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 $a \in Assign(F)$, we define $\pi_{\mathsf{RI}}(a) := D$,

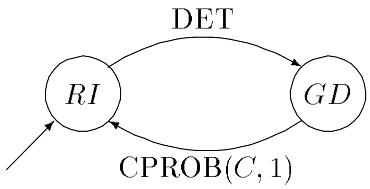

Figure 4.1: GLSM realising GSAT; transition condition $C \equiv$ mcount(maxSteps).

```
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 $\forall a' \in Assign(F) : D(a') = 2^{-n}$ and $n = \# Var(F)$ 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 $s' = max(\{score(a') \mid a' \in A'\})$ the maximal score within this set. Now, we define $A'' = \{a' \in A' \mid score(a') = s'\}$, the set of neighbouring assignments with maximal score. Based on A'', we finally define the state semantics for state type GD:

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

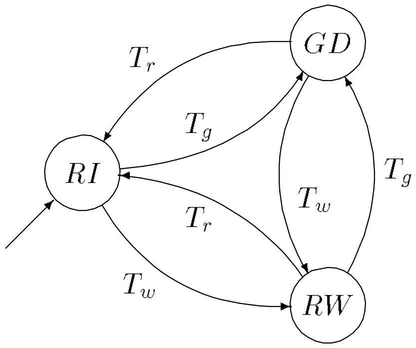

Figure 4.4: GLSM realising GWSAT; transition types: $T_r \equiv \mathrm{CPROB}(C, 1)$ with condition $C \equiv \mathsf{mcount}(\mathsf{maxSteps}); T_w \equiv \mathrm{CPROB}(\neg C, \mathsf{wp}); T_g \equiv \mathrm{CPROB}(\neg C, \mathsf{1} - \mathsf{wp})$.

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.

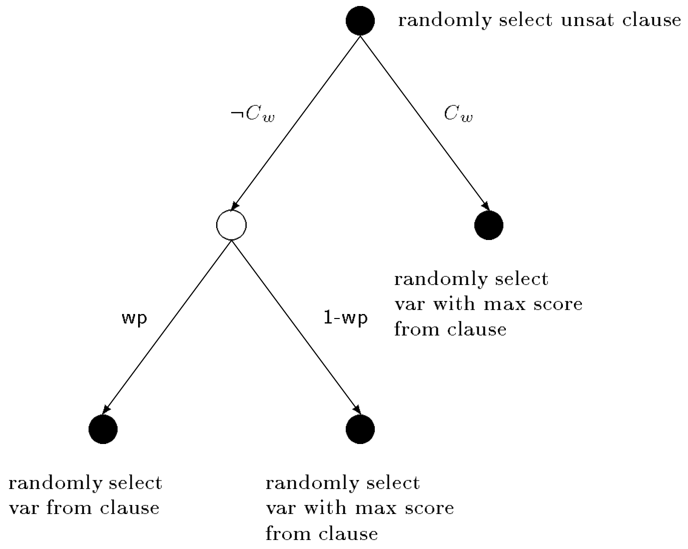

Figure 4.6: Decision tree representation for GLSM state WS; condition $C_w \equiv \exists x \in \mathit{Var}(c') : \mathit{score}_b(x) = 0$, where c' is the selected clause.

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.

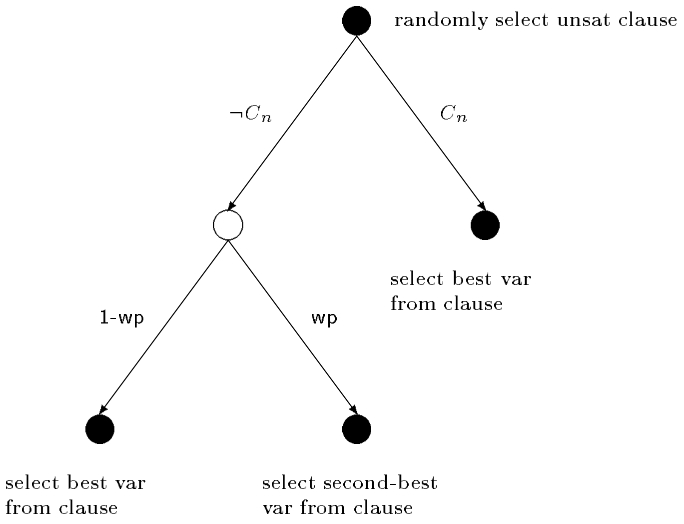

Figure 4.7: Decision tree representation for GLSM state NOV; condition $C_n \equiv$ "best var does not have minimal age"; "best" and "second-best" relate to GSAT-score of variables.

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

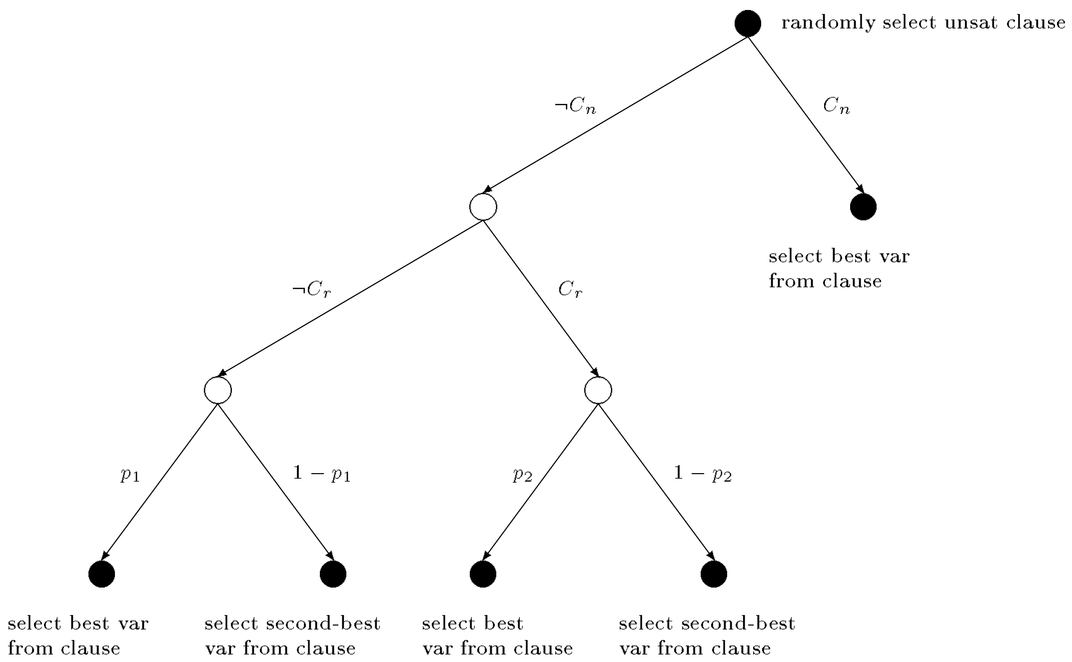

Figure 4.8: Decision tree representation for GLSM state RNOV; condition $C_n \equiv$ "best var does not have minimal age"; condition $C_r \equiv$ "score diff between best and second best var" > 1; "best" and "second-best" relate to GSAT-score of variables; $p_1 = \max\{1 - 2p, 0\}$, $p_2 = \min\{2 -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

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 $\mathsf{wp} = 1$.

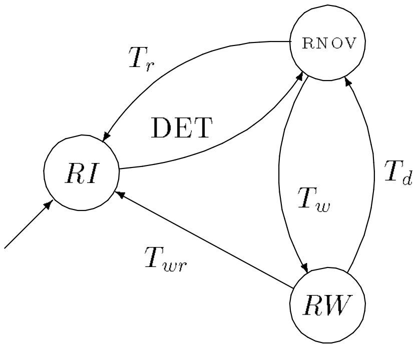

Figure 4.9: GLSM realising R-Novelty; transition types: $T_r \equiv \text{CPROB}(C_r, 1)$, $T_w \equiv \text{CPROB}(C_w \wedge \neg C_r, 1)$, $T_d \equiv \text{CPROB}(\neg C_r, 1)$, $T_{wr} \equiv \text{CPROB}(C_r, 1)$ with conditions $C_r \equiv \textsf{mcount}(\textsf{maxSteps})$, $C_w \equiv \textsf{mcount}(100)$.

R-Novelty's variable selection strategy is even more deterministic than Novelty's; in particular, it is completely deterministic for $\textsf{wp} \in \{0, 0.5, 1\}$. 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.

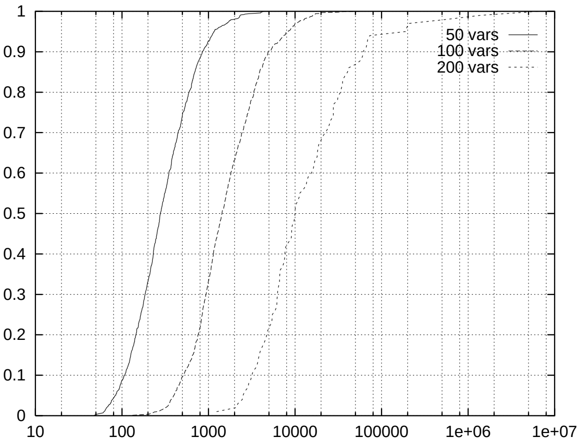

Figure 4.10: Hardness distributions for Random-3-SAT test-sets. The x-axis shows the median search cost (number of local search steps) for WalkSAT per instance.

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 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

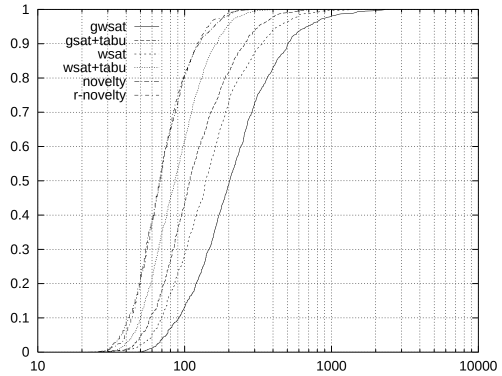

Figure 4.11: Problem instance `uf100-430/easy`, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

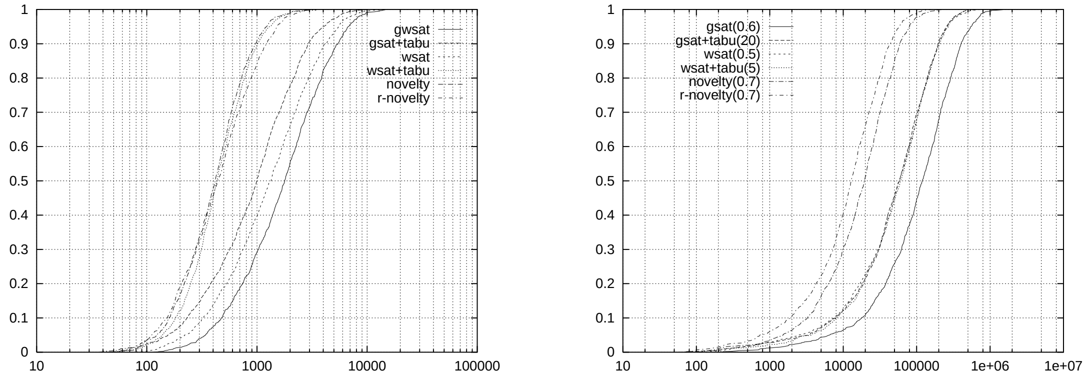

Figure 4.12: Problem instance `uf100-430/med` *(left)* and `uf100-430/hard` *(right)*, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries / instance.

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

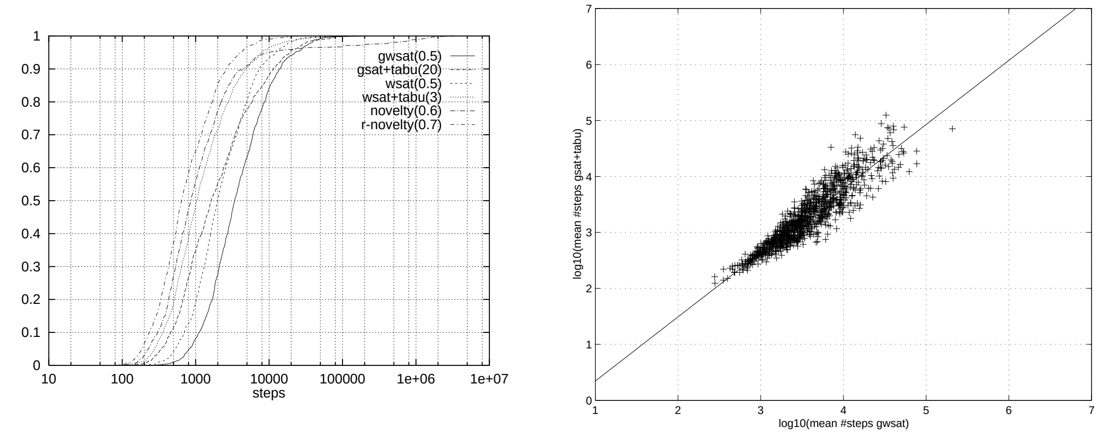

Figure 4.13: *Left:* Hardness distributions across test-set `uf100-430` for various algorithms; x-axis: median number of local search steps / solution. *Right:* Correlation between average local search cost for GWSAT (horizontal) and GSAT+tabu (vertical) for test-set `uf100-430`, using approx. optimal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

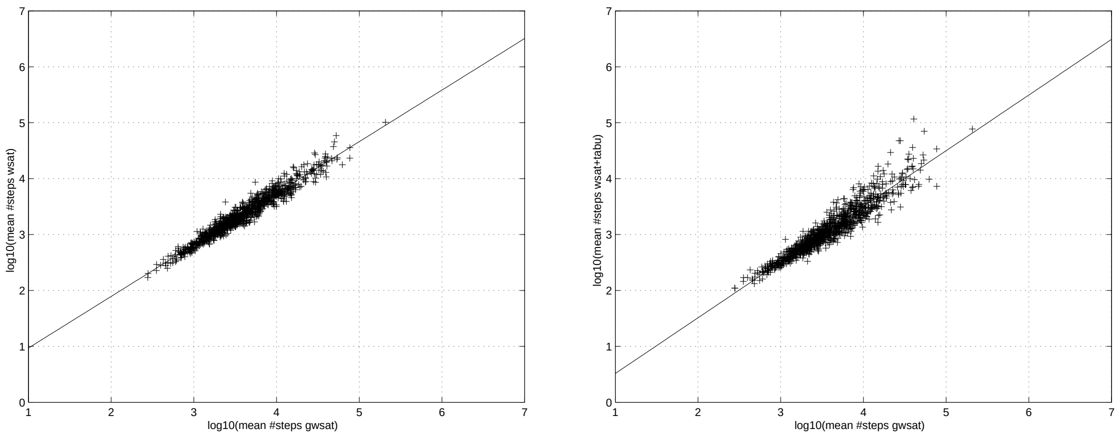

Figure 4.14: *Left:* Correlation between average local search cost for GWSAT (horizontal) and GSAT+tabu (vertical) for test-set `uf100-430`, using approx. optimal noise. *Right:* Same for GWSAT and WalkSAT+tabu.

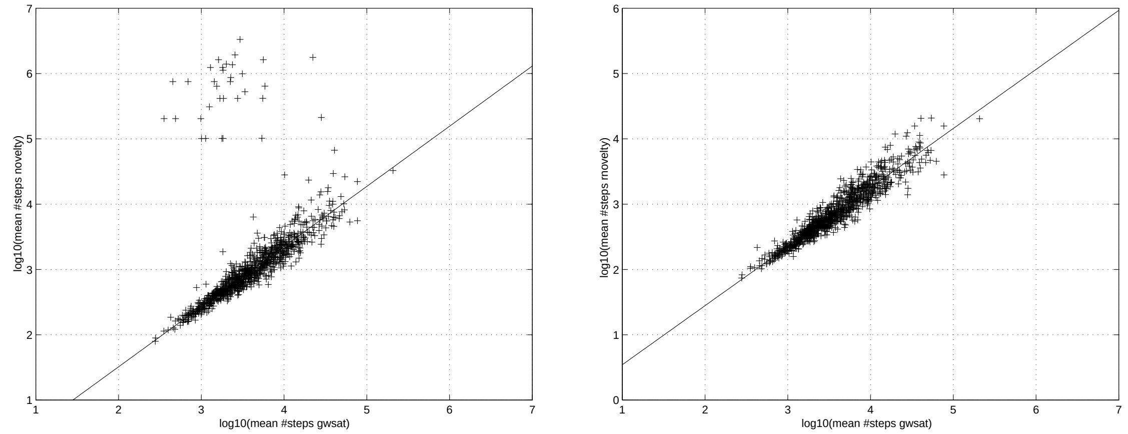

Figure 4.15: *Left:* Correlation between average local search cost for GWSAT (horizontal) and Novelty (vertical) for test-set `uf100-430`, using approx. optimal noise. *Right:* Same for GWSAT and R-Novelty.

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

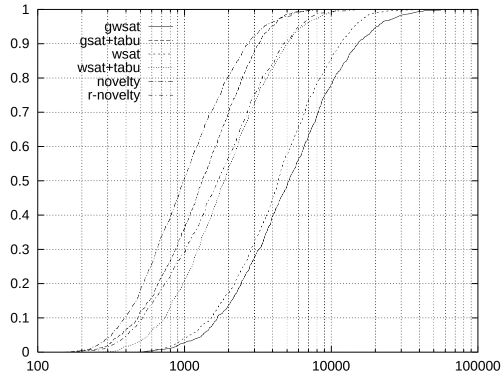

Figure 4.16: Problem `flat100-239/easy`, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

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

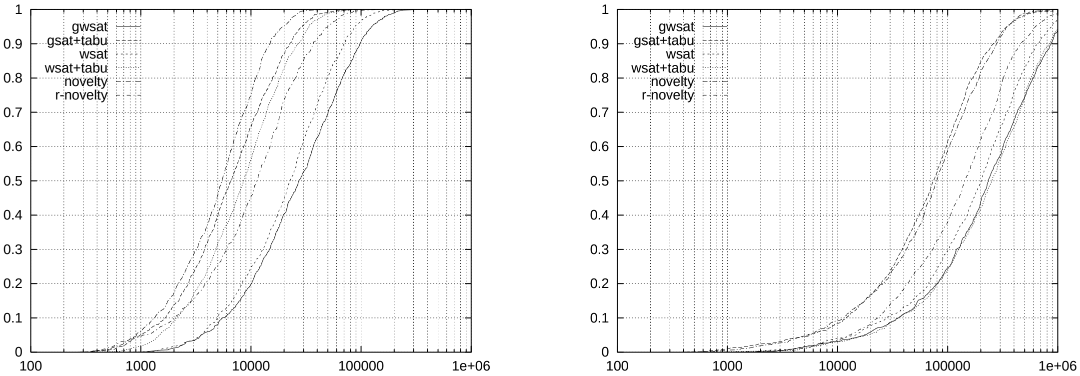

Figure 4.17: Problem `flat100-239/med` *(left)* and `flat100-239/hard` *(right)*, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries / instance.

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.

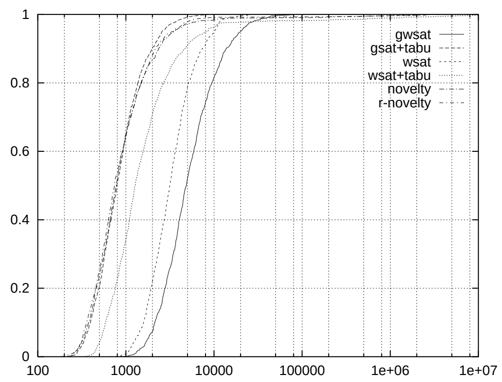

Figure 4.18: Hardness distributions across test-set `flat50-239` for various algorithms; x-axis: median number of local search steps / solution.

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.

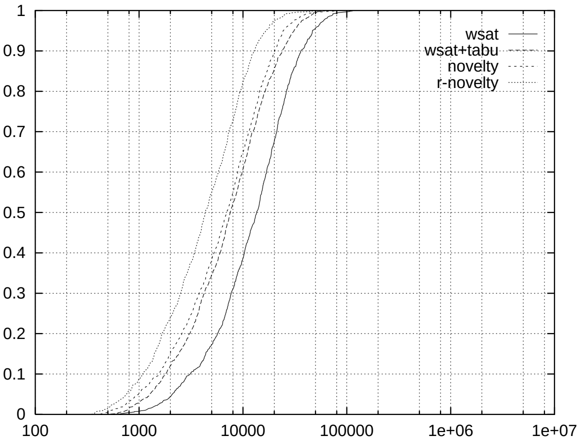

Figure 4.19: Problem instance `bw_large.a`, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

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.

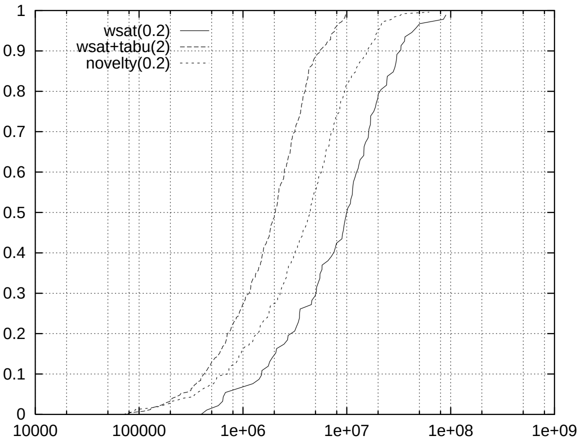

Figure 4.20: Problem instance `bw_large.c`, RLDs for various algorithms, approx. optimal noise, based on 250 tries.

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

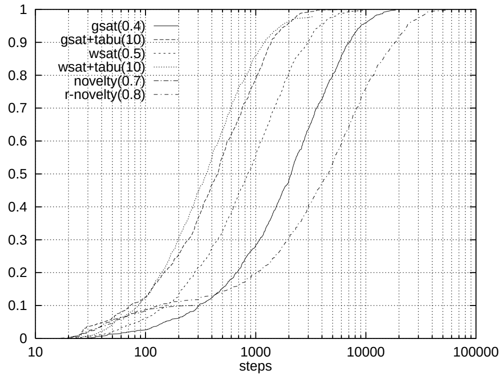

Figure 4.21: Problem instance `ais6`, RLDs for various algorithms, approx. optimal noise, based on 1,000 tries.

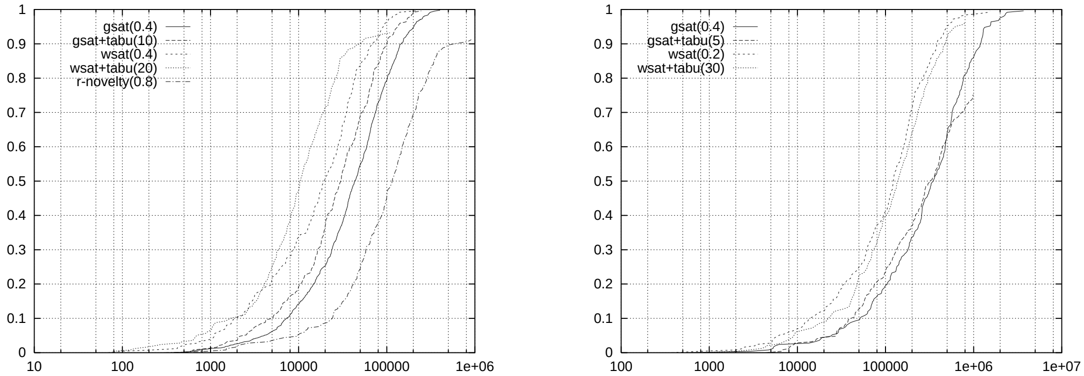

Figure 4.22: Problem instances `ais8` *(left)* and `ais10` *(right)*, RLDs for various algorithms, approx. optimal noise, based on 250 tries / instance.

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.