On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT

Holger H. Hoos

University of British Columbia
Department of Computer Science
2366 Main Mall, Vancouver, B.C., Canada V6T 1Z4
hoos@cs.ubc.ca

Abstract

Stochastic local search (SLS) algorithms for the propositional satisfiability problem (SAT) have been successfully applied to solve suitably encoded search problems from various domains. One drawback of these algorithms is that they are usually incomplete. We refine the notion of incompleteness for stochastic decision algorithms by introducing the notion of "probabilistic asymptotic completeness" (PAC) and prove for a number of well-known SLS algorithms whether or not they have this property. We also give evidence for the practical impact of the PAC property and show how to achieve the PAC property and significantly improved performance in practice for some of the most powerful SLS algorithms for SAT, using a simple and general technique called "random walk extension".

Introduction

Stochastic local search (SLS) algorithms for the propositional satisfiability problem (SAT) have attracted considerable attention within the AI community over the past few years. They belong to the most powerful methods for practically solving large and hard instances of SAT, and outperform the best systematic search methods on a number of domains. However, one of the problems with these algorithms is the fact they are usually *incomplete*, *i.e.*, they cannot be used to prove that a given problem instance is unsatisfiable and — maybe worse — for soluble problem instances, there is no guarantee that such an algorithm actually finds a solution. Early local SLS algorithms like GSAT (Selman, Levesque, & Mitchell 1992) were mainly based on hill-climbing and got easily trapped in local minima of the objective function induced by the given problem instance. These algorithms used *random restart* after a fixed number of steps to avoid premature stagnation of the search. Later, different strategies were used to effectively escape from local minima without using random restart; one of the most popular of these mechanism is *random walk* (Selman, Kautz, & Cohen 1994), which allows randomised up-hill moves with a fixed probability. Today, the best-performing SLS algorithms for SAT use various more sophisticated strategies, combined with random restart, to prevent early stagnation of the search. However, although these algorithm show a very impressive performance, there are almost no theoretical results on their concrete or asymptotic behaviour.

In this paper, we refine the notion of completeness for the class of Las Vegas decision algorithms (a superclass of SLS algorithms). We introduce the term "probabilistically approximately complete" (PAC) to formalise the notion of probabilistic algorithms which are incomplete, but find a solution of soluble instances with a probability approaching one as the run-time approaches infinity. Next, we present theoretical results on the PAC property for a number of popular SLS algorithms for SAT, including the recently introduced Novelty and R-Novelty variants of the WalkSAT architecture (McAllester, Selman, & Kautz 1997).

However, in practice asymptotic properties like PAC are not always relevant. Therefore, we present some empirical results which indicate that in the cases considered here, the theoretical results have practical consequences when applying the respective algorithms to well-known benchmark problems. Based on these empirical results, we then show how Novelty and R-Novelty, two of the best-performing SLS algorithms for SAT known today, can be further improved with respect to both, their theoretical properties and practical behaviour.

SLS Algorithms for SAT

Stochastic local search approaches for SAT became prominent in 1992, when independently Selman, Levesque, and Mitchell (Selman, Levesque, & Mitchell 1992) as well as Gu (Gu 1992) introduced algorithms based on stochastic local hill-climbing which could be shown to outperform state-of-the-art systematic SAT algorithms on a variety of hard subclasses of SAT (Buro & Kleine-Büning 1992; Selman, Kautz, & Cohen 1994). The algorithms considered here are model finding algorithms for CNF formulae. The underlying state space is always defined as the set of all truth assignments for the variables appearing in the given formula. Local search steps 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. In this paper, we focus on the well-known GSAT and

```
procedure GWSAT(F, maxTries, maxSteps, wp)
  for try := 1 to maxTries do
    a := randomly chosen assignment of the variables in formula F;
    for step := 1 to maxSteps do
      if a satisfies F then return a;
      with probability wp do
        c := randomly selected clause which is unsatisfied under a;
        v := randomly selected variable appearing in c;
      otherwise
        v := randomly selected variable the flip of which minimises
             the number of unsatisfied clauses;
      end with;
      a := a with v flipped;
    end for;
  end for;
  return "no solution found";
end GWSAT;
```

Figure 1: The GWSAT algorithm.

```
procedure WalkSAT(F, maxTries, maxSteps, Select)
  for try := 1 to maxTries do
    a := randomly chosen assignment of the variables in F;
    for step := 1 to maxSteps do
      if a satisfies F then return a;
      c := randomly selected clause which is unsatisfied under a;
      v := variable from a selected according to a heuristic Select;
      a := a with v flipped;
    end for;
  end for;
  return "no solution found";
end WalkSAT;
```

Figure 2: The WalkSAT algorithm.

WalkSAT family of algorithms (Selman, Kautz, & Cohen 1994; Gent & Walsh 1993b; McAllester, Selman, & Kautz 1997), which provided a substantial driving force for the development of SLS algorithms for SAT and have been very successful when applied to a broad range of problems from different domains.

The GWSAT algorithm is outlined in Figure 1. Starting from a randomly chosen variable assignment, it repeatedly flips variables according to the following heuristic: With a fixed probability *wp*, a currently unsatisfied clause is randomly selected and one of the variables appearing in it (also randomly selected) is flipped; this is called a *random walk step*. In the remaining cases, one of the variables which, when flipped, achieve the maximal increase (or least decrease) in the total number of satisfied clauses is selected and flipped. If after *maxSteps* such flips no solution is found, the search is started from a new, randomly chosen assignment. If after *maxTries* such tries still no solution is found, the algorithm terminates unsuccessfully. GWSAT with *wp*=0 corresponds to the original GSAT algorithm.

WalkSAT algorithms (*cf.* Figure 2) also start from a randomly chosen variable assignment and repeatedly select one of the clauses which are violated by the current assignment. Then, according to some heuristic a variable occurring in this clause is flipped using a greedy bias to increase the total number of satisfied clauses. For the original WalkSAT algorithm, in the following referred to simply as WalkSAT, the following heuristic is applied. If in the selected clause variables can be flipped without violating other clauses, one of these is randomly chosen. Otherwise, with a fixed probability p a variable is randomly chosen from the clause and with probability $1-p$ a variable is picked which minimises the number of clauses which are currently satisfied but would become violated by the variable's flip (number of breaks).

Other, more recently introduced WalkSAT algorithms (McAllester, Selman, & Kautz 1997) are given by the following heuristics for selecting the variable to be flipped within the selected clause:

WalkSAT/TABU Same as WalkSAT, but uses a tabu-list of length *tl* to ensure that after a variable has been flipped it cannot be flipped for the next *tl* steps. If within the selected clause, all variables are tabu, no variable is flipped (a so-called *null-flip*).

Novelty Considers the variables in the selected clause sorted according to their score, *i.e.*, the difference in the total number of satisfied clauses a flip would cause. If the best variable according to this ordering (*i.e.*, the one with maximal score) is not the most recently flipped one, it is flipped, otherwise, it is flipped with a fixed probability *1-p*, while in the remaining cases, the second-best variable is flipped.

R-Novelty Like Novelty, but in the case where the best variable is the most recently flipped one the decision between the best and second-best variable probabilistically depends on their score difference — the details are not important here and can be found in (McAllester, Selman, & Kautz 1997). Additionally, every 100 steps, instead of using this heuristic, the variable to be flipped is randomly picked from the selected clause.

As for GWSAT, if after *maxSteps* such flips no solution is found, the search is started from a new, randomly selected assignment; and if after *maxTries* such tries still no solution is found, the algorithm terminates unsuccessfully.

Asymptotic Behaviour of Algorithms

Las Vegas algorithms[1] are stochastic algorithms which, if they find a solution for the given problem, guarantee the correctness of this solution. This is captured by the following definition (adapted from (Hoos & Stützle 1998)):

Definition 1 *Let* Π *be a problem class. An algorithm* A *is a* Las Vegas algorithm *for problem class* Π*, if (i) whenever for a given problem instance* $\pi \in \Pi$ *it returns a solution* s*,* s *is guaranteed to be a valid solution of* π*, and (ii) on each given instance* π*, the run-time of* A *is a random variable* $RT_{A,\pi}$.

Stochastic local search algorithms are special cases of Las Vegas algorithms. According to this definition, Las Vegas algorithms are always correct, but they are not necessarily complete, *i.e.*, even if a given problem instance has a solution, a Las Vegas algorithm is generally not guaranteed to find it. However, even an incomplete Las Vegas algorithm might be asymptotically complete in the sense that by running it long enough, the probability of missing an existing

[1] The term was originally coined by Laszlo Babai in 1979 (personal communication).

solution can be made arbitrarily small. This property, which is often referred to as "convergence" in the literature on optimisation algorithms, is theoretically interesting (*cf.* (Geman & Geman 1984)) and is also potentially very relevant for practical applications. We formalise this notion for Las Vegas algorithms for decision problems in the following way.

Definition 2 *Let Π be a decision problem and A a Las Vegas Algorithm for Π. For a given problem instance $\pi \in \Pi$, let $P_s(RT_{A,\pi} \leq t)$ denote the probability that A finds a solution for π in time $\leq t$ and let $P \subseteq \Pi$. Then we call A* probabilistically approximately complete (PAC) for P, *if for all* soluble *instances $\pi \in P$, $\lim_{t\to\infty}[P_s(RT_{A,\pi} \leq t)] = 1$. Furthermore, we call A* essentially incomplete for P, *if it is not PAC for P, i.e., there is a soluble instance $\pi \in P$, for which $\lim_{t\to\infty}[P_s(RT_{A,\pi} \leq t)] < 1$. If A is PAC / essentially incomplete for Π, we call A* probabilistically approximately complete (PAC) / essentially incomplete.

These concepts refine the usual distinction between complete and incomplete algorithms. The simplest stochastic search algorithm which is provably PAC, is "random picking": Given a set S of candidate solutions (for SAT: all assignments for the variables appearing in the given formula), iteratively select an arbitrary element of S such that in each step, each element of S is selected with equal probability $p = 1/|S|$. Obviously this algorithm is PAC, as the probability of finding a solution in t steps, assuming that there are $k > 0$ different solutions in S, is $1 - (1 - kp)^t$. But of course, random picking is typically hopelessly inefficient in practice, in the sense that even for NP-complete problems like SAT, algorithms like GSAT or WalkSAT are orders of magnitude faster than it in finding solutions. Nevertheless, in practice, the PAC property can be important, especially with respect to the robustness of Las Vegas algorithms, as it guarantees that that the algorithm will, given sufficient time, almost certainly find a solution for a soluble problem instance.

Theoretical Results

In this section we prove a number of results regarding the PAC property of various well-known SLS algorithms for SAT. In all cases, we consider the "pure" search strategies without random restart. The rationale behind this is the following. Random restart can be easily added to any given search strategy and the resulting algorithm will always be PAC if, as the run-time increases, an arbitrarily large number of restarts can occur. However, this trivial result is practically irrelevant, as the run-times for which this asymptotic behaviour can be observed are even considerably higher than for random picking. On the other hand, as we will show later, other mechanisms, such as random walk, which achieve the PAC property are much more effective in practice.

GSAT

We start with proving that the basic GSAT algorithm is essentially incomplete. Although the fact that GSAT can get stuck in local minima of the objective function is well-known, we are not aware of a formal proof. We therefore give a proof here which also demonstrates the technique we use later to prove some previously unknown essential incompleteness results. Note that generally, to show that an SLS algorithm for SAT is not PAC, *i.e.*, essentially incomplete, it is sufficient to find a satisfiable problem instance and a reachable state of the algorithm from which no solution can be reached. Since all algorithms considered here start their search by randomly picking an arbitrary variable assignment, all assignments are reachable and for proving essential incompleteness we can assume any current variable assignment.

Theorem 1 *Basic GSAT, i.e., GWSAT with $\mathrm{wp} = 0$ (without restart), is essentially incomplete.*

Proof. Let $F_1 = \bigwedge_{i=1}^{10} c_i$ be the CNF formula consisting of the clauses:

$$\begin{array}{rcl}
c_1 & \equiv & \neg x_1 \vee x_2 \vee \neg z_1 \\
c_2 & \equiv & \neg x_1 \vee x_2 \vee z_1 \\
c_3 & \equiv & \neg x_1 \vee x_2 \vee \neg z_2 \\
c_4 & \equiv & \neg x_1 \vee x_2 \vee z_2 \\
c_5 & \equiv & x_1 \vee \neg x_2 \vee \neg z_1 \\
c_6 & \equiv & x_1 \vee \neg x_2 \vee z_1 \\
c_7 & \equiv & x_1 \vee \neg x_2 \vee \neg z_2 \\
c_8 & \equiv & x_1 \vee \neg x_2 \vee z_2 \\
c_9 & \equiv & x_1 \vee x_2 \vee \neg y \\
c_{10} & \equiv & x_1 \vee x_2 \vee y
\end{array}$$

F_1 is satisfiable and has 8 models ($x_1 = x_2 = \top; y, z_1, z_2$ arbitrary). Consider the 4 possible assignments for which $x_1 : \bot, x_2 : \bot, y : \bot$ and call this set A. Analogously, let B denote the set consisting of the 4 assignments with $x_1 : \bot, x_2 : \bot, y : \top$. Note that $A \cup B$ does not contain any solution. Assume that GSAT's current assignment is an element of A. Each assignment from A satisfies all clauses except c_{10} and the variables receive the following scores: $x_1, x_2 : -1$, $y, z_1, z_2 : 0$. Since GSAT always flips one of the variables with the highest score, z_1, z_2 or y is flipped. By flipping z_1 or z_2, another assignment in A is reached. by flipping y, depending on the values of z_1 and z_2 an assignment of B is reached. Now, all clauses except c_9 are satisfied and the variable receive the same scores as before. Again, only z_1, z_2 or y can be flipped. While in the former case, only assignments from B can reached, by flipping y always an assignment in A is obtained. Therefore, starting from an assignment in A or B, GSAT cannot reach a solution which proves the theorem. □

Adding Random Walk — GWSAT

Next, we show that by adding random walk (Selman, Kautz, & Cohen 1994), basic GSAT can be made PAC. This result extends earlier work which established that satisfiable 2-SAT instances are solved by pure random walk in $O(n^2)$ time on average (Papadimitriou 1991). Using similar techniques, we first show that from an arbitrary assignment, the hamming distance to the nearest solution can always be decreased by a random walk step (flipping a variable in a randomly selected, currently unsatisfied clause).

Lemma 1 *Let a be the current (non-solution) assignment, and s the solution with minimal hamming distance h from a. For arbitrary a and s there is always a random walk step which decreases h by one.*

Proof. Assume that no such random walk step exists. Then none of the variables whose values are different in a and s can appear in an unsatisfied clause. But since a is not a solution, there has to be at least one clause c which is violated by a; now the variables appearing in c have the same value in a and s, therefore s also violates c and cannot be a solution. Thus the initial assumption has to be false, which proves the lemma. □

Based on this lemma, instead of directly showing that GWSAT is PAC, we prove a slightly more general result.

Theorem 2 *Consider the class of SLS algorithms for SAT which accept arbitrary CNF formulae as their input and search the space of assignments of the given formula. Any such algorithm which, for any current assignment, executes a random walk step with a probability of at least $p > 0$ at any given time, is PAC.*

Proof. Consider an arbitrary algorithm A satisfying the conditions from the theorem. For a given CNF formula with n variables and k clauses, we show that there exists a $p' > 0$ such that from each non-solution assignment the algorithm can reach a solution with a probability $p'' \geq p'$: Let a be the current (non-solution) assignment, and s the solution with minimal hamming distance h from a. Using Lemma 1 inductively, one can construct a sequence of h random walk steps from a to s. Next, we derive a lower bound for the probability with which A will execute this sequence.

Note first that for any assignment a', the number of unsatisfied clauses is always $\leq k$, and the number of literals occurring in unsatisfied clauses is $\leq k \cdot l$, where l is the maximal number of literals per clause for the given instance. Therefore, if A executes a random walk step, the probability to select a variable which decreases h is at least $1/(k \cdot l)$. Thus, the overall probability of executing a random walk step which decreases the hamming distance to the nearest solution is at least $p/(k \cdot l)$. Since we have a lower bound p for the probability of executing a random walk step independently from the current assignment, a lower bound for the probability of executing the correct h step sequence to reach the solution can be estimated as $[p/(k \cdot l)]^h$.

Finally, note that always $h \leq n$, and therefore the probability of reaching a solution from any given assignment a is at least $p' = [p/(k \cdot l)]^n$. Consequently, the following lower bound for the probability that A finds a solution within t steps can be easily obtained:[2]

$$1 - (1 - p')^{\lfloor t/n \rfloor}$$

For $t \to \infty$, this bound obviously converges to 1, which proves A's approximate completeness. □

Since GWSAT satisfies the conditions of Theorem 2 for arbitrary walk probabilities $wp > 0$ (let $p = wp$), we immediately get the following

Corollary 1 *GWSAT is approximately complete for all $wp > 0$.*

Note that the proof of Theorem 2 relies critically on the fact that the algorithm can decrease the hamming distance to the nearest solution in each step. Our proof shows that this is guaranteed if arbitrarily long sequences of random walk steps can be performed with a probability $p'' \geq p' > 0$ which is independent of the number of steps that have been performed in the past.

[2]This estimate is based on the observation that when partitioning a run of t steps into segments of n steps, the success probabilities for each of these can be independently bounded by p'.

WalkSAT Algorithms

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

Theorem 3 *Novelty (without restart) is essentially incomplete for arbitrary noise parameter settings p.*

Proof. Let $F_2 = \bigwedge_{i=1}^{6} c_i$ be the CNF formula consisting of the clauses:

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

F_2 is satisfiable and has exactly one model ($x_1 = x_2 = \top, y = z_1 = z_2 = \bot$). Now assume that the algorithm's current assignment is $a_1 \equiv (x_1 = x_2 = y = z_1 = z_2 = \top)$. In this situation, all clauses except c_3 are satisfied. Applying Novelty, c_3 will be selected and the variables receive the following scores: $x_1 : 0, x_2 : 0, y : -1$. Since regardless of the noise parameter, Novelty always flips the best or second-best variable, either x_1 or x_2 will be flipped.

Because both cases are symmetric, we assume without loss of generality that x_1 is flipped. This leads to the assignment $a_2 \equiv (x_1 = \bot, x_2 = y = z_1 = z_2 = \top)$ which satisfies all clauses except c_2. The scores for x_1 and x_2 are both 0, and since x_1 is the most recently flipped variable, x_2 will be picked now. The current assignment at this point is $a_3 \equiv (x_1 = x_2 = \bot, y = z_1 = z_2 = \top)$ which satisfies all clauses except c_4. Again, x_1 and x_2 have the same score of 0, but now x_2 is the most recently flipped variable, so x_1 will be flipped. Now, the current assignment is $a_4 \equiv (x_1 = \top, x_2 = \bot, y = z_1 = z_2 = \top)$, which leaves only c_1 unsatisfied. As before, x_1 and x_2 both receive a score of 0, and x_2 will be flipped. But this leads back to the assignment $a_1 \equiv (x_1 = x_2 = y = z_1 = z_2 = \top)$; therefore, Novelty got stuck in a loop.

Therefore, we found a soluble problem instance F_2 and an initial assignment a_1 for which Novelty can never reach a solution which proves the theorem. □

Using the same formula F_2 and analogous arguments, it is now easy to prove:

Theorem 4 *WalkSAT/TABU and R-Novelty (without restart) are essentially incomplete for arbitrary tabu-list lengths and noise parameter settings, resp.*

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

Practical Relevance

While the results from the last section are theoretically interesting in a similar sense as, *e.g.*, the well-known convergence result for Simulated Annealing (Geman & Geman 1984), it is not clear that they are also practically relevant. We therefore investigated the following two questions:

1. For PAC algorithms, is the convergence of the success probability fast enough to be observable when applying these algorithms to hard problem instances?
2. For essentially incomplete algorithms, is the characteristic stagnation behaviour observable for any of the usual benchmarks for SAT algorithms?

For answering these questions, we empirically analysed the behaviour of GWSAT, WalkSAT, WalkSAT/TABU, Novelty, and R-Novelty when applied to a number of benchmark problems for SAT algorithms, including hard Random-3-SAT instances like,*e.g.*, used in (Selman, Levesque, & Mitchell 1992; Parkes & Walser 1996; McAllester, Selman, & Kautz 1997), SAT-encoded graph colouring instances similar to the ones described in (Selman, Levesque, & Mitchell 1992), and SAT-encoded blocks world planning problems taken from (Kautz & Selman 1996). For our experiments, we applied each algorithm multiply to the same problem instance and measured the number of solutions found as well as the number of steps required for finding each solution. In order to observe a close-to-asymptotic runtime behaviour of the pure strategies, we used extremely high values of the *maxSteps* parameter and no random restart ($maxTries = 1$); we also made sure that between the length of the longest successful run and the cutoff (given by *maxSteps*) there was at least an additional factor of ten. Furthermore, we optimised the noise parameter settings for each benchmark problem and for each problem size such that the expected number of steps required to find a solution was minimised.

This study shows that the answers to the questions raised above are positive in both cases: PAC behaviour as well as essential incompleteness can be observed on the benchmark problems considered here. This is exemplified by the following results.[3] When applied to a set of 1,000 satisfiable, hard 3-colouring problems in random graphs (50 nodes, 239 edges), GWSAT and WalkSAT solve all instances in all runs with a mean number of less than 10,000 steps per run. When performing the same experiment for WalkSAT/TABU, Novelty, and R-Novelty, however, while still all instances were solved, 27, 9, and 6 of them, resp., were not solved in some of the runs, even when using a huge cutoff value of $maxSteps = 10^7$. Similar results could be obtained for Novelty applied to hard Random-3-SAT problems (100 variables, 430 clauses) and for large blocks world planning instances (like `bw_large.c` from (Kautz & Selman 1996)). In our experiments, we never observed stagnation behaviour for GWSAT or WalkSAT, while for the WalkSAT variants shown to be essentially incomplete, the typical stagnation behaviour often occurred.

Improving SLS Performance

However, consistent with (McAllester, Selman, & Kautz 1997), we generally observed significantly improved performance of WalkSAT/TABU, Novelty, and R-Novelty over WalkSAT or GWSAT in the cases where stagnation behaviour did not occur. Therefore, these algorithms seem to be superior to WalkSAT and GWSAT except for the stagnation behaviour caused by their essential incompleteness. In theory this problem can be easily solved. As mentioned before, adding random restart would make these algorithms PAC. However, in practice this approach critically relies on the use of good *maxSteps* settings. Unfortunately, in general these are extremely difficult to find *a priori* and today, to our best knowledge, there exist no theoretical results on how to determine good settings. The only empirical results we are aware of, are for GSAT when applied to hard Random-3-SAT problem distributions (Gent & Walsh 1993a); but these results are limited to Random-3-SAT and rely on properties of the respective problem distributions rather than the individual instances. On the other hand, while using inappropriately chosen *maxSteps* settings generally still eliminates essential incompleteness, in practice, it leads to extremely poor performance.

Another way of making these algorithms PAC is to extend them with random walk in such a way, that for each local search step, with a fixed probability a random walk step is performed. Random walk apparently has an advantage over random restart, since at least in GWSAT, it is more robust w.r.t. the additionally introduced parameter *wp* than random restart is w.r.t. *maxSteps* (*cf.* (Parkes & Walser 1996)). One reason for this empirically observed phenomenon is related to the inherent randomness of the random walk sequences; random restarts occur after a fixed cutoff time, whereas random walk sequences are probabilistically variable in their length and frequency of occurrence. Furthermore, when using random restart, the search process is re-initialised; consequently, a new try cannot benefit from the search effort spent in previous tries (unless information is carried from one run to the next, which is not the case for the algorithms considered here). The amount of perturbation introduced by a random walk sequence, however, probabilistically depends on the length of the sequence such that small perturbations are much more likely to occur.

Based on these considerations, we extend Novelty and R-Novelty with random walk such that in each search step, with probability *wp*, the variable to be flipped is randomly picked from the selected clause, while in the remaining cases, the variable is selected according to the heuristic for Novelty or R-Novelty, resp. For R-Novelty, we further-

[3] A complete description of the empirical study and its results can be found in (Hoos 1998) and will be presented in more detail in an extended version of this paper.

more omit the deterministic loop breaking strategy which randomly flips a variable from the selected clause every 100 steps. The two algorithms thus obtained, Novelty$^+$ and R-Novelty$^+$, are obviously PAC, as they satisfy the conditions of Theorem 2; but it is not clear whether this is sufficient to overcome the stagnation behaviour observed in practice. We therefore empirically compared their behaviour to the original algorithms, using the same benchmark instances as described above. Different from the original algorithms, when using identical noise and *maxSteps* settings and a probability of $wp = 0.01$ for executing random walk steps, the modified algorithms solved all problem instances from the Random-3-SAT and graph colouring test-set in all runs. For the blocks world planning instance `bw_large.c`, where R-Novelty found only 28 solutions in 1,000 runs á 10^8 steps, R-Novelty$^+$ found 1,000 solutions with an expected local search cost of $8.09 \cdot 10^6$ steps per solution. We also compared the performance of the original and the modified algorithms when applied to instances for which no stagnation behaviour occurs. Here, no significant performance differences could be observed. This indicates that the essentially incomplete behaviour which causes the high means and standard deviations of these hardness distributions is efficiently eliminated by the random walk extension, while for instances which do not suffer from essentially incomplete behaviour, the performance remains mainly unaffected.

Conclusions

In this paper, we have shown a series of new theoretical results on the asymptotic behaviour of a number of state-of-the-art stochastic local search algorithms for SAT. As we have shown, these results are not only of theoretical interest, but their impact can also be observed when applying them to solve well-known benchmark problems. While theoretically, the standard random restart technique which is generally used with these algorithms is sufficient to make them PAC, the practical performance thus obtained is usually relatively poor, unless the cutoff parameter *maxSteps* is carefully tuned. Using a different approach, which is analogous to the random walk extension of GSAT introduced in (Selman, Kautz, & Cohen 1994), these difficulties can be avoided. By extending Novelty and R-Novelty with random walk, these essentially incomplete algorithms could be made PAC; at the same time, their empirically observed performance could be considerably improved without any additional parameter tuning.

Our results suggest that the PAC property is an important concept for the theoretical analysis as well as for the practical performance of modern SLS algorithms for SAT. Some of the best-performing algorithms, like WalkSAT/TABU, Novelty, and R-Novelty, are essentially incomplete and the corresponding stagnation behaviour can be observed when applying them to standard benchmark problems. This indicates that these algorithms, as observed for the pure R-Novelty strategy (without restart and loop-breaking) in (McAllester, Selman, & Kautz 1997), are on the verge of being too deterministic. Empirical evidence indicates that at least for the cases studied here, using a simple and generally applicable technique, the random walk extension, this problem can be overcome, resulting in hybrid algorithms which show superior performance as well as considerably increased robustness. However, although our empirical results so far are very suggestive, they are of a somewhat preliminary nature. Therefore, we feel that a considerably extended analysis should be undertaken to further investigate the practical impact of the PAC property in general, and the random walk extension in particular.

Acknowledgements

I wish to thank Joe Culberson, Ian Gent, Bart Selman, David McAllester, and Henry Kautz for their input during various discussions. Furthermore I gratefully acknowledge the comments of the anonymous referees and David Poole which helped to improve this paper. This research was partially supported IRIS Phase-III Project BOU, "Preference Elicitation and Interactive Optimization."

References

Buro, M., and Kleine-Büning, H. 1992. Report on a SAT Competition. Technical Report 110, Dept. of Mathematics and Informatics, University of Paderborn, Germany.

Geman, S., and Geman, D. 1984. Stochastic Relaxation, Gibbs Distribution, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6:721–741.

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

Gent, I. P., and Walsh, T. 1993b. Towards an Understanding of Hill–Climbing Procedures for SAT. In *Proceedings of AAAI'93*, 28–33. MIT press.

Gu, J. 1992. Efficient Local Search for Very Large-Scale Satisfiability Problems. *ACM SIGART Bulletin* 3(1):8–12.

Hoos, H. H., and Stützle, T. 1998. Evaluating Las Vegas Algorithms — Pitfalls and Remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 238–245. Morgan Kaufmann Publishers, San Francisco, CA.

Hoos, H. H. 1998. *Stochastic Local Search — Methods, Models, Applications*. Ph.D. Dissertation, Fachbereich Informatik, Technische Universität Darmstadt.

Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of AAAI'96*, volume 2, 1194–1201. MIT Press.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for Invariants in Local Search. In *Proceedings of AAAI'97*, 321–326.

Papadimitriou, C. 1991. On Selecting a Satisfying Truth Assignment. In *Proc. 32nd IEEE Symposium on the Foundations of Computer Science*, 163–169.

Parkes, A. J., and Walser, J. P. 1996. Tuning Local Search for Satisfiability Testing. In *Proceedings of AAAI'96*, volume 1, 356–362. MIT Press.

Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise Strategies for Improving Local Search. In *Proceedings of AAAI'94*, 337–343. MIT Press.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of AAAI'92*, 440–446. MIT Press.