Warped Landscapes and Random Acts of SAT Solving

Dave A. D. Tompkins and Holger H. Hoos*
Department of Computer Science
University of British Columbia

Abstract

Recent dynamic local search (DLS) algorithms such as SAPS are amongst the state-of-the-art methods for solving the propositional satisfiability problem (SAT). DLS algorithms modify the search landscape during the search process by means of dynamically changing clause penalties. In this work, we study whether the resulting, 'warped' landscapes are easier to search than the landscapes that correspond to the original problem instances. We present empirical evidence indicating that (somewhat contrary to common belief) this is not the case, and that the main benefit of the dynamic penalty update mechanism in SAPS is an effective diversification of the search process. In most other high-performance stochastic local search algorithms, the same effect is achieved by the strong use of randomised decisions throughout the search. We demonstrate that in SAPS, random decisions are only required in the (standard) search initialisation procedure, and can be completely eliminated from the remainder of the subsequent search process without any significant change in the behaviour or performance of the resulting algorithms compared to the original, fully randomised SAPS algorithm. We conjecture that the reason for this unexpected result lies in the ability of the deterministic variants of the scaling and smoothing mechanism and the subsidiary iterative best improvement mechanism underlying SAPS to effectively propagate the effects of initial randomisation throughout a search process that shows the sensitive dependence on inditial conditions that is characteristic for chaotic processes.

1 Introduction and Background

The Propositional Satisfiability Problem (SAT) is one of the most prominent hard combinatorial decision problems; it plays an important role in many areas of computing and is of substantial theoretical and practical interest. A popular and very successful approach to solving SAT is to apply Stochastic Local Search (SLS) algorithms. A well-known class of SLS algorithms for SAT is based on the idea of Dynamic Local Search (DLS); DLS algorithms dynamically change the evaluation function of a subsidiary local search algorithm, and hence the search landscapes, during the search process. Well-known DLS algorithms for SAT include Morris' Breakout Method [8], GSAT with clause weighting [11], Frank's GSAT with rapid weight adjustment [1], the Discrete Lagrangian Method (DLM) [14], the Smoothed Descent and Flood (SDF) and Exponentiated Sub-Gradient (ESG) algorithms [9, 10], as well as our recent Scaling and Probabilistic Smoothing (SAPS) algorithm, which achieves state-of-the-art performance for SAT and MAX-SAT [6, 13].

All of these DLS algorithms use dynamically changing clause penalties in order to modify (or *warp*) the underlying search landscape during the search. In particular, the SAPS algorithm works as follows: All propositional variables are randomly initialised (one or zero), and each clause penalty is initialised to one. The evaluation function for SAPS is the sum of the unsatisfied clause penalties. The subsidiary local search algorithm of SAPS is a simple iterative best improvement method that in each step flips the variable that decreases the evaluation function the most, with ties broken randomly. (Note that the same basic search method is underlying the GSAT algorithm [7].) If no variable flip can reduce the evaluation function, then the algorithm is at a local minimum. At a local minimum, either with probability wp a variable is randomly selected and flipped (a so-called random walk step), or the clause penalties are adjusted by a scaling step, and possibly a smoothing step. At a scaling step, all unsatisfied clause penalties are multiplied by a fixed value $\alpha > 1$, which is a parameter of the algorithm. After a scaling step, a smoothing step occurs with probability P_{smooth}, in which case all penalties are adjusted to $clp_i = clp_i \times \rho + \overline{clp} \times (1 - \rho)$, where clp_i is the penalty of a given clause c_i, $\overline{clp}$ is the average of all clause

*Corresponding author; mailing address: Holger H. Hoos, Department of Computer Science, University of British Columbia, 2366 Main Mall, Vancouver, BC, Canada, V6T 1Z4; e-mail: hoos@cs.ubc.ca

weights after scaling, and the parameter ρ has a fixed value between zero and one. (A detailed description of SAPS can be found in [6].)

In this work, we study the factors underlying the effectiveness of DLS algorithms such as SAPS for solving hard SAT instances. In particular, we investigate the commonly held belief that the warping of the search space achieved by these algorithms makes the respective problem instances easier to solve and hence can be seen as a form of learning. We present evidence that this belief is essentially incorrect and provide better insights into the role of warping, as realised by the scaling and smoothing mechanism in SAPS, in the context of the impressive performance of state-of-the-art DLS algorithms for SAT. Furthermore, we investigate the role of random decisions in DLS algorithms and present empirical evidence that, contrary to earlier results for other SLS algorithms for SAT, for SAPS the only place where randomisation is required is in the determination of the initial truth assignment. This demonstrates that even a completely deterministic variant of the scaling and smoothing mechanism used in SAPS achieves a sufficient degree of diversification and mobility of the search process, leading to the same excellent performance as the original, heavily randomised SAPS algorithm. Furthermore, the run-time of the resulting algorithm for a given SAT instance solely depends on the initial assignment, which suggests possible directions for further, substantial performance improvements.

2 Warped Intelligence

While solving a given problem instance, DLS algorithms are dynamically warping their underlying search space whenever they update their clause penalties. It has been suggested that the success of this approach is a result of the fact that the clause penalties represent accumulated 'knowledge' about the search space (see, *e.g.*, [1]). In other words, DLS algorithms are 'learning' about the landscape they search. In particular, the resulting warped landscape will be easier to search than the original space. Often, this reflects back to a popular analogy that a DLS algorithm 'fills in the holes', *i.e.*, the local minima, of a given search landscape. Although the notion that the effectiveness of DLS algorithms is explained by their ability to produce warped landscapes in which solutions can be found more efficiently appears to be widely accepted, little evidence has been provided to support this hypothesis.

To investigate the validity of this proposed explanation, we examined the SAPS algorithm, and took *snapshots* of the clause penalties whenever the algorithm had found a solution. These clause penalties were then used to generate weighted SAT instances, which we refer to as the *SAPS generated weighted instances*, where the weight of each clause is simply the corresponding clause penalty used by SAPS at the point when a solution was found. (Here and in the following, we will use the terms 'weight' and 'clause weight' to refer to weights that are statically associated with the clauses of a given, weighted SAT instance, while the terms 'penalty' and 'clause penalty' are reserved for dynamically changing penalty values associated with clauses during the run of a DLS algorithm.) Many existing SLS algorithms for SAT can be generalised easily and naturally to weighted SAT instances by replacing their standard evaluation function, defined as the total number of clauses unsatisfied under a given truth assignment, by a function that maps truth assignments to the corresponding weighted sum of unsatisfied clauses. (Note that standard SAT instances can be seen as a special case of weighted SAT instances, where all clause weights are equal.) We propose that if (on average) an algorithm can solve the weighted problem instance in fewer search steps than the corresponding unweighted problem instance, then the weights are truly making the instance easier.

If the efficiency of SAPS were based on its ability to learn clause weights that render the given instance easier, this should be reflected in reduced search cost when SAPS is initialised with the clause weights from a previous successful run. Note that this is equivalent to restarting the algorithm by randomly reinitialising all variables, while keeping the clause penalties the same. In other words, if all the clause penalties represent 'knowledge' that the algorithm has accumulated about the search space, then the modified SAPS algorithm is starting with all of that knowledge *a priori*.

For this experiment, and all remaining experiments reported in this paper, we chose a number of well-known random and structured instances from SATLIB `www.satlib.org`) [5]. For all experiments, we did not op-

Instance	Unweighted	SAPS Generated Weighted Instances			Permuted SAPS Weighted Instances			Randomly Generated Weighted Instances		
		$q_{0.25}$	Median	$q_{0.75}$	$q_{0.25}$	Median	$q_{0.75}$	$q_{0.25}$	Median	$q_{0.75}$
uf100-easy	81	0.98	1.01	1.06	0.98	1.00	1.02	1.31	1.36	1.46
uf100-hard	3,763	1.08	1.11	1.14	1.00	1.05	1.09	1.03	1.06	1.10
uf250-hard	197,044	0.98	1.06	1.14	0.35	1.01	1.04	0.97	1.03	1.06
uf400-hard	2,948,181	0.92	1.04	1.17	1.00	1.04	1.07	0.95	1.10	1.19
ais10	20,319	1.06	1.09	1.11	1.03	1.08	1.12	1.04	1.11	1.19
bw_large.a	2,499	0.90	0.93	0.98	0.99	1.02	1.04	1.01	1.04	1.07
bw_large.b	34,548	0.97	1.02	1.08	1.00	1.02	1.06	0.99	1.07	1.11
flat100-hard	24,248	0.99	1.02	1.04	0.97	1.02	1.05	0.98	1.01	1.04
logistics.c	9,446	0.97	1.03	1.06	1.02	1.05	1.07	1.05	1.07	1.14
ssa7552-038	3,960	0.86	0.91	0.95	0.94	0.98	1.00	1.02	1.08	1.12

Table 1: Comparison of SAPS performance on SAPS generated weighted instances, permuted SAPS generated instances and randomly weighted instances. For each test instance, SAPS was run 1000 times to obtain a run-length distribution. From that distribution, 25 characteristic runs were selected according to the quantiles $q_{0.02}$, $q_{0.06}$, ..., $q_{0.98}$. Weighted instances were then generated using the final clause penalties from each of the characteristic runs. SAPS was run on each of the 25 weighted instances 250 times, and the median run from each weighted instance was identified. Those 25 medians formed a distribution, and basic descriptive statistics of that distribution (median, $q_{0.25}$, $q_{0.75}$) are presented above. For each weighted result the value shown is the ratio of the respective run-length statistic of the weighted instance to the median run-length of the unweighted instance. The procedure was then repeated on 25 instances where the weights were randomly permuted, and then finally on 25 instances with randomly generated weights.

timise the SAPS parameters (α,ρ,P_{smooth},wp) and used the default values $(1.3, 0.8, 0.05, 0.01)$. Generally, we measure the performance of an algorithm as the median or mean number of steps required to reach a solution.

As can be seen from the results in Table 1, the performance of SAPS on the SAPS generated weighted instances was not significantly better than the performance of SAPS on the original (unweighted) instances, and in some instances it was worse. Furthermore, we found no correlation between the length of the SAPS run from which the penalties were taken (short, average, long) and the hardness of the resulting weighted instance.

Next, we considered variants of the SAPS generated weighted instances in which the weights had been randomly permuted by assigning each weight to a randomly chosen clause. The results in Table 1 demonstrate that SAPS behaves similarly on the original SAPS generated instances and the permuted instances. This provides further evidence that the individual weights do not reflect specific knowledge about the respective clauses.

Finally, we looked at instances generated with completely random weights that were uniformly sampled at random from the interval (0,1], which we refer to as *randomly weighted instances*. In most of the experiments, we found no significant performance differences when we compared the performance results of SAPS on the SAPS generated weighted instances and the randomly weighted instances. A few minor exceptions, in which the randomly generated weights rendered the instance slightly harder than SAPS generated weights or unit weights, are seen in Table 1 (bw_large.a and ssa7552-038).

To further investigate this matter, we considered two larger SATLIB test-sets: flat100, a set of 100 randomly generated, hard, SAT-encoded graph colouring instances with 300 variables each, and uf100, a set of 100 instances from the solubility phase transition region of Random-3-SAT with 100 variables each. (Note that the flat100 instances contain a certain amount of structure induced by the specific mechanism used for generating random graph colouring instances and, more importantly, by the encoding into SAT.) In both cases, we repeated the previously explained procedures on all 100 instances of the respective test-set in order to determine whether for any of these instances using SAPS weights rendered them significantly easier than using randomly generated

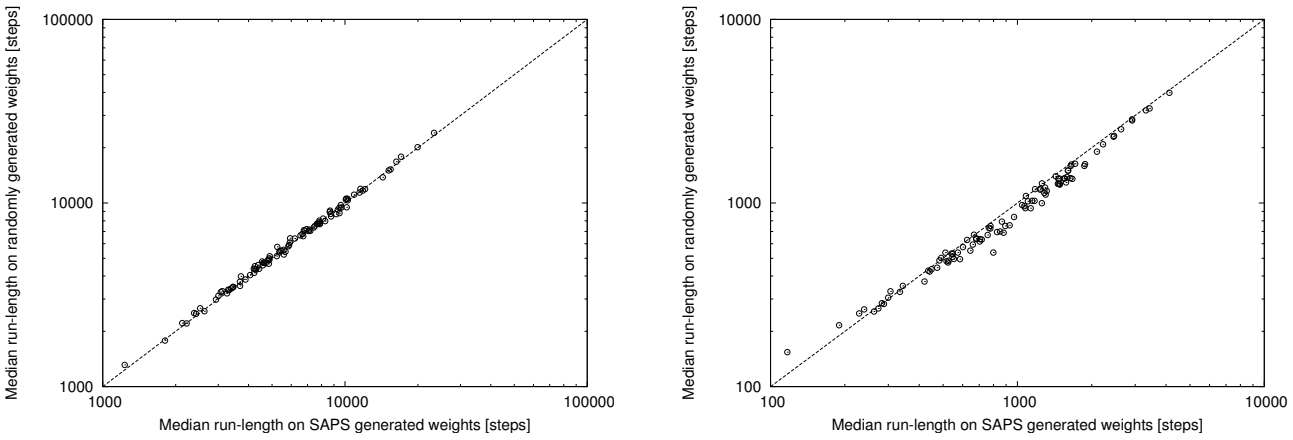

Figure 1: Comparison of SAPS performance on SAPS generated weighted instances and randomly weighted instances obtained for the SATLIB test-sets flat100 (left) and uf100 (right). For each test instance, SAPS was run 1000 times to obtain a run-length distribution. From that distribution, 25 characteristic runs were selected according to the quantiles $q_{0.02}$, $q_{0.06}$, ..., $q_{0.98}$. Weighted instances were then generated using the final clause penalties from each of the characteristic runs. SAPS was run on each of the 25 weighted instances 250 times, and the median run from each weighted instance was identified. The median of those medians was selected as the run-length shown above. The same procedure was carried out using 25 variants of each test instance with randomly generated weights.

weights. The results presented in Figure 1 show that this is not the case.

Although we have established that SAPS does not generally find the SAPS generated weighted instances easier, it may be the case that other algorithms can use these weights to their advantage. In particular, this could be the case for weighted static algorithms such as GWSAT [12]. Unlike DLS algorithms (such as SAPS), which dynamically change the clause penalties and may hence steer away from the original penalty values unless the instance can be solved relatively quickly, these static algorithms will continue to use the original weights and may thus be in a better position to exploit them. However, preliminary experiments with GWSAT (results not presented here) indicate that similar to SAPS, GWSAT does not appear to perform better when using SAPS generated weights as opposed to the unit weights that characterise the original, unweighted problem instances.

We also studied a slight variation of GSAT, RGSAT (Restarting GSAT), which uses the same best improvement search method underlying both GSAT as well as SAPS, but restarts the search from another random truth assignment whenever it cannot perform an improving search step, *i.e.*, whenever it encounters a local minimum or plateau of the given search landscape. Note that RGSAT has no plateau search capabilities, but can also never get stuck in a local minimum. The performance of RGSAT is quite poor, but it provides interesting insights into the hardness of SAT instances for simple local search methods. We conducted a series of experiments where we took snapshots of the warped landscape at every local minimum encountered within a SAPS search trajectory on SATLIB instance uf100-easy. On each of the corresponding weighted instances, we then ran RGSAT multiple times and we found that for RGSAT, these instances were becoming progressively easier to solve. However, we also observed that RGSAT's performance generally increases with the ruggedness of a given search landscape, where (following common practice) we consider a landscape more rugged if it contains fewer plateaus and more strict local minima. It is easy to see that during a SAPS run that starts from unit penalties, the landscape ruggedness can be expected to progressively increase, and hence it is not surprising that RGSAT finds the resulting warped landscapes easier to search than the landscape of the original, unweighted instance.

Overall, based on our empirical results, there is no evidence to support the hypothesis that the warped landscapes generated by SAPS are easier to search for any reasonably powerful SLS algorithm, and hence, that the

clause penalties determined over successful runs of SAPS reflect any general knowledge on how to solve the given problem instance more efficiently. Based on the similarity of the underlying penalty update mechanisms, we predict that the same result will apply to the warped landscapes obtained from other DLS algorithms, but testing this prediction will require some additional experimentation.

Based on these findings, it is worthwhile to revisit our understanding of the behaviour of DLS algorithms. The penalty update system of SAPS is based on two mechanisms: scaling and smoothing. Scaling clearly affects the mobility of the algorithm, *i.e.*, its ability to escape from local minima and other non-solution areas of the given search space. This was the original motivation for introducing the scaling mechanisms, and it seems to serve that purpose well. It has been suggested that DLS algorithms use scaling (or similar mechanisms) to 'fill in the holes' (*i.e.*, local minima) of the given landscape [8, 11].

While this analogy is appealing, it also turns out to be very deceiving. Consider a regular 3-SAT clause $c \equiv x_1 \vee x_2 \vee x_3$, and the impact of increasing the penalty of this clause on a given search landscape. Recall that the search landscape for a CNF formula F with n variables is an n-dimensional hypercube formed by the 2^n truth assignments of F. Any assignment a has n neighbours that differ from a in the truth value of exactly one variable. If the search is at a position in the space where clause c is unsatisfied ($x_1 = x_2 = x_3 = \bot$), then $n - 3$ neighbours will also leave c unsatisfied, and increasing the clause penalty of c will raise the level (*i.e.*, increase the evaluation function value) of those neighbours as well. So the level of the current assignment a will be raised relative to only three local neighbours. But recall that $1/8$ of the entire search space also has clause c unsatisfied, and so while locally the space has changed relative to three local neighbours, globally $2^{(n-3)}$ points in the space have been affected. Hence, by modifying a clause penalty, a DLS algorithm may effect a beneficial change in a very small local region of the search landscape, but at the global level, it is possible and rather likely that the side-effects of this local warping can be quite detrimental, *e.g.*, by giving rise to new local minima. (The possibility of such detrimental side-effects has also been hinted at in [8].)

The smoothing mechanism is required to compensate for these undesired, global side-effects. Essentially, smoothing helps the SAPS algorithm to 'forget' clause penalties that were important in order to escape from a local minima region, but are no longer helpful after the search has left that region. In some sense, scaling and smoothing try to balance short-term versus long-term memory effects of the search. Since in DLS algorithms like SAPS, long-term memory effects seem to be mostly undesired side-effects of crucially important short-term modifications of the search landscape, the main role of smoothing appears to be to limit the long-term effects of these local changes. (See also the discussion of long-term *vs* short term effects in [1].) Hence, when we re-use the penalties from a previous run of SAPS, we start in a completely different area of the search space, and the most recent local warping (which dominates the SAPS search behaviour) is of no benefit. Recent local changes from the previous search will be gradually 'forgotten' through smoothing, leaving the accumulated residual 'long term' memory (clause penalties) of the previous search, which according to our results does not appear to be effective in reducing search cost.

Overall, the results from our empirical investigation suggest that the penalty-based mechanisms used by SAPS and other DLS algorithms for warping the search landscape mainly serves as a diversification mechanism, which allows the search process to effectively avoid and overcome stagnation due to local minima and plateau regions while maintaining its heuristic guidance.

3 Random Decisions Need Not Apply

By definition and in practice, random decisions are an essential ingredient of stochastic local search and they are often crucial for achieving high performance in a robust way [3]. But it may be noted that the strongly randomised search mechanisms found in SAT algorithms such as GWSAT or WalkSAT [12], serve essentially the same purpose as the scaling and smoothing mechanism in SAPS: effective diversification of the search. Together with an earlier observation that high-performance DLS algorithms, after an initial search phase, often become nearly deterministic [10], this raises the question where and to which extent randomness is needed in the context of DLS algorithms.

We begin our investigation of this issue with the observation that after the search is initialised (at a randomly selected truth assignment) SAPS uses three types of random decisions:

1. *Random tie-breaking:* When two or more variables would give the identical best improvement when flipped, one of them is chosen at random.

2. *Random walk steps:* When a local minimum is encountered, a random walk is performed with probability wp.

3. *Probabilistic smoothing:* Scaling, which also occurs only when a local minimum is encountered, is followed by smoothing with probability P_{smooth}.

When examining the importance of each of these mechanisms in more detail, we found the following:

1. *Random tie-breaking:* On long search trajectories, after the algorithm has performed scaling and smoothing, most clause penalties become unique, and so the possibility of encountering a tie becomes increasingly unlikely.

2. *Random walk steps:* We have not encountered any empirical evidence that the random walk mechanism (originally inherited from the ESG algorithm) is important for the performance of SAPS, and in our experience setting $wp = 0$ does not degrade performance.

3. *Probabilistic smoothing:* The circumstances under which probabilistic smoothing will produce a substantially different search trajectory from periodic smoothing appear to be rare, and we have not seen any instances where this difference has significantly altered the algorithm's performance.

Together, these observations suggest that none of the random decisions discussed so far are crucial to the behaviour and performance of SAPS. To test this hypothesis, we designed a SAPS variant called SAPS/NR, which differs from SAPS only in the following three aspects:

1. *Deterministic tie-breaking:* Whenever a tie between variables occurs, the SAPS/NR algorithm deterministically chooses the variable with the lowest index value.

2. *No random walk steps:* The wp parameter is always set to zero, so that random walk steps are never performed.

3. *Periodic smoothing:* The probabilistic smoothing is replaced with deterministic periodic smoothing, where smoothing occurs every $1/P_{smooth}$ local minima.

At first glance, it may seem that SAPS/NR is completely deterministic, but we must emphasise that the initialisation of SAPS/NR is identical to the procedure in SAPS, and consequently the initial starting position for each run of SAPS/NR is completely random. The results of a simple experiment in which we measured the performance differences between SAPS and SAPS/NR are reported in Table 2; clearly, there is no significant difference between the behaviour of SAPS and SAPS/NR on our test instances. In particular, it is somewhat surprising that there appears to be no significant reduction in the variability of the run-time for the same problem instance.

To see how SAPS/NR behaves when the number of random decisions is reduced even further, we conducted an additional experiment in which we first reduced the total randomness to zero by initialising all variables to a fixed initialisation value and confirmed that all runs were identical. We then allowed one variable to be randomly initialised, then two variables, and so on. The results of this experiments, reported in Table 3, shows that even for as few as 32 random decisions (*i.e.*, 32 random variable initialisations), the behaviour of SAPS/NR is not substantially different from that of the fully randomised original SAPS algorithm. Even more surprisingly, even for 4 random decisions, as illustrated in Figure 2, the run-time, which can now take only 16 different values, often exhibits the variability that is observed for fully randomised SAPS as well as for all other state-of-the-art SLS algorithms for SAT [4].

Instance	SAPS		SAPS/NR	
	Mean	*c.v.*	Mean	*c.v.*
uf100-easy	102	0.75	103	0.70
uf100-hard	5,572	0.95	5,458	0.97
uf250-hard	296,523	0.98	282,668	1.02
uf400-hard	4,349,480	0.75	3,662,192	0.83
ais10	32,810	1.01	31,527	0.99
bw_large.a	3,374	0.85	3,245	0.81
bw_large.b	50,025	0.95	50,266	0.94
flat100-hard	35,124	1.02	33,519	0.98
logistics.c	12,873	0.76	12,458	0.83
ssa7552-038	4,460	0.44	4,399	0.41

Table 2: Performance comparison of SAPS and SAPS/NR. For each instance, SAPS and SAPS/NR were run 1000 times, and the mean and coefficient of variation (*c.v.*) of the run lengths are shown for each instance. The *c.v.* is calculated as the standard deviation divided by the mean ($\sigma/\bar{x}$). Note that *c.v.* $= 1$ characterises an exponential run-length distribution, which is typical for high-performance SLS algorithms for SAT.

# Random Decisions	flat100-hard		uf100-hard		bw_large.a	
	Mean	*c.v.*	Mean	*c.v.*	Mean	*c.v.*
0	24,198	0.00	4,764	0.00	2,160	0.00
1	28,491	0.51	5,316	0.44	3,016	0.40
2	30,315	0.68	4,834	0.67	2,989	0.59
4	33,966	0.90	5,408	0.88	3,254	0.75
8	33,467	0.98	5,413	0.94	3,218	0.81
16	34,074	0.99	5,560	0.97	3,285	0.84
32	34,113	0.99	5,476	0.96	3,276	0.83
64	34,099	0.99	5,435	0.96	3,265	0.84
# vars	33,769	0.98	5,458	0.97	3,245	0.81
SAPS	33,519	0.98	5,572	0.95	3,374	0.85
(# random)	(9,780)		(2,750)		(1,590)	

Table 3: The effect of varying the number of random decisions in the initialisation of SAPS/NR. For a SAPS/NR experiment with n variables and k random decisions, $(n - k)$ variables were randomly selected and assigned a truth value ($\top$,$\bot$) randomly, and then those variables were fixed for the remainder of that experiment. In each experiment, 1000 runs were conducted (250 for uf400-hard) and the run-length mean and *c.v.* were found. Each experiment was then repeated 100 times on every instance, where the randomly selected $(n - k)$ variables to fix were different for every experiment. The median results of those 100 experiments are presented above for each instance. For comparison, the corresponding values for SAPS are shown, in addition to the mean number of random decisions made in a run of SAPS.

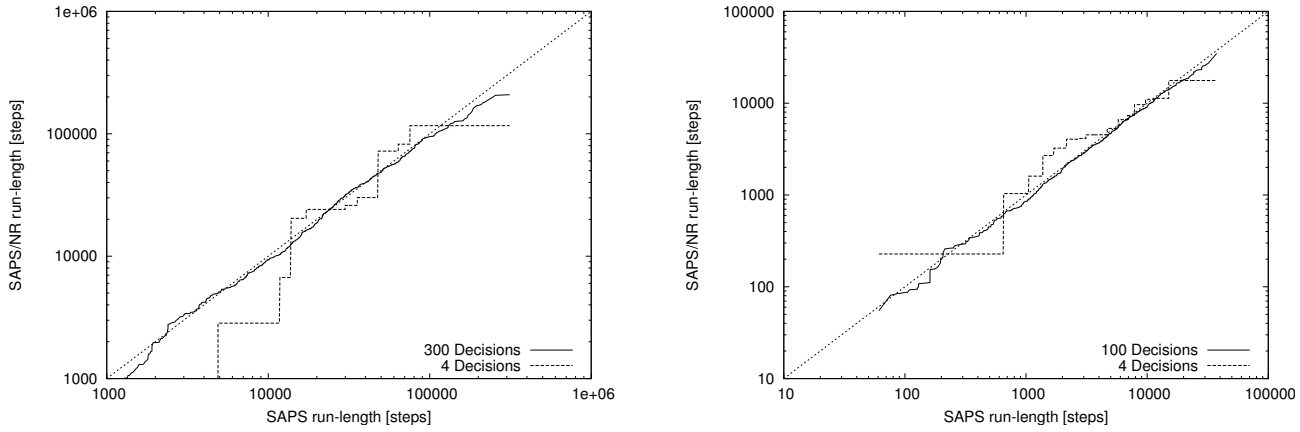

Figure 2: Quantile-quantile plot of the run-length distributions for SAPS/NR *vs* regular SAPS on SATLIB instances flat100-hard (left) and uf100-hard (right). The data is based on 1000 runs on each of SAPS, SAPS/NR (4 random decisions) and SAPS/NR (n random decisions, where n is the number of variables). For the 1000 runs with only 4 random decisions, $(n-4)$ variables were selected at random and assigned a truth value ($\top$,$\bot$) randomly, and then those variables were fixed for all 1000 runs.

This result appears to be in stark contrast with earlier observations by Gent and Walsh [2], who demonstrated that the initial starting position is typically not important for the performance of variants of GSAT. Indeed, it can be shown that for the regular SAPS algorithm, deterministic initialisation does not lead to significantly different behaviour, since apparently, the other random decisions are sufficient for ensuring sufficient diversification and mobility of the search process. The ESG algorithm also exhibits the same behaviour, where ESG has only two of the three sources of randomness described for SAPS (no probabilistic smoothing), but for a deterministic initialisation those two sources provide the same degree of variability as observed for regular ESG or SAPS. However, in the absence of other random decisions, the initial starting position is not only important, but also sufficient for achieving the same (desirable) behaviour as exhibited by SAPS, ESG and other high-performance SLS algorithms for SAT.

In this context, it is interesting to note that implementations of all SLS algorithms on traditional computers are in fact deterministic, since they use pseudo-random number generators instead of a true source of randomness as the basis for any "random" decision. Much in tune with current theoretical thinking, this suggests that true randomness is probably not needed for achieving good performance in SLS algorithms and other methods for solving hard combinatorial problems. Instead, the crucial role of pseudo-random decisions is to diversify the search process in a way that is independent from features of the given problem instance, in order to compensate for weaknesses in the heuristics that otherwise guide the search. Our results suggest that for DLS algorithms such as SAPS/NR, the complex interaction between the subsidiary greedy local search process and the effects of the warping of the search space accomplished by the scaling and smoothing mechanisms to a large extent fulfill the same role. This leads to a search process that is chaotic in the sense of a dynamical system that shows extremely sensitive dependence on its initial conditions. (Note that chaotic behaviour is often defined based on sensitive dependence on initial conditions.) We conjecture that this chaotic nature of the search propagates and amplifies the effects of the random initialisation through arbitrarily long search trajectories and thus reduces the need for further random decisions throughout the search process, while achieving the high mobility and diversification that is crucial for achieving the excellent performance of SAPS. Clearly, further investigation is required in order to validate this interpretation of the search process in SAPS/NR and SAPS.

4 Conclusions and Further Work

In this paper, we presented an initial investigation into the factors underlying the excellent performance of state-of-the-art dynamic local search algorithms for SAT. Our empirical results show that the warping of search landscapes effected by the scaling and smoothing mechanism of SAPS, a state-of-the-art dynamic local search algorithm for SAT, does not typically correspond to a global simplification of the given problem instance. The local warping achieved by scaling plays a crucial role in allowing the subsidiary greedy local search algorithm to escape from local minima of its evaluation function, but can have large, unwanted effects on the search landscape on a global scale. Therefore, it is crucial to provide a mechanism that allows for undoing these unwanted changes. In SAPS, this function is performed by the smoothing stage, which helps the algorithm 'forget' about recent scaling effects. Hence the primary effect of scaling and smoothing corresponds to a form of reactive short-term memory; however, because of the nature of the smoothing operation, SAPS shows long-term memory effects which typically do not appear to be helpful for the search process. Currently, we are extending the work presented here by trying to isolate the short-term from the long-term memory of the algorithm, as well as by further studying the nature of the dynamically changing landscapes in dynamic local search using standard search space analysis techniques. We believe that it may be possible to devise a modified long-term memory mechanism that, unlike the current smoothing scheme in SAPS, can be useful for rendering problem instances easier as the search progresses.

Furthermore, we have shown that, as a result of the properties of the scaling and smoothing mechanism, the role of randomness in SAPS is somewhat different from other SLS algorithms; in particular, after randomly initialising the search process, all further random decisions can be eliminated without any significant effect on the behaviour and performance of the algorithm. The resulting variant of SAPS, SAPS/NR, essentially shows chaotic behaviour in that the length of successful runs is extremely sensitively dependent on the initial truth assignment. We believe that this is a non-trivial effect of the complex interaction between the simple subsidiary greedy search algorithm and the dynamics of penalty updates. We are currently investigating this chaotic behaviour in more detail; furthermore, we are trying to find heuristics that allow us to initialise SAPS/NR in a way that biases it toward substantially shorter run-times. Particularly for structured problem instances, we are hopeful that such heuristic initialisation methods can be devised, leading to further improvements in the state-of-the-art in SAT solving.

References

[1] Jeremy Frank. Learning short-term clause weights for GSAT. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 384–389, 1997.

[2] Ian P. Gent and Toby Walsh. Towards an understanding of hillclimbing procedures for SAT. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, page 283, 1993.

[3] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, pages 661–666, 1993.

[4] Holger H. Hoos and Thomas Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[5] Holger H. Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT. In H. van Maaren I. P. Gent and T. Walsh, editors, *SAT20000: Highlights of Satisfiability Research in the year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*, pages 283–292. IOS Press, Amsterdam, The Netherlands, 2000.

[6] Frank Hutter, Dave A.D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *LNCS 2470: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 233–248. Springer Verlag, 2002.

[7] David Mitchell, Bart Selman, and Hector Levesque. Problem solving: Hardness and easiness - hard and easy distributions of SAT problems. In *Proceeding of the 10th National Conference on Artificial Intelligence (AAAI-92), San Jose, California*, pages 459–465. AAAI Press, Menlo Park, California, USA, 1992.

[8] Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the Eleventh National Conference in Artificial Intelligence (AAAI'93)*, pages 40–45, 1993.

[9] Dale Schuurmans and Finnegan Southey. Local search characteristics of incomplete SAT procedures. In *Proceedings of the Seventeenth National Conference in Artificial Intelligence (AAAI'00)*, pages 297–302, 2000.

[10] Dale Schuurmans, Finnegan Southey, and Robert C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 334–341, 2001.

[11] Bart Selman and Henry A. Kautz. Domain-independant extensions to GSAT : Solving large structured variables. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 290–295, 1993.

[12] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, Seattle, 1994.

[13] Dave A.D. Tompkins and Holger H. Hoos. Scaling and probabilistic smoothing: Dynamic local search for unweighted MAX-SAT. In *LNAI 2671: Proceedings of the 16th Conference of the Canadian Society for Computational Studies of Intelligence (AI 2003)*, pages 145–159. Springer Verlag, 2003.

[14] Zhe Wu and Benjamin W. Wah. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proceedings of the Seventeenth National Conference in Artificial Intelligence (AAAI'00)*, pages 310–315, 2000.