6]

Random-3-SAT To study the dependence of SLS performance on the branching of local minima, we investigate the correlation between the average solution cost for GWSAT and the average *blmin* values obtained by sampling GWSAT trajectories (as described above). First, we study some *blmin* distributions; later on, as in previous sections, we do a correlation analysis, computing correlation coefficients and performing linear regression. To keep the computation times within acceptable limits, we use the same reduced Random-3-SAT test-sets as in Section 6.4.

[5] Note that, for the SLS algorithms considered here, the number of direct neighbours is equal to the number of variables.

[6] To limit this effect, we had to keep the tries rather short.



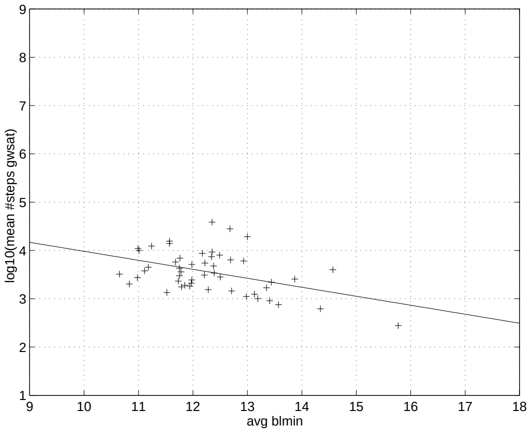


Figure 6.14: Correlation between average *blmin* and average local search cost for GWSAT (approx. optimal noise) for test-sets `uf50-218` *(left)* and `uf100-430` *(right)* (Random-3-SAT at phase transition).

Figure 6.13 shows two typical *blmin* distributions, here for the easy and hard instance from the `uf50-218` test-set. As can be seen from the plots, the average branching of LMIN states is quite low; this indicates that the overall structure of plateaus is very brittle, and for most problems, the LMIN states encountered have a large number of direct escapes when using random walk. Consequently, when encountering a local minimum region, an SLS algorithm's chance for escaping within a small number of random walk steps is rather high. Also, the branching seems to be even lower for the hard instance. This observation suggests a possible correlation between the average local minima branching and the hardness of a problem.

The results of an analysis of this correlation for the reduced Random-3-SAT test-sets with different problem sizes are shown in Figure 6.14. The scatter plots and *lms* fits indicate a negative correlation which gets stronger with increasing problem size. Table 6.10 shows the corresponding correlation coefficients and regression parameters; for the larger instances, also a greater inclination of the regression line is observed.

Generally, the observed correlation indicates that problems having a less branched plateau structure tend to be harder for GWSAT. This might seem a bit counter-intuitive, because using random walk, less branched plateau structures should be both easier to escape from and less difficult to search. But somehow this effect is either not present or dominated by another phenomenon. Since we know from Section 6.2 that the number of solutions is highly correlated with local search cost, one could hypothesise that, like the standard deviation of the objective function *sdnclu*, the average branching is somehow coupled with the number of solutions, which in turn dominates other influences on the mean local search cost.

Figure 6.15 (left) shows the correlation between the average local minima branching and the number of solutions for test-set `uf50-218`. The correlation coefficient of 0.68 indicates

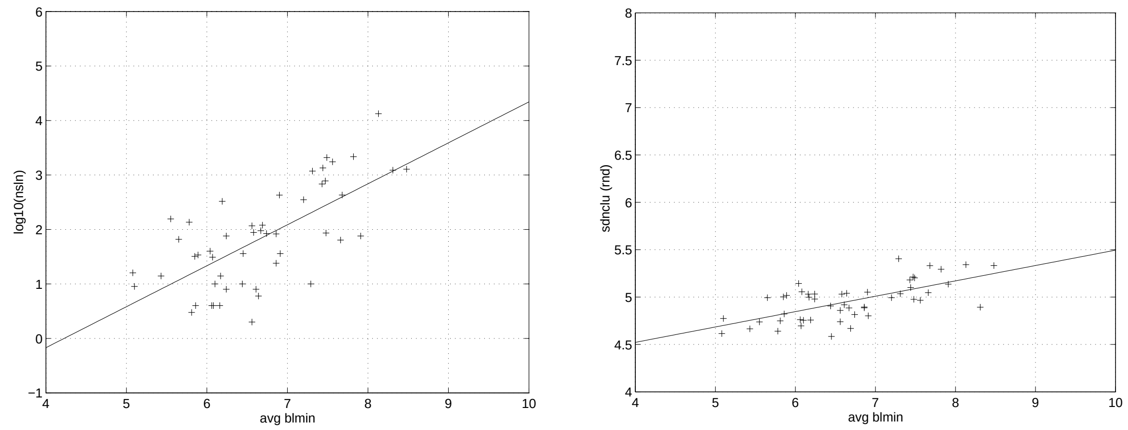


Figure 6.15: *Left:* Correlation between average *blmin* and number of solutions; *right:* same for average *blmin vs sdnclu*; all data for test-set `uf50-218` (Random-3-SAT at phase transition).

test-set	a	b	r
uf20-91/50	-0.06	2.13	-0.13
uf50-218/50	-0.20	4.14	-0.44
uf100-430/50	-0.19	5.84	-0.44

test-set	a	b	r
uf50-163/50	-0.14	3.48	-0.68
uf50-218/50	-0.20	4.14	-0.44
uf50-273/50	-0.14	3.30	-0.35

Table 6.10: Correlation between average *blmin* and the average local search cost, *left:* for different problem sizes at phase transition, *right:* across phase transition; a, b are the parameters of a *lms* linear fit using the function $ax + b$, r is the correlation coefficient.

that there is a slightly stronger correlation than the one observed between *sdnclu* and the number of solutions for the same test-set. As shown in Figure 6.15 (right), we also observe a rather strong correlation (correlation coefficient = 0.66) between the average local minima branching and *sdnclu* for the same test-set. This suggests the following explanation: For Random-3-SAT, a larger number of solutions tends to come along with clustering phenomena which, in turn, are characterised by both a larger branching of local minima as well as a higher standard deviation of the objective function values, *i.e.*, a more "rugged" search space topology. As a consequence, the average local minima branching can be used to estimate the number of solutions and therefore also the hardness of Random-3-SAT instances.

Determining the correlation between the average local minima branching and the mean local search cost for the 50 variable test-sets across the phase transition, we get similar results as the ones for *sdnclu* reported in Section 6.4. As can be seen from Figure 6.16 and the correlation data in Table 6.10, the correlation gets weaker with an increasing clauses per variable ratio. At the same time, for an increasing number of clauses, the average local minima branching decreases monotonically. This suggests that, for a fixed number of variables, the search space topology gets more and more rugged with an increasing number of clauses — an assumption which is confirmed by the results of [Yok97]. Furthermore,

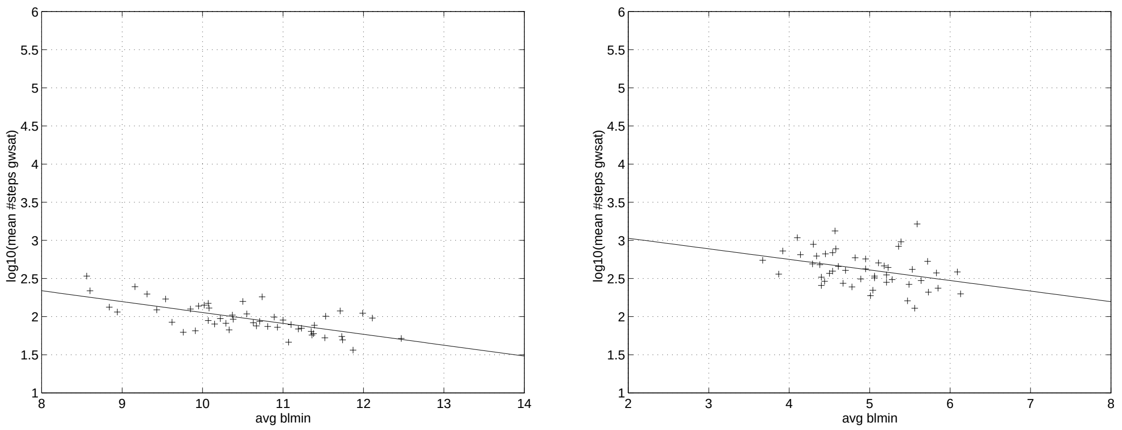


Figure 6.16: Correlation between average *blmin* and average local search cost (GWSAT, approx. optimal noise) for under- and overconstrained Random-3-SAT instances, test-sets `uf50-163` *(left)* and `uf50-273` *(right)*.

instance	variables	clauses	solutions	avg *lsc*	avg *blmin*
flat30-60/n19	90	300	432	1775.8	3.40
uf90-388/med	90	388	404	2492.1	10.35
flat50-115/n4	150	545	5892	4539.2	3.73
uf150-645/med	150	645	5884	10231.9	16.87

Table 6.11: Comparison of average local search cost and average local minima branching factors for Graph Colouring and Random-3-SAT instances.

at the phase transition the inclination of the regression line is maximal which indicates a stronger dependence of the average solution cost on the average local minima branching. Again, this observation is consistent with analogous observations for the dependence of the local search cost on the number of solutions and the explanation given above. Note that for high clauses per variable ratios, the average local minima branching seems to be a better estimator for the local search cost than *sdnclu* (*cf.* Tables 6.3 and 6.10).

Graph Colouring Next, we analysed the average local minima branching factor for instances from the Graph Colouring domain as compared to Random-3-SAT instances with the same search space size. The results are reported in Table 6.11; they indicate that Graph Colouring instances, which are significantly easier to solve for SLS algorithms, have a much lower average local minima branching factor than the hard Random-3-SAT instances. Also, with increasing problem size, the branching factor seems to grow faster for the Random-3-SAT instances than for the more structured Graph Colouring instances.

instance	variables	clauses	solutions	avg lsc	avg $blmin$
bwp/anomaly	48	261	1	944.2	2.69
uf48-210/n49	48	210	1	1,509.5	6.34
bwp/medium	116	953	2	1,787.9	3.44
uf116-499/n92	116	499	16	12,484.5	13.44

Table 6.12: Comparison of average local search cost and average local minima branching factors for Blocks World Planning and Random-3-SAT instances.

instance	variables	clauses	solutions	avg lsc	avg $blmin$
ais6	61	581	24	2,766.42	2.33
uf61-263/n75	61	263	24	1,695.37	7.19
ais8	113	1,520	40	64,147.11	2.83
uf113-486/n17	113	486	40	58,267.11	12.01

Table 6.13: Comparison of average local search cost and average local minima branching factors for All-Interval-Series and Random-3-SAT instances.

Blocks World Planning Performing a similar analysis for the two smallest Blocks World Planning instances, we obtained very similar results to those for the Graph Colouring domain (*cf.* Table 6.12). Clearly, the Blocks World Planning instances, which are significantly easier to solve for GWSAT, have a much lower average local minima branching factor than the hard Random-3-SAT instances. As for observed for the Graph Couloring instances, with increasing problem size, the branching factor seems to grow faster for Random-3-SAT than for the more structured Blocks World Planning domain.

All-Interval-Series Applying analogous methodology to the two smaller All-Interval-Series instances surprisingly gives a different result (*cf.* Table 6.13). Although compared to corresponding Random-3-SAT instances the local search cost (for WalkSAT, approx. optimal noise setting) is significantly higher, we observe much lower local minimum branching factors. This is not consistent with our observations for the other problem domains and strongly suggests that there are other factors which account for the relative hardness of instances from the All-Interval-Series domain.

Summing up the results of this section, by analysing the local minima branching factor $blmin$ along GWSAT trajectories we found evidence that the local minima structure is generally rather brittle than compact, *i.e.*, for the LMIN state visited by GWSAT, there is usually a relatively large number of direct escape routes when using random walk. In contrast, the often used plateau metaphor would suggest that the average $blmin$ value should be rather high, *i.e.*, close to n (the number of variables of the given problem instance), making it much more difficult for algorithms like GWSAT to escape from such plateaus. We also observed a positive correlation between the average $blmin$ value and the average local search cost for Graph Colouring and Blocks World Planning instances. This is consistent with our intuition that highly branched structures, which tend to have a higher potential for non-deterministic looping behaviour, are more difficult for SLS algorithms to search for solutions. However, for the All-Interval-Series domain, we observed very low branching factors despite the hardness of the instances. For Random-3-SAT, a negative correlation between the number of solutions and the average $blmin$ value dominates our observed results, suggesting that the average local minima branching factor can be used for estimating the number of solutions for Random-3-SAT instances.

6.7 Related Work

The analysis of search space structure and its influence on SLS performance is a rather new area of research. There is, however, a number of studies which are related to the approaches followed here.

Section 6.2 is mainly based on methodology used by Clark *et al.* [CFG$^+$96] for studying the correlation between the number of solutions and local search cost for CSP and SAT. However, reviewing the results reported in [CFG$^+$96], we find that their results for SAT are considerably less conclusive than those for CSP: Mainly, the reported correlation is considerably weaker, and while for CSP, a minimal regression gradient can be found at the phase transition, an analogous phenomenon could not be observed for SAT. Our results reported in Section 6.2 eliminate these inconsistencies by overcoming several methodological weaknesses of their analysis. Nevertheless, their main result, the existence of a strong negative correlation between the number of solutions and average local search cost, is confirmed by our refined analysis.

Parkes [Par97] follows a different approach by analysing the clustering of solutions for Random-3-SAT instances. His results show that solutions usually occur in large clusters; however, from his study it remains unclear whether this is beneficial or detrimental for SLS performance. Our analysis of stratified Random-3-SAT test-sets suggests that, while clustering effects might have an influence on SLS performance, there have to be other, more dominant factors. These probably include clustering phenomena regarding non-solution plateaus, which could be related to Parkes' results concerning "failed clusters".

Our approach of analysing state type distributions and local minima structure is partly related to Yokoo's analysis of how constrainedness affects a simple hill-climbing algorithm's performance for Random-3-SAT and (un-encoded) Graph Colouring test-sets across the phase transition [Yok97]. Our approach of measuring search space state type distributions is related to his counting of local minima states. However, his analysis is solely based on the exhaustive analyis of search spaces for very small instances. He also observes that strict local minima are very rare, but does not investigate the frequency of occurrence for other state types. His main result, stating that with increasing constrainedness, the number of local minima states and the size of the local minima basins decrease, are complemented by our analysis presented here.

Probably the most detailed study of search space topology so far can be found in [FCS97, Fra97b]. Unlike our approach, this work focusses on more global features of search space structure, like the size and number of different types of plateau regions. Their analysis is mainly restricted to a number of random problem instance distributions, including hard Uniform Random-3-SAT, the search spaces of which are sampled using GSAT. While many aspects of their approach towards search space analysis appear to be promising, their results are very limited in their scope. They show that local minima regions as well as "benches", *i.e.*, plateau regions with neighbouring states that have a lower objective function value, are the characteristic structures for a large part of the search space and are highly variable in size. Their analysis addresses neither the correlation between these features and SLS performance, nor the topology of local minima regions, which is highly relevant in the context of SLS behaviour. It is also not clear how their results apply to SAT-encoded instances from other problem domains.

The importance of plateaus and local minima for SLS behaviour has been also stressed by Gent and Walsh [GW93b, GW93a], who established that GSAT spends most of its time searching plateaus. Many SLS methods are based on the notion that the ability of escaping from local minima leads to considerably improved performance. Among these are GSAT variants like GWSAT [SKC94], HSAT [GW93b], and WalkSAT; but also different approaches like Tabu Search [MSG97, Glo89], Simulated Annealing [KJV83, SKC93, Spe93], or the Breakout Method [Mor93]. Recently it could be shown that information on search space structure can be successfully used for automatically tuning search behaviour: Boyan's and Moore's STAGE approach [BM98] improves local search behaviour based on features of the underlying search spaces which are measured during the search.

6.8 Conclusions

In this chapter, we investigated various aspects of search space structure and studied their correlation to the search cost for SLS algorithms. Analysing the number of solutions per instance and its correlation to average local search cost for GWSAT across various Random-3-SAT test-sets, we confirmed earlier results for basic GSAT by observing a strong negative correlation. The strength of this correlation, however, decreases monotonically for growing constrainedness, indicating that for over-constrained problem instances, there is considerably more variance in local search cost than for critically or under-constrained instances with a similar or identical number of solutions. At the same time, we could show that the influence of the number of solutions is maximal at the phase transition region.

To abstract from the strong impact of the number of solutions on local search cost, in our further analysis, we used mainly sets or pairs of instances with a similar or identical number of solutions. Analysing hardness distributions for such subsets (strata) of Random-3-SAT, we found that the local search cost is still extremely variable within these strata. From an analysis of a single-solution stratum we found evidence that the clustering of solutions, as studied in [Par97], seems to have no major effect on local search cost for Random-3-SAT.

Consequently, there have to be other features of search space structure which are responsible for the large hardness variation within the strata.

As possible candidates for such features, we have studied the standard deviation of the objective function (*sdnclu*) over the search space and the average local minima branching factor (avg *blmin*) along GWSAT trajectories. To our best knowledge, these two measures have not been studied before in the context of search space structure. Intuitively, large *sdnlcu* values should indicate a rugged search space structure for which stochastic local search approaches based on hill-climbing are more effective than for featureless, flat search spaces with many plateaus. This intuition could be empirically confirmed for the Graph Colouring and Blocks World Planning domains, while for Random-3-SAT test-sets and All-Interval-Series instances we did not observe this correlation.

Since we know from previous work, that the number of local minima states has an impact on local search cost [Yok97], we extended these results by studying the structure of local minima regions. Surprisingly, we found that local minima regions have a very brittle structure, such that for each local minimum state only a small number of its neighbours belong to the same plateau region. Thus, the plateau regions resemble multi-dimensional systems of narrow canyons rather than compact basins. Consequently, using techniques like random walk, it should be extremely easy to directly escape from plateau regions with only a small number of escape steps.

For Graph Colouring and Blocks World Planning instances, we observed a positive correlation between the average *blmin* value and the average local search cost. This conforms to the intuition that highly branched structures are more difficult for SLS algorithms, because they have a higher potential for non-deterministic looping behaviour and a smaller number of escape routes per local minima state. For the Random-3-SAT and All-Interval-Series domains, our observations did not confirm this intuition. This indicates the existence of other search space features, which have a significant influence on SLS performance.

In summary, we identified and analysed a number of search space features influencing the hardness of instances for local search. Some of these features, like the average local minima branching, appear to be considerably different between Random-3-SAT and SAT-encoded instances from other domains. However, our results cannot explain all the observed differences in instance hardness within and across problem domains. Consequently, there have to be other, yet unknown factors which affect local search performance. In this sense, our analysis of search space structure is far from being complete, but extends the previous state of knowledge with new methodological approaches and novel insights.

Chapter 7

SAT Encodings and Search Space Structure

SLS algorithms for SAT have been successfully applied to hard combinatorial problems from different domains. On one hand, these applications have become possible by using powerful SLS algorithms like WalkSAT. But they also critically rely on the use of suitable SAT-encodings. Because of the $\mathcal{NP}$-completeness result for SAT [Coo71], we know that instances from any $\mathcal{NP}$-complete problem can be encoded in SAT, and even 3-SAT, in polynomial time; consequently, the size of the encoded instances is also polynomial in the size of the original representations. However, this alone does not guarantee the existence of suitable SAT-encodings, since sufficiently small and efficiently computable SAT representations could still be extremely hard for existing SLS algorithms. In particular, this might happen if the encoding induces search space features which impede local search. Thus, the analysis of how encoding strategies affect search space structure and SLS performance is a very interesting and relevant issue in the context of a generic problem solving approach based on SAT as an intermediate representation domain and SLS algorithms as solvers for this domain (*cf.* Chapter 1).

In this chapter, we investigate the impact of encoding strategies on SLS performance and search space structure. Since this is a complex issue which has rarely been studied before, we can only undertake an initial investigation here. After giving further motivation and background on SAT-encodings, we present two case studies. First, we study sparse versus compact encodings for hard Random Binary CSP instances. Furthermore, we compare the performance of a state-of-the-art SLS algorithm for CSP with the performance of some of the best SLS algorithms for SAT applied to instances from this domain. The second case study extends the analysis of sparse versus compact encodings to hard Random Hamilton Circuit instances. For this problem domain, we also investigate the effects of symmetry elimination on different problem representations and encodings. After a brief look on related work, the chapter closes with a summary of our main results and some conclusions.

7.1 Motivation and Background

Until now, our study of SLS algorithms, their behaviour, and their applications has been focussed on the algorithms themselves, the analysis and characterisation of their run-time behaviour, and on the factors which influence their performance. Coming back to the generic problem solving model from Chapter 1, one issue has not been addressed so far: the influence of the encoding strategies used for translating problems from other domains into the intermediate domain (in our case, SAT) on the performance of SLS algorithms for the encoded problem instances. In Chapter 4, where we evaluated a number of modern SLS algorithms for SAT on various problem classes, we always used encodings which are already known to be effective in the sense that they facilitate good SLS performance. However, devising such encodings is usually not simple. At the same time, finding effective encodings is a crucial condition for successfully applying the generic problem solving scheme.

When developing encodings, two fundamental issues are space and time complexity: As argued before, both encoding problem instances and decoding solutions have to be efficient enough that the overhead compared to directly using a solver for the original domain is not too high. Also, the encoded problem instances should not grow too large compared to the original representation. When using SAT as an intermediate domain, and applying SLS algorithms for solving the encoded instances, intuitively it seems that the number of propositional variables should be kept small, because this way, smaller search spaces are obtained. However, it is not clear whether encodings which are optimised for a minimal number of propositional variables are really advantageous, as they might induce search space features which significantly impede stochastic local search.

Furthermore, some problem domains (such as the Hamilton Circuit Problem discussed later in this chapter, or problems from various planning domains) allow different basic representations, where *a priori* it is not clear which of several representations is most efficient in conjunction with the generic problem solving approach outlined before. Another issue is symmetry elimination and pruning — depending on the original problem formulation, often the search space can be pruned before encoding the problem; also, problems have often symmetric solutions which can be collapsed before encoding them into the intermediate domain. This is the case for the Hamilton Circuit Problem, where for the basic representations we will use later, a symmetry is given by the starting point of the cyclic tour through a given graph. Again, *a priori* it is not clear, whether eliminating such symmetries is advantageous w.r.t. maximising SLS performance on the encoded problem instances.

Today, these and many other issues are not well understood. Clearly, at this point and in the context of this thesis we can only study some particular aspects of this widely open problem. In the following, we therefore restrict ourselves to three basic questions:

- Does it pay off to minimise the number of propositional variables, *i.e.*, are compact encodings which achieve small search spaces preferable to sparse encodings?
- Is it advisable to reduce the number of propositional variables required for a given en-

coding by eliminating symmetric solutions from the original problem formulation?

- Is the generic problem solving approach competitive with applying SLS algorithms directly to the un-encoded problem instances?

While we cannot give definitive answers to these questions here, the two case studies presented in the following demonstrate how these issues can be investigated and also suggest tentative answers which can and should be tested and further investigated in the future.

7.2 Constraint Satisfaction Problems

Constraint Satisfaction Problems have been formally introduced in Chapter 1. Like SAT, finite discrete CSP is an $\mathcal{NP}$-complete combinatorial problem. Conceptually it is very closely related to SAT; as a consequence, most SLS algorithms for SAT can be generalised to CSP in a rather straightforward way. Many combinatorial problems from different domains can be quite naturally represented as CSP instances; compared to SAT encodings of these problems, CSP representations are more compact and often easier to devise. Therefore, when transforming combinatorial problems into SAT, CSP is often implicitly or explicitly used as an intermediate representation (*cf.* Chapter 4, the SAT-encodings of the Graph Colouring and All-Interval-Series problems). Nevertheless, as mentioned before, SAT is conceptually simpler than CSP, and consequently, SAT algorithms are often easier to develop, implement, and evaluate. In this section, we investigate SAT-encodings of binary CSPs and compare the application of SLS-based SAT algorithms to SAT-encoded CSPs with directly solving CSP instances using SLS algorithms for CSP.

7.2.1 SAT Encodings of the CSP

CSP instances can be easily encoded into SAT by using propositional variables to represent the assignment of values to single CSP variables and clauses to express the constraint relations [dK89]. To simplify the formalism for specifying SAT encodings of discrete finite CSP instances, we assume (without loss of generality) that the variable domains D_i are all equal to $\mathbb{Z}_k = \{0, 1, \ldots, k-1\}$, where k is an arbitrary positive integer. Furthermore, we denote the arity of a constraint relation C_j by $\sigma(C_j)$.

A very natural SAT-encoding for a given CSP $P = (X, \mathcal{D}, \mathcal{C})$ with $X = \{x_1, \ldots, x_n\}$, $\mathcal{D} = \{D_1, \ldots, D_n\}$, and $\mathcal{C} = \{C_1, \ldots, C_k\}$ is based on propositional variables $c_{i,\nu}$ which, if assigned the value $\top$, represent the assignment $x_i = \nu$, where $\nu \in D_i$. P can then be represented by a CNF formula comprising the following sets of clauses:

(1)	$\neg c_{i,\nu_1} \vee \neg c_{i,\nu_2}$	$(1 \leq i \leq n; \nu_1, \nu_2 \in D_i; \nu_1 \neq \nu_2)$
(2)	$c_{i,1} \vee c_{i,2} \vee \ldots \vee c_{i,k}$	$(1 \leq i \leq n)$
(3)	$\neg c_{i_1,\nu_1} \vee \neg c_{i_2,\nu_2} \vee \ldots \vee \neg c_{i_s,\nu_s}$	$(x_{i_1} = \nu_1; x_{i_2} = \nu_2; \ldots; x_{i_s} = \nu_s$ violates some constraint $C \in \mathcal{C}$ with $\sigma(C) = s)$

Intuitively, these clause sets ensure that each constraint variable is assigned exactly one value from its domain (1, 2) and that this assignment is compatible with all constraints (3). Obviously, the number of propositional variables required for encoding a given CSP instance is linear in the number of constraint variables and their domain sizes while the number of clauses is at least linear in the number of constraint variables and depends critically on the domain sizes and the arity of the constraints. This encoding has been frequently used for translating problems from other domains into SAT (*cf.* Chapter 4, Section 4.4). We call it the *sparse encoding*, because it generates large SAT instances the models of which have only a small fraction of the propositional variables set to $\top$.

Since the search space size for a given formula is exponential in the number of propositional variables it comprises, it appears to be reasonable to minimise the number of variables required by a SAT-encoding. In the case of the SAT-encodings for CSP instances, the number of propositional variables required for encoding a given CSP instance can be significantly reduced compared to the sparse encoding. This is achieved by a binary encoding of the value assigned to a constraint variable x_i using a group of $\kappa = \lceil \log_2 k \rceil$ propositional variables $c_{i,j}$ (k is the domain size of the CSP instance). To formalise this *compact encoding*, we use an auxiliary function which maps a positive integer ν to a propositional clause which is true exactly if the corresponding assignment does *not* correspond to the binary representation of ν:

$$c(i, \nu) = \bigvee_{j=1}^{\kappa} l(\nu, j, c_{i,j})$$

$$l(\nu, j, z) = \begin{cases} \neg z & \textit{if } (\nu \text{ div } 2^{i-1}) \bmod 2 > 0 \\ z & \textit{otherwise} \end{cases}$$

Now, we can represent a CSP instance P by a CNF formula using the following sets of clauses:

(1)	$c(i, \nu)$	$(1 \leq i \leq n; k \leq \nu < 2^{\kappa})$
(2)	$c(i_1, \nu_1) \vee c(i_2, \nu_2) \vee \ldots \vee c(i_s, \nu_s)$	$(x_{i_1} = \nu_1, x_{i_2} = \nu_2, \ldots, x_{i_s} = \nu_s$ violates some constraint $C \in \mathcal{C}$ with $\sigma(C) = s)$

Intuitively, these groups of clauses ensure that no constraint variable is assigned an invalid value (1) and that the assignment does not violate any constraint in $\mathcal{C}$ (2). This compact encoding requires $O(n \cdot \log k)$ propositional variables which is a considerable reduction compared to the $O(n \cdot k)$ variables used for the sparse encoding. Usually, the clauses encoding

test-set	instances	variables	domain size	α	β	vars	clauses
csp20-10-s	100	20	10	0.5	0.38	200	$\approx$ 4,305
csp20-10-c	100	20	10	0.5	0.38	80	$\approx$ 3,504

Table 7.1: Random Binary CSP test-sets, using sparse (`-s`) and compact (`-c`) SAT-encoding; as the number of clauses varies between the instances, we report the average number of clauses here.

the constraint relations dominate the size of the formulae produced by both encodings, such that the number of clauses will be similar. But because of the difference in the number of propositional variables, the compact encoding generates search spaces which are smaller by a factor of $O(n \cdot (k - \log k))$.

7.2.2 Benchmark Instances

For our empirical study presented in the next sections, we focussed on Uniform Random Binary CSP, a random distribution of CSP instances based on random binary constraint relations. More specifically, the problem distribution is characterised by the *constraint graph density* α and the *constraint tightness* β; α specifies the fixed probability that between two randomly chosen constraint variables a constraint relation exists, while β is the expected fraction of value pairs which are compatible with a given constraint relation between two variables.

For this problem class, a phase transition phenomenon, similar to the one for Uniform Random-3-SAT, has been observed [Smi94]. To obtain a test-set of hard problem instances, CSP instances were sampled from the phase transition region of Uniform Random Binary CSP with 20 variables and domain size 10, characterised by a constraint graph density of $\alpha = 0.5$ and a constraint tightness of $\beta = 0.38$. Filtering out the insoluble instances with a complete CSP algorithm, test-set `csp20-10`, containing 100 soluble instances from this problem distribution, was generated. These instances were translated into corresponding sets of SAT instances using the sparse and the compact encodings (*cf.* Table 7.1). The generator and the test-sets used for the subsequent studies were kindly provided by Thomas Stützle (TU Darmstadt, Germany).

7.2.3 Two SLS Algorithms for CSP

To study the performance difference between SLS algorithms for SAT applied to SAT-encoded CSP instances and SLS algorithms for CSP directly applied to the CSP instances, we use two competetive variants of the well-known *Min-Conflicts Heuristic (MCH)* [MJPL92], which are obtained by extending the basic MCH with random walk *(WMCH)* and tabu-lists *(TMCH)*, respectively [SSS97]. These algorithms are conceptually closely related to

```
function WMC(a) is
  % stage 1: variable selection:
  randomly select a variable x_i occurring in a currently violated constraint;
  % stage 2: value selection:
  with probability wp do
    V' := set of values for x_i which minimise the number of conflicts;
  otherwise
    V' := D_i;
  end with;
  ν := draw(V');
  a' := a with x_i set to ν;
  return (a');
end WMC;
```

Figure 7.1: GLSM state for realising WMCH.

WalkSAT and WalkSAT+tabu and can be modelled as 1-state+init GLSMs using the same overall structure as GSAT (*cf.* Figure 4.1, page 75) but the WMC and TMC states defined in Figures 7.1 and 7.2 instead of the GD state, and using a modified initialisation state which assigns a randomly chosen value to each constraint variable and, for TMCH, initialises the tabu-list (initially empty).

For both algorithms, in each local search step, first a constraint variable involved in a violated constraint is randomly selected,[1] then a new value ν for this variable is chosen. The Min-Conflicts Heuristic's general approach for selecting the new value ν is to minimise the number of constraint violations (conflicts); thus, the scoring function is closely related to the one used by GSAT or Novelty. WMCH and TCMH use different mechanisms for selecting ν; while WMCH extends the basic selection strategy with random walk much in the spirit of WalkSAT, TCMH uses a tabu-list of variable/value pairs to avoid stagnation behaviour. Additionally, TMCH incorporates an aspiration criterion which allows values which are tabu to be selected when they improve on the overall best solution since the last restart.

7.2.4 SLS Performance

Our analysis of SLS performance is divided into three parts. First, we compare the performance of WalkSAT on the different encodings; as a result of this, we see that while the local search cost for the sparse encodings is uniformly lower than for the compact encoding, there is a tight correlation of SLS performance between both encodings of individual instances across the test-set. Next, we compare the different SLS-based SAT algorithms' performance when applied to the sparsely encoded test-set. As we will show, the performance of all SLS algorithms, when applied to the same problem instance, is closely correlated — which in-

[1] in this context, all random choices from sets are based on uniform distributions, *i.e.*, each element has the same probability for being selected.

```
function TMC(a) is
  % stage 1: variable selection:
  randomly select a variable x_i occurring in a currently violated constraint;
  % stage 2: value selection:
  s' := minimal number of conflicts when setting x_i to any value ∈ D_i;
  if s' < minimal number of conflicts ever encountered during this search then
    V' := set of values for x_i which minimise the number of conflicts;
  else
    V' := set of values for x_i which are currently not tabu;
    V'' := elements of V' which minimise the number of conflicts;
  end if;
  ν := draw(V');
  a' := a with x_i set to ν;
  if length of tabu-list > tl then
    remove oldest element from tabu-list;
  add (x_i, ν) to tabu list;
  return (a');
end TMC;
```

Figure 7.2: GLSM state for realising TMCH.

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	$\frac{Q_{75}}{Q_{25}}$
csp20-10-s	23,408.70	28,770.74	1.23	12,516.37	6,168.41	30,260.64	4.91
csp20-10-c	137,689.53	199,287.07	1.45	70,042.56	22,841.89	162,885.79	7.13

Table 7.2: Test-set csp20-10, basic descriptive statistics of hardness distributions for sparse and compact SAT-encoding, using WalkSAT (approx. optimal noise, 100 tries/instance).

dicates that the performance of the different SLS methods depends on the same intrinsic features of the problem instances. Finally, we compare the performance of SLS algorithms for CSP applied to the original CSP instances with that of the SLS-based SAT algorithms applied to the sparsely encoded test-set; the result of this analysis indicates that the latter approach is quite competitive.

Sparse *vs* Compact SAT Encodings

To investigate the performance differences for SLS-based SAT algorithms between the two different encodings, we first determined the distribution of the average local search cost (hardness distribution) for WalkSAT across the test-sets csp20-10-s and csp20-10-c. For these and all the following experiments, we used approx. optimal noise settings and cutoff parameters high enough (maxSteps $\geq 10^6$) to guarantee maximal success probabilities. The average local search cost for each instance is estimated from 100 tries. The results, as can be seen in Figure 7.3, clearly indicate that WalkSAT works significantly more efficient on the sparse than on the compact encoding. Furthermore, it can be seen from the normalised

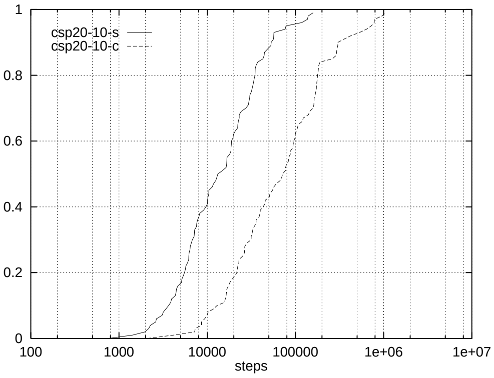


Figure 7.3: Hardness distributions (average local search cost per instance) for WalkSAT, using approx. optimal noise, when applied to test-sets `csp20-10-s` and `csp20-10-c` (sparse *vs* compact SAT-encoding for CSP).

standard deviation and the percentile ratios of the hardness distributions (*cf.* Table 7.2) that the compact encoding induces more variability between the instances than the sparse encoding. Note that the variability for the sparse encoding is comparable to Random-3-SAT at the phase transition (*cf.* Table 4.10, page 96).

To investigate this in more detail, next, we analysed the correlation between WalkSAT's performance on the different encodings of the individual instances, using the same methodology as in Chapter 4 and the parameter settings described above. Figure 7.4 shows the correlation results as a scatter plot; as can be easily seen, there is a strong linear correlation between the logarithm of WalkSAT's mean local search cost on the differently encoded test-sets. This indicates that generally, the algorithm's behaviour depends more on properties of the original problem instance than on features induced by the encoding. However, the correlation shows some noise, especially for instances with high local search cost, which means that independent from instance hardness, the encoding has an additional influence on SLS performance. But since the correlation is very strong (correlation coefficient ≈ 0.95), this influence seems to be rather limited.

As we have seen from our analysis of the hardness distributions, w.r.t. SLS performance, the compact encoding seems to be generally inferior compared to the sparse encoding. Performing regression analyses (as introduced in Chapter 4) for the performance correlation in Figure 7.4 shows additionally that the performance difference slightly decreases as the instances get harder. Thus, the compact encoding is particularly inferior for relatively easy problem instances, while for intrinsically hard instances, the influence of the encoding on SLS performance is somewhat weaker.

To make sure that WalkSAT behaves regularly on both test-sets, we analysed the RLDs for the easy, medium, and hard instance (determined as described in Chapter 4) in both encodings. As expected, the results show that for RLDs, best-fit *ged* approximations (*cf.*

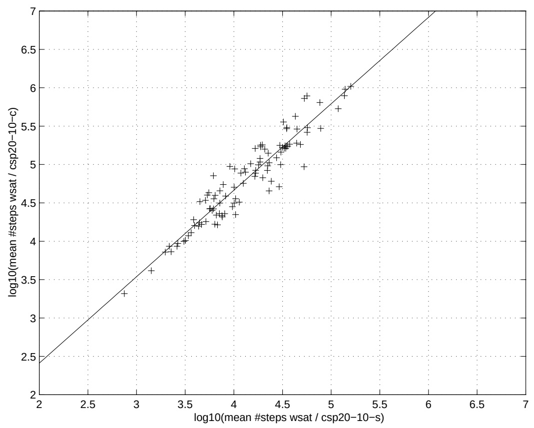


Figure 7.4: Correlation of average local search cost per instance between sparse (horizontal) and compact SAT-encoding (vertical) for CSP instances from test-set `csp20-10` when using WalkSAT with approx. optimal noise.

Chapter 5) pass a standard χ^2 test for the $\alpha = 0.05$ acceptance level. Interestingly, when using standard exponential distributions (which do not model the initial search phase), the approximations for the medium and hard instances for both encodings still pass the test. This indicates that these instances are relatively hard, such that the initial search phase has no significant influence on WalkSAT's behaviour.

Comparison of SLS Performance

Next, we compared the performance of different SLS algorithms on the sparsely encoded test-set `csp20-10-s`. We applied the same methodology as in Chapter 4, Section 4.5, when we evaluated SLS algorithms on Random-3-SAT test-sets. In particular, for each SLS algorithm we measured the distribution of the average local search cost across the test-set, using approximately optimal noise settings and sufficiently high maxSteps settings to guarantee maximal success rates in all cases. The resulting hardness distributions are shown in Figure 7.5. Obviously, for the major part of the test-set, the following performance ordering can be observed: Novelty shows almost identical performance as GSAT+tabu, followed closely by R-Novelty and WalkSAT+tabu; next comes WalkSAT, which performs significantly worse, and finally GWSAT. Not very surprisingly, given their essential incompleteness, Novelty, R-Novelty, and WalkSAT+tabu suffer from stagnation behaviour on 3–7% of the problem instances. Interestingly, for GSAT+tabu, although most probably essentially incomplete as well, such stagnation behaviour could not be observed on the given test-set.

In a next step, we analysed the correlation between the different algorithm's average performance across the test-set. As for Random-3-SAT (Chapter 4, Section 4.5), we chose

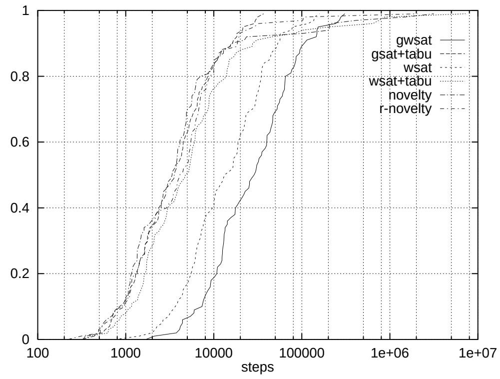


Figure 7.5: Hardness distributions (average local search cost per instance) for different SLS algorithms, when applied to test-set `csp20-10-s` (using approx. optimal noise).

encodings	r	a	b	o
gwsat *vs* gsat+tabu	0.9845	0.9471	**-0.6609**	0
gwsat *vs* wsat	0.9874	0.9587	-0.1108	0
gwsat *vs* wsat+tabu	0.9676	0.9047	-0.4128	6
gwsat *vs* novelty	0.9809	**0.8571**	-0.3775	7
gwsat *vs* r-novelty	0.9567	0.9105	-0.4939	3

Table 7.3: Test-set `csp20-10-s`, pairwise hardness correlation for various algorithms with approx. optimal noise, based on 100 tries / instance; r is the correlation coefficient, a and b are the parameters of the *lms* regression analysis, and o the number of outliers which have been eliminated before the analysis (see text).

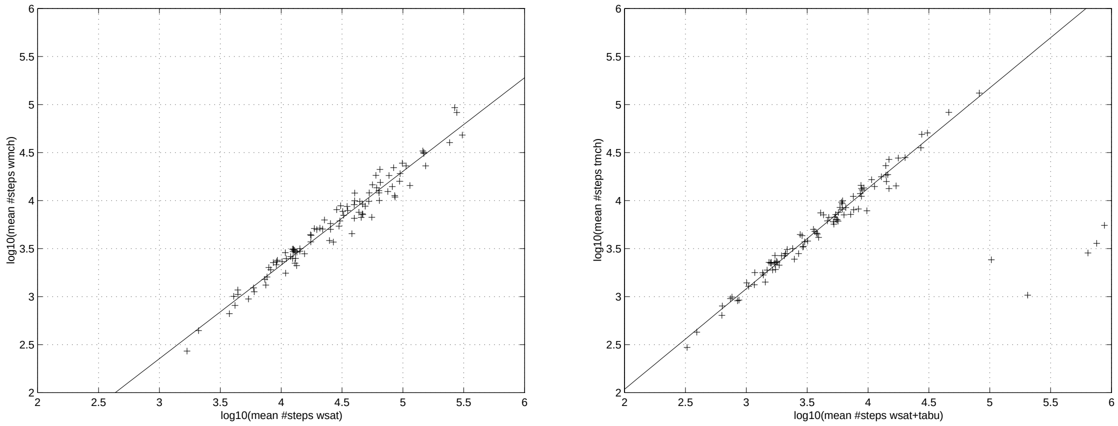


Figure 7.6: Correlation of average local search cost per instance for WalkSAT *vs* WMCH *(left)* and WalkSAT+tabu *vs* TMCH *(right)* applied to test-set `csp20-10`; all algorithms use approx. optimal noise, the SAT algorithms use the sparse encoding.

encodings	r	a	b
wsat *vs* wmch	0.9793	0.9752	-0.5715
wsat+tabu *vs* tmch	0.9894	1.0458	-0.0556
novelty *vs* tabu-gh	0.9614	1.0153	-0.3615

Table 7.4: Test-set `csp20-10`, pairwise hardness correlation for CSP and SAT algorithms (applied to sparse encoding) with approx. optimal noise, based on 100 tries / instance; r is the correlation coefficient, a and b are the parameters of the *lms* regression analysis, outliers have been eliminated before the analysis (see text).

GWSAT as a reference algorithm. The results of the correlation analysis are reported in Table 7.3. When ignoring the cases, where stagnation behaviour occurred, there is generally a strong correlation between the algorithms' average local search cost across the test-set, indicating that the hardness of instances w.r.t. to the given set of algorithms is an intrinsic property of the instances. The data from the regression analysis (a and b values in Table 7.3) indicate that Novelty scales best with instance hardness, followed by WalkSAT+tabu and R-Novelty, WalkSAT and GSAT+tabu, and finally, GWSAT. The interesting observation here is that, analogously to the results on Random-3-SAT instances (*cf.* Table 4.11, page 97), GSAT+tabu shows very competitive average performance but scales worse with instance hardness than the WalkSAT variants.

SLS Algorithms for SAT *vs* CSP

Given the close conceptual relation between SLS algorithms for SAT and CSP, directly comparing the performance of the corresponding algorithms is interesting in the light of

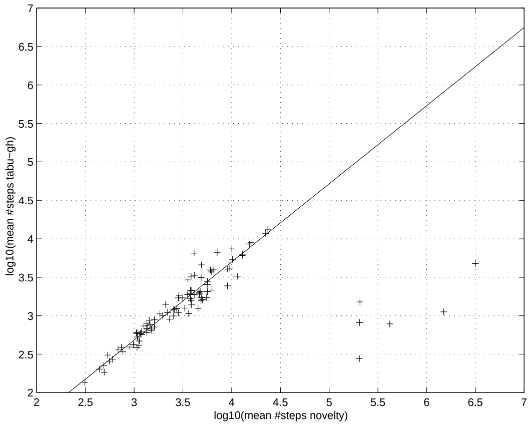


Figure 7.7: Correlation of average local search cost per instance for Novelty *vs* Galinier's and Hao's tabu search algorithm for CSP applied to test-set `csp20-10`; all algorithms use approx. optimal noise, Novelty uses the sparse encoding.

investigating the effectivity of the generic problem solving approach discussed in Chapter 1. Specifically, we first compare the performance of WalkSAT and WMCH as well as WalkSAT+tabu and TMCH, where the CSP algorithms work directly on the instances of test-set `csp20-10`, while the SAT algorithms are applied to the sparsely encoded problem instances (test-set `csp20-10-s`). The TMCH and WMCH implementations used for our empirical study were kindly provided by Thomas Stützle (TU Darmstadt, Germany).

Our performance comparison is based on correlation analyses, using the same methodology as in the previous section. The results of these correlation analyses are graphically depicted in Figure 7.6 and summarised in Table 7.4. Clearly, there is a very strong correlation between the performance of the SAT algorithms and the corresponding CSP algorithms across the test-set. While WMCH is slightly better than WalkSAT (both using approx. optimal noise), the performance difference between TMCH and WalkSAT+tabu is practically negligible, except for the fact that WalkSAT+tabu suffers from stagnation behaviour for 6 instances, while TCMH shows no such outliers. However, as argued in Chapter 5, this effect of WalkSAT+tabu's essential incompleteness should be easy to overcome by slightly modifying the algorithm.

The results from the corresponding regression analyses (*cf.* Table 7.4) indicate that while WMCH scales slightly better than WalkSAT with instance hardness, for TMCH and WalkSAT+tabu, the situation is reversed: here, WalkSAT+tabu has a slight scaling advantage, as can be seen from the fact that the slope of the regression line (a parameter) is greater than one.

Finally, we apply the same analysis to the best-performing SLS-based SAT algorithm and the best SLS-based CSP algorithm we are aware of — the tabu search algorithm by Galinier and Hao [GH97]. Again, using optimal noise parameters for both algorithms, we find that, measuring the average number of local search steps per solution, Galinier's and Hao's CSP algorithm has an advantage of a factor between 2 and 4 over Novelty (ignoring the outliers, see below). The correlation between both algorithm's performance is very strong (*cf.* Figure 7.6, Table 7.4); however, like WalkSAT+tabu, Novelty suffers from stagnation behaviour (on 7 instances), while the CSP algorithm shows no such behaviour. Interestingly, the regression analysis indicates that both algorithms show approximately the same scaling behaviour w.r.t. instance hardness. This is somewhat surprising, since Galinier's and Hao's algorithm and Novelty are conceptually significantly less closely related than WMCH and WalkSAT, or TMCH and WalkSAT+tabu.

Generally, the results from comparing the performance of SLS-based algorithms for CSP and SAT on the test-set of problem instances indicate that when measuring the average number of steps per solution, the differences are surprisingly small. We deliberately refrained from comparing CPU-times for these algorithms, because the implementations for the SAT algorithms are significantly more optimised. Furthermore, while the SAT algorithms can be applied to arbitrary CNF formulae, the implementations of the CSP algorithms are restricted to binary CSPs — which severely limits the range of their application. Of course, our analysis is too limited to give a conclusive answer, but the results reported here suggest that for solving hard CSP instances, encoding them into SAT and using a state-of-the-art SLS algorithm for SAT to solve the SAT-encoded instances might be very competitive compared to using specialised SLS-based CSP solvers.

7.2.5 Search Space Structure

After investigating the influence of the encoding strategy on SLS performance, we now analyse the search space structure using the methods from Chapter 6. In particular, we measure the number of solutions, the standard deviation of the objective function, and local minima branching for both encodings of the easy, medium, and hard problem instance from the `csp20-10` test-set using the same methodology as in Chapter 6. Only here, because the formulae generated by the different encodings have different numbers of clauses and variables, these measures have to be normalised to make them comparable. Specifically, the normalised *sdnclu* value is defined as the standard deviation of the objective function after it has been scaled to the interval $[0, 1]$, while for a LMIN state, the normalised *blmin* value measures the fraction of neighbouring states with the same objective function value.

Table 7.5 summarises the results from the analyses of the various search space features. Obviously, the number of solutions is identical for both encodings (this is guaranteed by the definition of the encodings). Since at the same time, the compact encoding achieves a significant reduction of search space size, we observe a vast difference in solution density between the sparse and compact encoding. Given the dominant role of this factor on SLS performance established earlier (*cf.* Chapter 6), it is rather surprising that WalkSAT's performance is so much worse for the compact encoding. However, comparing the normalised *sdnclu* and average *blmin* values between the two encodings, we see that for the compact

instance	solutions	solution density	norm. *sdnclu*	norm. *blmin*	avg *lsc*
csp20-10-s/easy	37,297	$2.01 \cdot 10^{-56}$	0.03655	0.015	775.80
csp20-10-s/medium	6	$3.73 \cdot 10^{-60}$	0.03606	0.017	11,162.69
csp20-10-s/hard	2	$1.24 \cdot 10^{-60}$	0.03597	0.017	134,777.38
csp20-10-c/easy	37,297	$3.09 \cdot 10^{-20}$	0.00114	0.131	2,249.19
csp20-10-c/medium	6	$4.96 \cdot 10^{-24}$	0.00111	0.157	77,883.52
csp20-10-c/hard	2	$1.65 \cdot 10^{-24}$	0.00113	0.161	1,000,456.82

Table 7.5: Easy, medium, and hard instance from test-set `csp20-10`, number of solutions, solution density, normalised *sdnclu* and average *blmin* values, as well as average local search cost (w.r.t. WalkSAT, using approx. optimal noise, 1,000 tries) for easy, medium, hard instance using different encodings.

encoding, the standard deviation of the objective function is much lower while the local minima are significantly more branched than for the sparse encoding. Intuitively, as argued in Chapter 6, this explains why WalkSAT has significantly more difficulty finding solutions in the search spaces induced by the compact encoding.

Regarding the differences in local search cost between the easy, medium, and hard problem instances when the same encoding is used, these are most probably mainly caused by the differences in solution density, as both, *sdnclu* and average *blmin* values, are relatively similar. This shows that while for a given encoding, the number of solutions is the dominant factor regarding local search cost, the different features induced by the encodings (some of which are measured by the *sdnclu* and average *blmin* values) have a drastically stronger influence on SLS performance.

7.3 Hamilton Circuit Problems

The *Hamilton Circuit Problem (HCP)* is a well-known combinatorial decision problem from graph theory. It is based on the notion of a *Hamilton Circuit* (sometimes also called *Hamilton Cycle*), *i.e.*, a cyclic path in a given graph G which visits evey vertex of G exactly once. For a given graph G, the Hamilton Circuit Problem is to decide whether G contains at least one Hamilton Circuit. The HCP is closely related to the Traveling Salesperson Problem (TSP), which can be regarded as a generalisation of the HCP where additionally the graph is weighted and the goal (in the optimisation variant) is to find a Hamilton Circuit with minimal weight, *i.e.*, a shortest round trip. Like the TSP, the HCP can be formulated for directed and undirected graphs and is $\mathcal{NP}$-complete in both cases. In this section we study the differences in SLS performance and search space structure for different SAT-encodings of directed HCP.

7.3.1 SAT Encodings of the HCP

The fundamental idea for encoding HCP into SAT is based on the observation that for a given graph $G = (V, E)$, each Hamilton Circuit (HC) corresponds to a cyclic permutation of G's vertex set V. However, a cyclic vertex permutation π_V corresponds only to a HC, if additionally for each pair of successive elements of π_V, there is an edge in E connecting the corresponding vertices. This can be formalised as a CSP the variables of which represent the positions of the permutation and their values determine the vertices appearing at these positions:

> For a given directed graph $G = (V, E)$, $HCP_p(G)$ is represented by the CSP $(X, \mathcal{D}, \mathcal{C})$, where $X = \{p_0, \dots, p_{\#V-1}\}$, for all i, $\mathcal{D}_{p_i} = V$, and $\mathcal{C}$ consists of the following constraints:[2]
>
> (1) $p_i = x \wedge p_k = y \iff x \neq y \vee i = k \quad (i, k \in \mathbb{Z}_{\#V})$
> (2a) $p_{i-1} = x \wedge p_i = y \iff (x, y) \in E \quad (i \in \mathbb{Z}_{\#V} - \{0\})$
> (2b) $p_{\#V-1} = x \wedge p_0 = y \iff (x, y) \in E$

This can be transformed into SAT using the sparse or compact SAT-encodings for CSP from Section 7.2. We refer to this encoding as the *position-based encoding*, since in the CSP formulation we assign vertices to permutation positions. However, there is a dual representation, based on the idea of representing the permutations by assigning permutation positions to vertices rather than vertices to permutation positions. The corresponding CSP formalisation of this *vertex-based encoding* is very similar to the one given above:

> For a given directed graph $G = (V, E)$ with $V = \{0, \dots, n-1\}$, $HCP_v(G)$ is represented by the CSP $(X, \mathcal{D}, \mathcal{C})$, where $X = \{v_0, \dots, v_{n-1}\}$, for all i, $\mathcal{D}_{v_i} = \mathbb{Z}_n$, and $\mathcal{C}$ consists of the following constraints:
>
> (1) $v_i = x \wedge v_k = y \iff x \neq y \vee i = k \quad (i, k \in \mathbb{Z}_n)$
> (2a) $v_i = x - 1 \wedge v_k = x \iff x \geq 1 \wedge (i, k) \in E \quad (i, k \in \mathbb{Z}_n)$
> (2b) $v_i = n - 1 \wedge v_k = 0 \iff (i, k) \in E \quad (i, k \in \mathbb{Z}_n)$

As for the position-based encoding, this CSP representation can be easily translated into SAT using one of the CSP encodings discussed before. Thus, we obtain four different SAT-encodings for the Hamilton Circuit Problem.

When re-examining these CSP-encodings, one might notice that they contain a certain redundancy. For any Hamilton Circuit, it is irrelevant which vertex is visited first, while the permutation-based encodings differentiate between solutions with different starting points. Thus, for each Hamilton Circuit in a given graph with n vertices, the corresponding CSP instance has n symmetrical solutions. This symmetry can be exploited to reduce the problem

[2] For clarity's sake, we are using a different notation here than in Definition 1.6; however, both are equivalent and can be easily transformed into each other.

test-set	instances	vertices	edges	d	vars	clauses
hc10-p	100	10	15	3	100	1,060
hc10-v	100	10	15	3	100	1.060
hc10-vc	100	10	15	3	40	1,110
hc10-pe	100	10	15	3	81	$\approx$ 733
hc10-ve	100	10	15	3	81	$\approx$ 733
hc10-vce	100	10	15	3	36	$\approx$ 781

Table 7.6: Random HCP test-sets, using different SAT-encodings; v = vertex-based, p = permutation based, c = compact (if not specified, encoding is sparse), e = symmetry eliminated (only, if specified); d is the mean vertex degree.

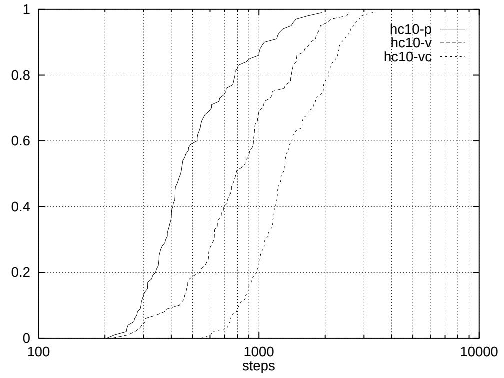
size of the CSP (and SAT) instances generated by our two encoding schemes. To achieve this *symmetry elimination*, we just fix an arbitrary element of the permutation to an arbitrary vertex; this eliminates one CSP variable for both encodings and reduces the size of all the remaining variables' domains by one.

For a graph with n vertices, we thus reduce the number of propositional variables from n^2 to $(n-1)^2$ when using the sparse encoding, and from $n\lceil\log_2 n\rceil$ to $(n-1)\lceil\log_2(n-1)\rceil$ when using the compact encoding. The number of clauses also shrinks because for the fixed element of the permutation, the number of constraints of type (2a) and (2b) is reduced. At the same time, the number of solutions is reduced by a factor of n. Consequently, for the sparse encoding, the net result of symmetry elimination is a reduction of the search space size by a factor of 2^{2n-1} ($= 2^{n^2} / 2^{(n-1)^2}$ = original search space size / reduced search space size) and an *increase* in solution density (solutions / search space states) by a factor of $2^{2n-1}/n$. For the compact encoding, the search space shrinks only by a factor of ca. 2, while the solution density *decreases* by a factor of ca. $n/2$. Considering the strong correlation between the number of solutions and local search cost observed in Chapter 6, one might expect that symmetry elimination reduces search cost for the sparse encoding, while for the compact encoding, savings are less likely.

7.3.2 Benchmark Instances

For empirically investigating SLS performance for different SAT-encodings of the HCP, we focussed on the Hamilton Circuit Problem in random graphs. In earlier work, when investigating the dependence of the existence of Hamilton Circuits on the average connectivity (edges per vertex), a phase transition phenomenon was observed [CKT91]. The associated peak in hardness for backtracking algorithms was located between $\kappa = e/(n \log n) \approx 0.9$ and 1.2, where n is the number of vertices and e the number of edges [FM95]. Based on these results, we created a test-set by randomly sampling soluble HCP instances from distributions of random graphs with $n = 10$ and $\kappa = 1$.

The test-set was generated using a HCP generator and solver developed and provided by Joe



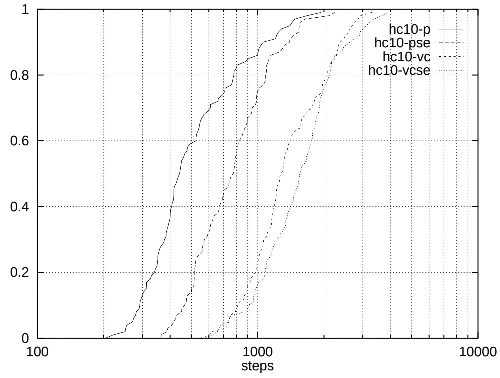


Figure 7.8: Hardness distributions (average local search cost per instance) for WalkSAT (using approx. optimal noise), when applied to different encodings of test-set `hc10`.

Culberson; insoluble instances were filtered out using the integrated systematic HCP solver, such that the test-set contains only soluble instances. This test-set (named `hc10`) was then encoded into SAT using an encoder implemented in the course of this work. This encoder allows to generate position- and vertex-based as well as sparse and compact encodings with and without symmetry elimination. For the study conducted here, we generated the test-sets described in Table 7.6; each test-set comprises 100 instances. Note that for the encodings using symmetry elimination, the number of clauses varies depending on the actual in- and out-degree of the vertex which is clamped to a fixed position in the permutation; in Table 7.6, we therefore report the average number of clauses across the test-set.

To reduce computational and overall experimental expenses, for some of our experiments we only used the minimum, median, and maximum elements (denoted as `easy`, `medium`, and `hard` instance) from the hardness distributions for WalkSAT with approx. optimal noise setting, using the position-based, sparse encoding. As we will see later, the hardness (here: average local search cost for WalkSAT with approx. optimal walk) is rather tightly correlated for the different encodings.

7.3.3 SLS Performance

In a first series of experiments, we studied the distribution of instance hardness (average local search cost) for WalkSAT over different encoding of the ten-vertex test-set (`hc10-`$*$). Since we are mainly interested in optimal performance here, we used approximately optimal noise settings for each of the encodings. These were determined based on measuring average local search cost for a particular, arbitrarily chosen instance from the original HCP test-set. Interestingly, we observed significant differences in the optimal noise for the various encodings; generally, the optimal noise was significantly higher for the compact encodings as well as for those using symmetry elimination.

Figure 7.8 shows these hardness distributions; the corresponding basic descriptive statistics

algorithm	mean	stddev	$\frac{\text{stddev}}{\text{mean}}$	median	Q_{25}	Q_{75}	Q_{10}	Q_{90}	$\frac{Q_{75}}{Q_{25}}$	$\frac{Q_{90}}{Q_{10}}$
p	572.56	339.75	0.59	435.21	351.94	694.32	289.45	1034.35	1.97	3.57
v	931.11	504.98	0.54	781.28	589.54	1148.76	386.17	1680.76	1.95	4.35
vc	1463.52	606.03	0.41	1258.44	1013.17	1904.27	800.31	2324.88	1.88	2.90
pe	826.46	364.63	0.44	753.26	524.86	989.15	452.85	1318.97	1.88	2.91
ve	935.02	433.64	0.46	883.92	616.22	1076.89	514.13	1496.82	1.75	2.91
vce	1648.54	680.20	0.41	1545.27	1113.37	1960.47	896.29	2514.87	1.76	2.81

Table 7.7: Test-set `hc10`, basic descriptive statistics of hardness distributions for different SAT-encodings, using WalkSAT (approx. optimal noise, 100 tries/instance).

are reported in Table 7.7. As we can see, the position-based, sparse encoding (`p`) is most efficient w.r.t. local search cost, followed by the vertex-based, sparse encoding (`v`); the vertex-based compact encoding (`vc`) induces significantly higher local search cost. Also, symmetry elimination generally leads to significantly increased local search cost (compare `p` *vs* `pse`, *etc.*). However, this increase is strongest for the sparse position-based encoding, and relatively moderate for the vertex-based compact encoding. This is quite plausible, since in the latter case, SLS behaviour is apparently already significantly hindered by the basic encoding, such that the additional impediments induced by symmetry elimination are less relevant to SLS behaviour. Note that the observed variation in instance hardness across the test-sets is significantly smaller than for Random-3-SAT or Random Binary CSP (*cf.* Tables 4.10, page 96 and 7.2, page 183).

Another interesting observation from the data presented in Table 7.7 is the fact that the relative variability of instance hardness across the test-set is negatively correlated with the efficiency of the encoding w.r.t. local search cost. In particular, the normalised standard deviation (stddev/mean) and the percentile rations are lower for the compact encodings than for the sparse encodings; also, these values are lower when symmetry elimination is applied. This suggests that for relatively easy problem instances, the influence of the encoding on SLS performance is much higher than for instances which are intrinsically harder — a fairly plausible assumption.

In a next step, we perform the same correlation analysis as for the Random-3-SAT and Graph Colouring test-sets in Chapter 4; only now, we don't compare different algorithms, but rather the same algorithm (WalkSAT with approx. optimal noise) applied to different encodings of the same set of HCP instances. The results of this analysis are shown in Table 7.8. As in the case of CSP (*cf.* Section 7.2), we observe a relatively strong correlation between the local search cost for different encodings of the same instance, indicating that features of the original instance, rather than those induced by the encoding, dominate SLS behaviour. Interestingly, the influence of symmetry elimination seems to be stronger for sparse than for compact encodings, as can be seen from the lower correlation coefficients for the corresponding hardness correlations (the higher noise in these correlations is caused by the encoding).

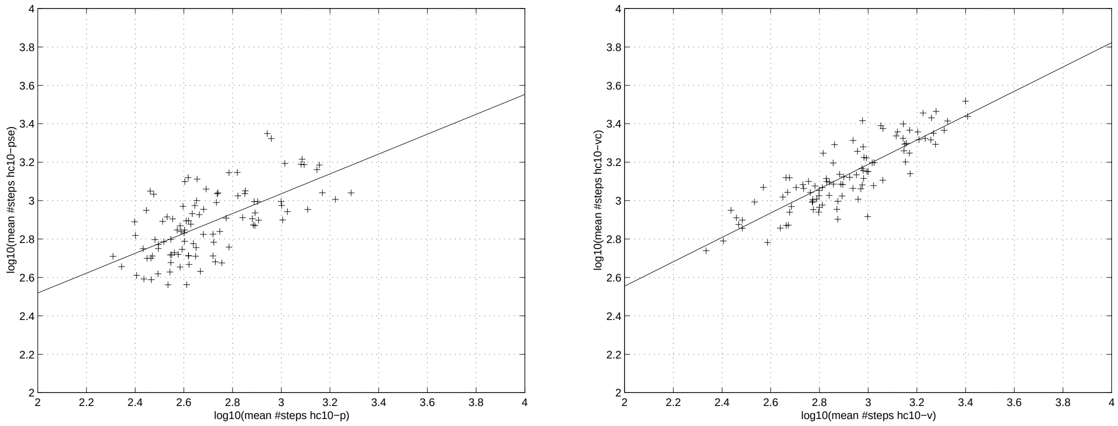


Figure 7.9: *Left:* Correlation between average local search cost for WalkSAT for test-sets `hc10-p` and `hc10-pse`, using approx. optimal noise. *Right:* Same for `hc10-v` *vs* `hc10-vc`.

encodings	r	a	b
hc10p *vs* hc10v	0.74	0.81	0.72
hc10v *vs* hc10vc	0.85	0.63	1.29
hc10p *vs* hc10pe	0.63	0.52	1.49
hc10v *vs* hc10ve	0.59	0.44	1.64
hc10vc *vs* hc10vce	0.79	0.80	0.67

Table 7.8: Test-set `hc10`, pairwise hardness correlation for WalkSAT with approx. optimal noise applied to different encodings, based on 100 tries / instance; r is the correlation coefficient, a and b are the parameters of the *lms* regression analysis.

Furthermore, the results from the regression analyses (a and b values in Table 7.8, *cf.* Chapter 4) show that the more efficient encodings (w.r.t. resulting local search cost for WalkSAT) scale worse with instance hardness (compare a values for p, v, and vc) — confirming our earlier observation for CSP that the influence of the encoding decreases with instance hardness. When applying symmetry elimination, this effect can also be observed: while generally, the local search cost tends to increase, this increase is stronger for relatively easy than for harder instances.

We also analysed the individual RLDs for different encodings of the medium hardness problem instance, using ed and ged approximations as described in Chapter 5. Since regardless of the encoding, this instance is rather easy to solve for WalkSAT, it is not surprising that the approximations using exponential distributions did not pass a standard χ^2 test (except for the vertex-based, compact encoding) because of the effect of the initial search phase on SLS behaviour for short run-times. But when using generalised exponential distributions which take into account the initial search phase (*cf.* Chapter 5), all RLD approximations pass the χ^2 test for the $\alpha = 0.05$ confidence level. This indicates that for HCP, independently from the encoding chosen, WalkSAT shows the typical behaviour as characterised in Chapter 5 for a number of different problem domains.

Generally, our results indicate that, as for CSP, sparse encodings are significantly superior w.r.t. SLS performance than compact encodings. Also, SLS performance is generally rather impeded than improved by symmetry elimination. Finally, there is a significant performance advantage for the vertex-based encoding over the position-based encoding.

7.3.4 Search Space Structure

After analysing the influence of the different encodings on local search cost, in this section, we investigate the role of some search space features (*cf.* Chapter 6) in this context. As we have seen before, SLS performance strongly depends on the number of solutions, but is also affected by other factors such as the standard deviation of the objective function and the average local minima branching.

Obviously, the number of solutions of a given HCP instance is identical for the position-based and vertex-based encodings, and it is not affected by using compact instead of sparse encodings. Symmetry elimination, however, reduces the number of solutions by a factor n (where n is the number of vertices in the given graph), as can be easily seen from the description of the encodings. At the same time, the search space is much smaller when symmetry elimination has been applied. As detailed in Section 7.3.1, this results in a net increase in the average solution density for the sparse encodings, while for the compact encoding, the net solution density descreases.

Table 7.9 shows the number of solutions and solution density for the easy, medium, and hard instances from the `hcp10` test-set for vertex-based, sparse encodings with and without symmetry elimination as well as the corresponding average local search cost for WalkSAT. Clearly, despite the higher solution density, the average local search cost increases when

instance	symmetry elim.	solutions	solution density	avg. *lsc*
hc10-p/easy	no	60	$4.73 \cdot 10^{-29}$	174.45
hc10-pse/easy	yes	6	$2.48 \cdot 10^{-24}$	529.33
hc10-p/medium	no	20	$1.58 \cdot 10^{-29}$	361.78
hc10-pse/medium	yes	2	$8.27 \cdot 10^{-25}$	740.10
hc10-p/hard	no	20	$1.58 \cdot 10^{-29}$	917.40
hc10-pse/hard	yes	2	$8.27 \cdot 10^{-25}$	935.06

Table 7.9: Test-sets `hc10-p` and `hc10-pse`; number of solutions, solution density and average local search cost for WalkSAT (using approx. optimal noise, based on 1,000 tries / instance) fo easy, medium, and hard problem instance.

instance	norm. *sdnclu*	norm. *blmin*	avg. *lsc*
hc10-p/medium	0.0503	0.057	361.78
hc10-v/medium	0.0502	0.056	476.30
hc10-vc/medium	0.0017	0.181	1,255.02
hc10-pe/medium	0.0561	0.063	740.10
hc10-ve/medium	0.0561	0.063	858.06
hc10-vce/medium	0.0021	0.196	1,776.35

Table 7.10: Medium hardness instance from test-set `hc10`, normalised sdnclu and blmin measures and average local search cost (w.r.t. WalkSAT, using approx. optimal noise, $1,000$ tries) for different encodings.

using symmetry elimination — a result, which is not consistent with our intuition about the dominant influence of the solution density on SLS performance. However, as observed before (*cf.* Section 7.3.3), the performance loss is most drastic for the easy instance, while for the hard instance the detrimental effect of symmetry elimination is very small. Similar observations can be made for the compact encodings; there, however, because symmetry elimination decreases the solution density, the increase in local search cost matches our intuition.

Next, we analysed the two other measures of search space structure discussed in Chapter 6: the standard deviation of the objective function (*sdnclu*) and the average local minima branching (*blmin*). As in Section 7.2, we normalise these measures to enhance their comparability between the different encodings. Table 7.10 shows the normalised *sdnclu* and average *blmin* values, as well as the average local search cost (w.r.t. WalkSAT, using approx. optimal noise) for the medium instance of test-set `hc10` under all six encodings discussed before. The results confirm our original intuition regarding the influence of these search space features on SLS performance: the local search cost seems to be negatively correlated to the normalised sdnclu value, and positively correlated to the normalised average local minima branching. In the analysis of SAT-encodings for the given HCP instance, this can be clearly seen from the results reported in Table 7.10, when comparing the corresponding values between the sparse and compact encodings, as well as between encodings with and without symmetry elimination. In the latter case, the effect of the increased local minima branching seems to outweigh a possible beneficial effect of a slightly increased *sndclu* value. Applying the same analysis to the easy and hard problem instances from the same test-set confirms these results.

In the light of these results, WalkSAT's performance differences on the various encodings can be explained in the following way: As for CSP, compact encodings induce search spaces characterised by extremely flat and feature-less objective functions and relatively highly branched local minima regions — features, which impede stochastic local search. A similar effect, but considerably weaker, is generally induced by symmetry elimination. This demonstrates that analysing relatively simple features of the search space can be useful for assessing different strategies for SAT-encoding instances from other problem domains, such as HCP.

7.4 Related Work

SAT-encodings of combinatorial problems are frequently used in the context of $\mathcal{NP}$-completeness proofs in complexity theory [GJ79]. A systematic approach to transforming hard combinatorial problems into SAT has been proposed by Stamm-Wilbrandt [SW93], who also gives space-efficient SAT-encodings for many $\mathcal{NP}$-complete problems from different domains. Iwama and Miyzaki introduced the concept of SAT-variable complexity and discussed different encodings of hard combinatorial problems, including the Hamilton Circuit Problem, which are optimised for a minimal number of propositional variables [IM94].

In contrast, the influence of different encoding strategies on the performance of SAT solvers is mainly unexplored. Although from the beginning of the development of modern SLS algorithms for SAT, these have been tested on SAT-encoded problems from domains like Graph Colouring or Blocks World Planning [SLM92, KS92], the influence of the encodings, to our best knowledge, has rarely been systematically investigated. Only recently, when it became clear that hard Blocks World Planning instances can be effectively solved using SLS algorithms for SAT, the issue of the influence of SAT-encodings on SLS performance has been further studied. Kautz and Selman use various techniques for optimising SAT-encodings for various well-known planning domains (including Blocks World) to achieve maximal SLS performance [KS96, KMS96]. The first systematic study of SAT-encodings and their impact on the average solution cost for state-of-the-art SAT algorithms we are aware of, is the study on automatic SAT-compilation for planning problems by Ernst, Millstein, and Weld [EMW97]. Different from [KS96], who use hand-crafted encodings, [EMW97] systematically evaluate a number of general encoding strategies which have been built into a fully automated SAT-encoder for planning problems from various domains. One of their results states that compact encodings of planning instances are typically very difficult for SLS-based SAT algorithms — an observation which is confirmed by our results for CSP and HCP.

Random binary CSPs have been studied extensively by various researchers, especially in the context of phase transition phenomena and their impact on the performance of CSP algorithms [Smi94, SD96, Pro96]. The sparse SAT-encoding of CSP has been introduced by de Kleer [dK89], who used it in the context of a comparative study of CSP and assumption-based truth maintainance system (ATMS) concepts and techniques. The compact CSP encoding is inspired by Iwama's and Miyazaki's work on SAT-variable complexity (see also below). Regarding SLS algorithms for CSP, one of the first and most influential algorithms is the Min-Conflicts Heuristic (MCH), which has been introduced by Minton *et al.* in 1992 [MJPL92]. Recently, the basic algorithm, which is conceptually closely related to WalkSAT [SKC94], has been extended with random walk and tabu search techniques by Stützle *et al.* [SSS97]. Analogous to the corresponding SAT algorithms, these extensions show a superior performance compared to the original MCH. The best currently known algorithm for the subclass of CSP studied here is the tabu search algorithm by Galinier and Hao [GH97], which is conceptually closely related to GSAT, as it is based on an analogous neighbourhood relation. Despite the close relationship between SLS algorithms for SAT and CSP, we are not aware of any previous comparative analysis of their performance on an identical set of problem instances. The problem instances, as well as the implementations of the CSP algorithms used in the context of this work have been kindly provided by Thomas Stützle (TU Darmstadt, Germany).

The Hamilton Circuit Problem in random graphs has been studied by Cheeseman *et al.* [CKT91] as well as by Frank and Martel [FM95]. Both studies focus on phase transition phenomena and their influence on the performance of complete HCP algorithms. Although the solubility phase transition does not as nicely coincide with the peak in difficulty as for other problem domains, such as Random-3-SAT, we used the results from [FM95] for generating the test-sets used in the context of this work. The actual test-sets were created using an HCP instance generator kindly provided by Joseph Culberson (University of Alberta, Canada). SAT-encodings of HCP were previously discussed in [IM94], who developed the compact, vertex-based SAT-encoding with symmetry elimination in the context of minimising the number of propositional variables required for translating hard combinatorial problems to SAT. In [Hoo96c, Hoo96b], we analysed and characterised the performance of GSAT and GWSAT on sets of randomly generated HCP instances using the compact, vertex-based encoding with symmetry elimination, showing that these problem instances are exceptionally difficult for both algorithms. However, in [Hoo96b], we also developed an HCP-specific SLS algorithm (GHC) which is conceptionally closely related to the Min-Conflicts Heuristic as well as to GWSAT and shows significantly improved performance on the given test-sets.

7.5 Conclusions

In this chapter, we studied the impact of encoding strategies on search space structure and SLS performance. Based on two case studies, covering Constraint Satisfaction and Hamilton Circuit Problems, we exemplified how some of the methods developed in the previous chapters can be used to investigate how different SAT encodings induce certain search space features which, in turn, affect the behaviour of SLS algorithms.

At the beginning of this chapter, in Section 7.1, we formulated three questions regarding the effectivity of specific encoding strategies and the competitiveness of the generic problem solving approach in general. Based on the results of our case studies for CSP and HCP, we will give some tentative answers here — however, as the underlying empirical evidence is very limited in its scope, these answers should be rather understood as testable hypotheses.

Does it pay off to minimise the number of propositional variables, i.e., are compact encodings which achieve small search spaces preferable to sparse encodings?

No. According to our results for CSP and HCP, while compact encodings have a significantly higher solution density, they induce search spaces which are rather flat and feature-less (characterised by low *sdnclu* values). At the same time, their local minima are significantly more branched, which makes escaping from these minima considerably more difficult for SLS algorithms like WalkSAT. In contrast, sparse encodings induce rugged search spaces with lowly branched local minima — features which generally increase the performance of SLS algorithms. So, in a nutshell, it seems to be much more advisable to use sparse rather than compact encodings, as the former tend to induce vast, but well-structured search spaces which are much easier to search than the search spaces produced by compact encodings, which are much smaller, but provide far less guidance for local search.

Is it advisable to reduce the number of propositional variables required for a given encoding by eliminating symmetric solutions from the original problem formulation?

Apparently not. Although, for the given example (HCP), when using sparse encodings, symmetry elimination increases the solution density, it seems to have detrimental effects on SLS performance which are similar to those induced by compact encodings, although the impact on SLS performance is less drastic. However, since we did not study the scaling of these effects with problem size, it cannot be ruled out that, for bigger instances, a different situation can be found. Preliminary experimentation indicates that for the problem class studied here, the detrimental effect of symmetry elimination can be also observed for larger problem sizes.

Is the generic problem solving approach competitive with applying SLS algorithms to the un-encoded problem instances?

Our results for Random Binary CSP instances indicate that this is the case. Although, when comparing the number of steps per solution for state-of-the-art algorithms, we observed a slight advantage for the direct CSP approach, this is easily outweighed by other advantages of the generic problem solving approach, such as the availability of extremely efficient implementations and its more general applicability. However, again it is not clear how the observed performance difference scales with problem size. Nevertheless, our result is good news regarding the generic problem solving approach based on efficient SAT-encodings and modern SLS algorithms for SAT, especially, since the same, simple encoding strategies and SLS algorithms could be successfully used to competitively solve a number of hard combinatorial problems from different domains.

Epilogue

In this thesis we have studied various aspects of stochastic local search (SLS), an algorithmic paradigm for solving hard combinatorial problems which has become increasingly popular in Artificial Intelligence and Operations Research over the past few years. Our work focussed on modelling SLS algorithms, empirically evaluating their performance, characterising and improving their behaviour, and understanding the factors which influence their efficiency. We studied these issues for the SAT problem in propositional logic as our primary application domain. SAT has the advantage of being conceptually very simple, which facilitates the design, implementation, and presentation of algorithms as well as their analysis. However, most of the methodology generalises easily to other combinatorial problems like CSP. The main contributions of this work can be summarised as follows.

- We introduced the *GLSM model*, a novel formalism for representing SLS algorithms which is based on the concept of a clear separation between search strategies and search control (Chapter 3). The GLSM model allows the adequate representation of many modern SLS algorithms for SAT (Chapter 4) and provides a basis for analysing and improving these (Chapter 5).

- We developed an adequate methodology for empirically analysing the behaviour of Las Vegas algorithms (a more general class of algorithms than SLS) which avoids weaknesses of established approaches (Chapter 2). We used this methodology for studying SLS algorithms for SAT and characterising their behaviour (Chapters 4 and 5).

- We studied the performance of a number of popular and very powerful SLS algorithms for SAT across a broad range of problem domains (Chapters 4 and 7). We found that the relative performance of modern SLS algorithms varies with the problem domain such that there is no single best algorithm. We could also show that there is a tight correlation between the performance of different SLS algorithms within the domains, which suggests that SLS performance generally depends on the same features of the underlying search spaces.

- We proved a number of results concerning the asymptotic behaviour (PAC property) of various modern SLS algorithms for arbitrary long runs of the corresponding pure strategies. We could show that some of the most powerful SLS algorithms for SAT suffer from essential incompleteness and demonstrated how to overcome this defect in practice (Chapter 5).

- We developed a functional characterisation of SLS behaviour, showing that for optimal and larger-than-optimal noise parameter settings, the RTDs for modern SLS algorithms, when applied to hard problem instances from different domains, can be well approximated using exponential distributions (EPAC behaviour, *cf.* Chapter 5). This result has a number of interesting consequences regarding the parameterisation and parallelisation of SLS algorithms, and it suggests a novel interpretation of SLS behaviour as random picking in a significantly smaller "virtual" search space.

- We analysed various aspects of search space structure and their influence on SLS performance (Chapter 6). We confirmed and refined earlier results regarding the dominant role of the number of solutions and investigated other search space features such as the variability of the objective function and the topology of local minima regions. Our results show that local minima regions resemble systems of narrow canyons rather than large basins; the branching of these structures is very low, especially for SAT-encoded problems from other domains. This explains the effectiveness of simple escape strategies like random walk.

- We investigated the influence of different SAT-encoding strategies on search space structure and SLS performance for test-sets of hard Constraint Satisfaction and Hamilton Circuit Problems (Chapter 7). Confirming earlier results for planning domains, our results show that compact encodings induce relatively small search spaces which are nevertheless very difficult for search for SLS algorithms. Similarly, eliminating symmetric solutions of the Hamilton Circuit Problem seems to have detrimental effects on SLS performance. We show how the techniques and methods developed earlier in this thesis can be used to analyse and identify and some of the search space features which are responsible for these effects. Furthermore, we show that for solving hard Random Binary CSP instances, using efficient SAT-encodings and state-of-the-art SLS algorithms for SAT is competitive to the best SLS algorithms for CSP.

Open Questions and Future Work

Many contributions of this thesis, including methodological aspects as well as the results of empirical and theoretical investigations, are interesting on their own. However, they also give rise to a number of questions, shed new light on some open problems, and suggest many routes for further research. In the following, we will briefly address some of these issues.

- Our empirical studies of SLS algorithms for SAT were mainly limited to algorithms which could be represented using simple instances of the basic GLSM model. Many of the advanced features and extensions of the GLSM model have not been used in this context; their potential for SAT and other problem domains has to be further investigated and assessed. In the light of our results for SAT, the heterogenous cooperative models, possibly including learning schemes and evolutionary population dynamics, appear to be particularly attractive.

- We have shown that the GLSM model can be used to improve some of the best currently known SLS algorithms for SAT. However, the corresponding GLSMs are still very simple; it remains to be seen, whether more complex combinations of pure SLS strategies show substantially improved performance and / or robustness. Further research on this issue should probably not be exclusively focussed on combinatorial decision problems like SAT, but also include classical optimisation problems like TSP or scheduling.

- While we could prove approximate completeness for GWSAT and essential incompleteness for WalkSAT+tabu, Novelty, and R-Novelty, the question whether WalkSAT is approximately complete remains open. Our empirical results give no indication of essential incompleness, therefore we suspect that WalkSAT might be approximately complete. Furthermore, we did not prove essential incompleteness for GSAT and GSAT+tabu; but we believe that the proofs are not difficult. As we have shown, essential incompleteness leads to poor performance and robustness of SLS algorithms in practice. Therefore, approximate completeness and the avoidance of stagnation behaviour in practice should be considered as important design objectives for future development of SLS algorithms.

- We developed functional characterisations of the run-time behaviour of a number of SLS algorithms for SAT when using optimal or larger-than-optimal noise settings. Since the same EPAC behaviour was found for all the algorithms and all hard problems studied here, we suspect that this type of behaviour is characteristic for SLS algorithms when applied to hard combinatorial problems. This hypothesis is backed up by preliminary results on modelling the behaviour of SLS algorithms for Constraint Satisfaction Problems. We did not model SLS behaviour for lower-than-optimal noise; but we believe that the corresponding RTDs can be approximated using combinations of the generalised exponential distribution family introduced in Chapter 5.

- As discussed in Chapter 5, the observed EPAC behaviour suggests a novel interpretation of SLS behaviour as random picking in a reduced "virtual" search space. The size of this search space is directly related to the average local search cost for solving the corresponding problem instance. While in Chapter 6, we found some search space features which are to some extent correlated to the average local search cost, we could not find any structural search space features which directly correspond to the virtual search space. Generally, from our perspective the attempt to find explanations for the observed SLS behaviour based on the simple mathematical models developed here seems to be one of the most exciting and promising issues for further research.

- While we studied search space structure and its influence on SLS performance as well as the effect of different SAT-encodings on search space structure, many questions in this context remain unanswered. Our results suggest that features different from the ones investigated here play an important role; further work should try to identify and characterise these. Furthermore, the analyses of the search space structural features induced by different encodings and their impact on SLS peformance presented in Chapter 7 should be extended to different problem sizes and other domains. We are convinced that further insights into these dependencies will be crucial for the further development of SLS algorithms, encoding strategies, and the general applicability of the generic problem solving approach discussed in Chapter 1.

- Finally, we are convinced that most of the methodology developed in the context of this thesis can equally well be used for studying SLS algorithms for optimisation problems. To some extent, the required generalisations have already been introduced in this work (*cf.* Chapter 2). Consequently, conducting investigations analogous to the ones presented here for optimisation problems like MAXSAT or MAXCSP should not be difficult. Since SLS algorithms for decision problems like SAT implicitly solve the corresponding optimisation problem, we expect that this line of future work will provide interesting insights into SLS behaviour in general.

Building on previous work on stochastic local search, our goal in this work was to enhance the general understanding of SLS algorithms and their behaviour, to evaluate their performance, and, ultimately, to improve existing algorithms. While, hopefully, we reached this goal to some extent, many fundamental questions regarding SLS algorithms and their applications remain unanswered. From our point of view, the most important of these are the precise understanding of the factors which determine the efficiency of SLS algorithms and a general assessment of the role of SLS algorithms in the context of the generic problem solving approach discussed in Chapter 1 of this thesis. While we cannot provide the ultimate solutions to these important problems, we hope that the contributions of this thesis might be regarded as a further step towards finding satisfying solutions to these problems and will prove to be useful for future SLS research.

Bibliography

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid Automata: an Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems I*, volume 736 of *Lecture Notes in Computer Sciene*, pages 209–229. Springer-Verlag, 1993.

[AO96] Ravindra K. Ahuja and James B. Orlin. Use of Representative Operation Counts in Computational Testing of Algorithms. *INFORMS Journal on Computing*, 8(3):318–330, 1996.

[Bab79] Laszlo Babei. Monte Carlo Algorithms in Graph Isomorphism Testing. Technical Report DMS 79–10, Université de Montréal, 1979.

[Bäc94] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice.* PhD thesis, Universität Dortmund, Fachbereich Informatik, February 1994.

[BAH+94] A. Beringer, G. Aschemann, H.H. Hoos, M. Metzger, and A. Weiß. GSAT versus Simulated Annealing. In A. G. Cohn, editor, *Proceedings of ECAI'94*, pages 130–134. John Wiley & Sons, 1994.

[Bea90] J.E. Beasley. OR-Library: distributing test-problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[Ben96] H. Bennaceur. The Satisfiability Problem Regarded as a Constraint Satisfaction Problem. In Wolfgang Wahlster, editor, *Proceedings of ECAI'96*, pages 155–160. John Wiley & Sons, Chichester, 1996.

[Bib80] Wolfgang Bibel. "Intellektik" statt "KI" — ein ernstgemeinter Vorschlag. *Rundbrief der Fachgruppe Künstliche Intelligenz in der Gesellschaft für Informatik*, 22:15–16, December 1980.

[Bib92] Wolfgang Bibel. *Deduktion: Automatisierung der Logik.* Vieweg, 1992.

[Bib93] Wolfgang Bibel. *Wissensrepräsentation und Inferenz.* Vieweg, 1993.

[BKB92] M. Buro and H. Kleine-Büning. Report on a SAT competition. Technical Report 110, Dept. of Mathematics and Informatics, University of Paderborn, Germany, 1992.

[BM98] Justin A. Boyan and Andrew W. Moore. Learning Evaluation Functions for Global Optimization and Boolean Satisfiability. In *Proceedings of AAAI'98*, pages 3–10. AAAI Press / The MIT Press, Menlo Park, CA, 1998.

[BP81] R.E. Barlow and F. Proschan. *Statistical Theory of Reliability and Life Testing.* Holt, Reinhart and Winston, Inc., 1981.

[BP96] R. Battiti and M. Protasi. Solving MAX-SAT with non-oblivious functions and history-based heuristics. In *DIMACS Workshop on Satisfiability Problem: Theory and Applications*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1996.

[CA93] J.M. Crawford and L.D. Auton. Experimental Results on the Crossover Point in Satisfiability Problems. In *Proceedings of AAAI'93*, pages 21–27. MIT Press, 1993.

[CA96] James M. Crawford and Larry D. Auton. Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence*, 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, *Proceedings of CP'96*, volume 1118 of *Lecture Notes in Computer Sciene*, pages 119–133. Springer Verlag, 1996.

[Çin75] Erhan Çinlar. *Introduction to stochastic processes.* Prentice-Hall, Englewood Cliffs, NJ, 1975.

[CKT91] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proceedings of IJCAI'91*, pages 331–337, 1991.

[Col40] H.C.C. Colles, editor. *Grove's Dictionary of Music and Musicians, supplementary volume.* The MacMillan Company, New York, 1940.

[Coo71] Stephen A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–156. Shaker Heights, Ohio, 1971.

[Cul92] Joseph Culberson. Iterated Greedy Graph Colouring and the Difficulty Landscape. Technical Report TR92–07, University of Alberta, June 1992.

[DABC93] O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. In *2nd DIMACS Implementation Challenge*, 1993.

[DG84] W. F. Dowling and J. H. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[dK89] J. de Kleer. A Comparison of ATMS and CSP Techniques. In *Proceedings of IJCAI'89*. Morgan Kaufmann, 1989.

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–42, 1996.

[EMW97] M. D. Ernst, T. D. Millstein, and D. S. Weld. Automatic SAT-Compilation of Planning Problems. In *Proceedings of IJCAI'97*, 1997.

[FCS97] Jeremy Frank, Peter Cheeseman, and John Stutz. When Gravity Fails: Local Search Topology. *(Electronic) Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[FM95] Jeremy Frank and Charles U. Martel. Phase Transitions in the Properties of Random Graphs. In *Workshop on Studying and Solving Really Hard Problems*, in association with CP'95, 1995.

[Fra94] Jeremy Frank. A study of genetic algorithms to find approximate solutions hard 3CNF problems. In *Golden West International Conference on Artificial Intelligence*, 1994.

[Fra97a] Jeremy Frank. Weighting for godot: Learning heuristics for GSAT. Unpublished Manuscript, from WWW, February 1997.

[Fra97b] Jeremy D. Frank. *Local Search for NP-Hard Problems.* PhD thesis, University of California, Davis, California, 1997.

[FRV97] Daniel Frost, Irina Rish, and Lluís Vila. Summarizing CSP Hardness with Continuous Probability Distributions. In *Proceedings of AAAI'97*, pages 327–333, 1997.

[Fuk97] Alex S. Fukunaga. Variable-selection heuristics in local search for SAT. In *Proceedings of IJCAI-97*, pages 275–280, 1997.

[GH97] Philippe Galinier and Jin-Kao Hao. Tabu Search For Maximal Constraint Satisfaction Problems. In *Proceedings of CP'97*, number 1330 in LNCS, pages 196–208. Springer Verlag, 1997.

[Gin93] Matthew Ginsberg. *Essentials of Artificial Intelligence.* Morgan Kauffmann, San Mateo, CA, 1993.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA, 1979.

[Glo89] F. Glover. Tabu Search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Glo90] F. Glover. Tabu Search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[GMPW97] Ian P. Gent, Ewan MacIntyre, Patrick Prosser, and Toby Walsh. The Scaling of Search Cost. In *Proceedings of AAAI'97*, 1997.

[Gol89] David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company Inc., 1989.

[GS97a] Carla P. Gomes and Bart Selman. Algorithm Portfolio Design: Theory vs. Practice. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 238–245. Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[GS97b] Carla P. Gomes and Bart Selman. Problem Structure in the Presence of Perturbations. In *Proceedings of AAAI'97*, pages 221–226, 1997.

[GSC97] Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-tailed distributions in combinatorial search. In *Proceedings of CP'97*, LNCS, pages 121–135. Springer Verlag, 1997.

[GSK98] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of AAAI'98*, pages 431–437. AAAI Press / The MIT Press, Menlo Park, CA, 1998.

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

[Gu94] J. Gu. Global optimization for satisfiability (SAT) problem. *IEEE Transactions on Data and Knowledge Engineering*, 6(3):361–381, 1994.

[GV98] Jens Gottlieb and Nico Voss. Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN V)*, volume 1498 of *Lecture Notes in Computer Sciene*, pages 755–764. Springer-Verlag, 1998.

[GW93a] Ian P. Gent and Toby Walsh. An empirical analysis of search in gsat. *(Electronic) Journal of Artificial Intelligence Research*, 1:47–59, 1993.

[GW93b] Ian P. Gent and Toby Walsh. Towards an understanding of hill–climbing procedures for SAT. In *Proceedings of AAAI'93*, pages 28–33. MIT press, 1993.

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

[Har78] Michael A. Harrison. *Introduction to Formal Language Theory*. Addison Wesley Publishing Company, 1978.

[HD87] A. Hertz and D. DeWerra. Using tabu search techniques for graph coloring. *Computing*, pages 345–351, 1987.

[Hen96] Thomas A. Henzinger. The Theory of Hybrid Automata. In *Proceedings of the 11th annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 278–292, 1996.

[HHSW94] Steffen Hölldobler, Holger H. Hoos, Antje Strohmaier, and Andreas Weiß. The GSAT/SA-Familiy – Relating greedy satisifability testing to simulated annealing. Technical Report AIDA-94-17, TH Darmstadt, 1994.

[HJ90] Pierre Hansen and Brigitte Jaumard. Algorithms for the Maximum Satisfiability Problem. *Computing*, 44:279–303, 1990.

[HM97] Y. Hamada and D. Merceron. Reconfigurable architectures: A new vision for optimization problems. In *Proceedings of CP'97*, LNCS, pages 209–221. Springer Verlag, 1997.

[Hog96] Tad Hogg. Refining the Phase Transition in Combinatorial Search. *Artificial Intelligence*, 81:127–154, 1996.

[Hol75] John H. Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[Hoo94] J.N. Hooker. Needed: An Empirical Science of Algorithms. *Operations Research*, 42(2):201–212, 1994.

[Hoo96a] J.N. Hooker. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics*, pages 33–42, 1996.

[Hoo96b] Holger H. Hoos. Aussagenlogische SAT-Verfahren und ihre Anwendung bei der Lösung des HC-Problems in gerichteten Graphen. Master's thesis, Technische Universität Darmstadt, Germany, 1996.

[Hoo96c] Holger H. Hoos. Solving hard combinatorial problems with GSAT — a case study. In *KI-96: Advances in Artifial Intelligence*, volume 1137 of *LNAI*, pages 107–119. Springer Verlag, 1996.

[HS96] Holger Hoos and Thomas Stützle. A Characterization of GSAT's Performance on a Class of Hard Structured Problems. Technical Report AIDA–96–20, FG Intellektik, TH Darmstadt, December 1996.

[HS98a] Holger H. Hoos and Thomas Stützle. Evaluating Las Vegas Algorithms — Pitfalls and Remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[HS98b] Holger H. Hoos and Thomas Stützle. Some surprising regularities in the behaviour of stochastic local search (poster abstract). In *Proceedings of CP'98 (to appear)*, 1998.

[HW94a] Tad Hogg and Colin P. Williams. Expected Gains from Parallelizing Constraint Solving for Hard Problems. In *Proceedings of AAAI'94*, pages 331–336, 1994.

[HW94b] Tad Hogg and Colin P. Williams. The Hardest Constraint Problems: A Double Phase Transition. *Artificial Intelligence*, 69:357–377, 1994.

[IM94] Kazuo Iwama and Shuichi Miyazaki. SAT-variable complexity of hard combinatorial problems. In *Proceedings of the World Computer Congress of the IFIP*, pages 253–258 (Volume 1). Elsevier Science B.V., North Holland, 1994.

[Jáj92] Jospeh Jájá. *An Introduction to Parallel Algorithms.* Addison-Wesley Publishing Company, 1992.

[JKS95] Yuejun Jiang, Henry Kautz, and Bart Selman. Solving problems with hard and soft constraints using a stochastic algorithm for max-sat. Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Reasearch available at http://www.cirl.uoregon.edu/aior/, June 1995.

[JM97] D.S. Johnson and L.A. McGeoch. The Travelling Salesman Problem: A Case Study in Local Optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization.* John Wiley & Sons, 1997.

[Joh90] David S. Johnson. Local Optimization and the Travelling Salesman Problem. In *Proc 17th Colloquium on Automata, Languages, and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 446–461. Springer-Verlag, 1990.

[KGC77] A. Kaufmann, D. Grouchko, and C. Cruon. *Mathematical Models for the Study of Reliability of Systems.* Academic Press, Inc., 1977.

[KJV83] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[KM90] Yves Kodratoff and Ryszard Michalski, editors. *Machine Learning - An Artificial Intelligence Approach, Volume III.* Morgan Kaufmann Publishers, Inc., 1990.

[KMS96] Henry Kautz, David McAllester, and Bart Selman. Encoding Plans in Propositional Logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.

[Kre74] Ernst Krenek. *Horizons Circled – Reflections on my Music.* University of California Press, Berkeley, 1974.

[Kri89] E. V. Krishnamurthy. *Parallel Processing: Principles and Practice.* Addison-Wesley Publising Company, 1989.

[KS92] Henry Kautz and Bart Selman. Planning as Satisfiability. In *Proceedings of ECAI'92*, Vienna, Austria, 1992.

[KS94] Scott Kirkpatrick and Bart Selman. Critical Behavior in the Satisfiability of Random Boolean Expressions. *Science*, 264:1297–1301, 1994.

[KS96] Henry Kautz and Bart Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of AAAI'96*, volume 2, pages 1194–1201. MIT Press, 1996.

[Kwa96] Alvin Kwan. The Validity of Normality Assumption in CSP Research. In *Proceedings of the Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, LNCS. Springer Verlag, 1996.

[LA97] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of CP'97*, LNCS, pages 341–355. Springer Verlag, 1997.

[LE93] Michael Luby and Wolfgang Ertel. Optimal Parallelization of Las Vegas Algorithms. Technical Report TR-93-041, International Computer Science Institute, Berkeley, CA, 1993.

[LK73] S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Travelling Salesman Problem. *Operations Research*, 21:498–516, 1973.

[LP84] Rudolf Lidl and Günther Pilz. *Applied abstract algebra*. Springer-Verlag, 1984.

[LSZ93] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal Speedup of Las Vegas Algorithms. *Information Processing Letters*, 47:173–180, 1993.

[Mac92] Alan K. Mackworth. *Constraint Satisfaction*, pages 285–293. Encyclopedia of Artificial Intelligence, second edition. Wiley, New York, 1992.

[McG96] Catherine C. McGeoch. Toward an Experimental Method for Algorithm Simulation. *INFORMS Journal On Computing*, 8(1):1–15, 1996.

[Min96] Steven Minton. Automatically Configuring Constraint Satisfaction Programs: A Case Study. *Constraints*, 1(1), 1996.

[Mit97] Tom M. Mitchell, editor. *Machine Learning*. McGraw-Hill, 1997.

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

[MJPL92] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 52:161–205, 1992.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From Timed to Hybrid Systems. In *Proceeding of the REX Workshop "Real Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Sciene*, pages 447–484. Springer-Verlag, 1992.

[MOF91] Olivier Martin, Steve W. Otto, and Edward W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5(3):299–326, 1991.

[Mor93] Paul Morris. The Breakout Method for Escaping from Local Minima. In *Proceedings of AAAI'93*, pages 40–45, 1993.

[MSG97] Bertrand Mazure, Lakhdar Sais, and Éric Grégoire. Tabu Search for SAT. In *Proceedings of AAAI'97*, pages 281–285, 1997.

[MSK97] David McAllester, Bart Selman, and Henry Kautz. Evidence for Invariants in Local Search. In *Proceedings of AAAI'97*, pages 321–326, 1997.

[MSL92] David Mitchell, Bart Selman, and Hector Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings of AAAI'92*, pages 459–465. MIT Press, 1992.

[Nil80] Nils J. Nilsson. *Principals of Artificial Intelligence*. Morgan Kauffmann, San Mateo, CA, 1980.

[NT89] Kumpati S. Narendra and Mandayam A.L. Thathachar. *Learning Automata: an introduction*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[Par97] Andrew J. Parkes. Clustering at the Phase Transition. In *Proceedings of AAAI'97*, pages 340–345, 1997.

[Pea84] Judea Pearl. *Heuristics*. Addison-Wesley, Reading, MA, 1984.

[PMG98] David Poole, Alan Mackworth, and Randy Goebel. *Computational Iintelligence: a logical approach*. Oxford University Press, New York, 1998.

[Pro96] Patrick Prosser. An Empirical Study of Phase Transitions in Binary Constraint Satisfaction Problems. *Artificial Intelligence*, 81:81–109, 1996.

[PW96] Andrew J. Parkes and Joachim P. Walser. Tuning Local Search for Satisfiability Testing. In *Proceedings of AAAI'96*, volume 1, pages 356–362. MIT Press, 1996.

[Rec73] I. Rechenberg. *Evolutionsstrategie — Optimierung technischer Systeme nach Prinzipien der biologischen Information*. Fromman Verlag, Freiburg, 1973.

[Rei90] Karl Rüdiger Reischuk. *Einführung in die Komplexitätstheorie*. Teubner Verlag, Stuttgart, 1990.

[Rei91] G. Reinelt. TSPLIB — A Traveling Salesman Problem Library. *ORSA Journal On Computing*, 3:376–384, 1991.

[RF97] Irina Rish and Daniel Frost. Statistical Analysis of Backtracking on Inconsistent CSPs. In *Proceedings of CP'97*, LNCS, pages 150–163. Springer Verlag, 1997.

[RN95] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[Roh76] V.K. Rohatgi. *An Introduction to Probability Theory and Mathematical Statistics*. John Wiley & Sons, 1976.

[RS97] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*. Springer-Verlag, 1997.

[Sch81] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, 1981.

[SD96] Barbara M. Smith and Martin E. Dyer. Locating the Phase Transition in Binary Constraint Satisfaction Problems. *Artificial Intelligence*, 81:155–181, 1996.

[Sha92] S. C. Shapiro, editor. *Encyclopedia of Artificial Intelligence, second edition*. Wiley, New York, 1992.

[Sho93] R. Shonkwiler. Parallel Genetic Algorithms. In *Proceedings of ICGA'93*, 1993.

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

[SKC93] Bart Selman, Henry A. Kautz, and Bram Cohen. Local Search Strategies for Satisfiability Testing. In M. Trick and D.S. Johnson, editors, *Second DIMACS Challenge on Cliques, Coloring, and Satisfiability.*, October 1993.

[SKC94] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise Strategies for Improving Local Search. In *Proceedings of AAAI'94*, pages 337–343. MIT Press, 1994.

[SLM92] Bart Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of AAAI'92*, pages 440–446. MIT Press, 1992.

[Smi94] Barbara M. Smith. Phase Transitions and the Mushy Region in Constraint Satisfaction Problems. In *Proceedings of ECAI'94*, pages 100–104, 1994.

[Spe93] W.M. Spears. Simulated Annealing for hard satisfiability problems. Technical report, Naval Research Laboratory, Washington D.C., 1993.

[SSS97] Olaf Steinmann, Antje Strohmaier, and Thomas Stützle. Tabu Search vs. Random Walk. In *Advances in Artificial Intelligence (KI-97)*, volume 1303 of *LNAI*, pages 337–348. Springer Verlag, 1997.

[Ste96] Olaf Steinmann. Kombinatorische Probleme, Optimierung und Parallelisierung: Eine Experimentelle Analyse von Multi-Flip Netzwerken. Master's thesis, Technische Universität Darmstadt, Germany, 1996.

[SW93] Hermann Stamm-Wilbrandt. Programming in propositional logic or reductions: Back to the roots (satisfiability). Technical report, Institut für Informatik III, Universität Bonn, Germany, 1993.

[Tai91] Éric D. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.

[Tai94] Éric D. Taillard. Parallel Tabu Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[Yok97] Makoto Yokoo. Why Adding More Constraints Makes a Problem Easier for Hill-Climbing Algorithms: Analyzing Landscapes of CSPs. In *Proceedings of CP'97*, number 1330 in LNCS, pages 357–370. Springer Verlag, 1997.

List of Symbols / Abbreviations

$\#S$	cardinality (number of elements) of set S
π	generally used for denoting problem instances; *also:* direct search transition function (GLSM model)
Π	generally used for denoting a problem class
a, a_i	search space positions, *for SAT:* propositional assignments
AIS	the All-Interval-Series Problem
AIS(n)	instance of the All-Interval-Series Problem with length n
blmin	branching of local minima states
BPLAT	plateau border state
BWP	Blocks World Planning Problem
$draw$	random selection function, selects element from given set according to uniform random distribution
D, D', D''	generally used for denoting probability distributions
$\mathcal{D}(\mathcal{X})$	the set of all probability distributions over set X
CSP	Constraint Satisfaction Problem (usually over finite, discrete domains)
cvr	clauses per variable ratio (for Random-3-SAT)
e, e'	probabilistic events, *i.e.* elements of the domain of a probability distributions
EPAC	exponentially distributed, probabilistically approximately complete
FSM	finite state machine
GCP	the Graph Colouring Problem
GCP(G, k)	graph colouring instace with graph G and k colours
GLSM	generalised local search machine
HC	Hamilton Circuit

HCP	Hamilton Circuit Problem
init	local search initialisation function
IPLAT	interior plateau state
LMAX	local maximum state
LMIN	local minimum state
lsc	local search cost
M,M_i,M_i'	generally used for denoting GLSM instances
N	neighbourhood relation (for local search)
$\mathcal{NP}$	complexity class $\mathcal{NP}$ (non-deterministic, polynomial time)
$\mathbb{N}$	the set of natural numbers, including zero
p, p_i	generally denote probabilities
$\mathcal{P}$	complexity class $\mathcal{P}$ (deterministic, polynomial time)
PAC	probabilistically approximately complete
$\mathbb{R}$	the set of real numbers
$\mathbb{R}^+$	the set of positive real numbers (without 0)
SAT	the Propositional Satisfiability Problem
sdnclu	standard deviation of the objective function (*i.e.*, number of unsatisfied clauses) for SLS-based SAT algorithms
SQD	solution quality distribution
stddev	standard deviation
SLMAX	strict local maximal state
SLMIN	strict local minimum state
SLOPE	slope state
SLS	stochastic local search
step	local search step function
TSP	the Travelling Salesperson Problem
UNSAT	the Propositional Unsatisfiability Problem (complement of SAT)
VAL	the Propositional Validity Problem
$\mathbb{Z}_n$	$= \{0, 1, \ldots, n-1\}$ the integer residues modulo n