ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-300

A Study of Local Minimum Avoidance Heuristics for SAT

Thach-Thao Duong [1] and **Duc Nghia Pham** [1] and **Abdul Sattar** [1]

Abstract.

Stochastic local search for satisfiability (SAT) has successfully been applied to solve a wide range of problems. However, it still suffers from a major shortcoming, i.e. being trapped in local minima. In this study, we explore different heuristics to avoid local minima. The main idea is to proactively avoid local minima rather than reactively escape from them. This is worthwhile because it is time consuming to successfully escape from a local minimum in a deep and wide valley. In addition, revisiting an encountered local minimum several times makes it worse. Our new trap avoidance heuristics that operate in two phases: (i) learning of pseudo-conflict information at each local minimum, and (ii) using this information to avoid revisiting the same local minimum. We present a detailed empirical study of different strategies to collect pseudo-conflict information (using either static or dynamic heuristics) as well as to forget the outdated information (using naive or time window smoothing). We select a benchmark suite that includes all random and structured instances used in the 2011 SAT competition and three sets of hardware and software verification problems. Our results show that the new heuristics significantly outperform existing stochastic local search solvers (including Sparrow2011 - the best local search solver for random instances in the 2011 SAT competition) on all tested benchmarks.

1 INTRODUCTION

Stochastic local search (SLS) for satisfiability (SAT) has successfully been applied to solve a wide range of problems, including random, planning, and hardware and software verification problems. Since the introduction of GSAT [11], many efficient SLS algorithms for SAT have been developed, including VW2 [10], PAWS [13], G^2WSAT [5], gNovelty$^+$ [8], TNM [15], and Sparrow [1].

Despite this significant progress, SLS solvers still have many shortcomings and are unable to compete against systematic algorithms in solving structured SAT problems as demonstrated through the series of SAT competitions[2]. One major limitation of SLS solvers is their poor behaviour on local minima. Because structured problems have tighter and more connected constraints than random instances, it is easier for SLS solvers to get trapped and also becomes harder for them to escape.

Contemporary local search methods predominantly use a random walk or a Novelty walk to escape from a local minimum [7, 5, 8]. Weighting techniques, such as variable weighting [10] and clause weighting [13, 8], have also been used to mitigate search stagnation. In addition, a switching heuristic was proposed to alternatively select candidate moves using either (i) non-weighting, (ii) variable or (iii) clause weighting schemes [16]. Variable weight distribution has also been used in TNM [15] to regulate its two adaptive noise heuristics. However, TNM does not directly use variable weighting to select variables. Recently, Sparrow [1], a variant of gNovelty$^+$ [8], uses a dynamic scoring function instead of the Novelty walk (as in gNovelty$^+$) to escape from local minima. Sparrow2011 [2], a new efficient implementation of Sparrow, won the Gold medal for random SAT category in the 2011 SAT competition. To the best of our knowledge, clause weighting and variable weighting have not been combined into a single scheme for variable selection.

In this study, we explore different heuristics to avoid local minima. The main idea is to proactively avoid local minima rather than reactively escape from them. This is worthwhile because it is time consuming to successfully escape from a local minimum in a deep and wide valley. In addition, revisiting an encountered local minimum several times makes it worse. The rest of this paper is organised as follows: in Section 2, we briefly discuss the motivation for this study. We assume that some most recently flipped variable prior to the occurence of a trap is the pseudo-conflict causing that trap. We then present in detail how to collect pseudo-conflict information as well as how to utilise this knowledge to avoid local minima. We also discuss various smoothing heuristics to discard outdated information. We then show how all these heuristics can be employed to improve the performance of gNovelty$^+$. In Section 3, we present a detailed empirical study of the impact of different combinations of our heuristics on gNovelty$^+$'s performance. Our benchmark suite includes all random, crafted and application instances used in the 2011 SAT competition as well as three sets of hardware and software verification problems. The results show that our approach significantly outperforms existing SLS solvers (including Sparrow2011 - the best SLS solver for random SAT instances in the 2011 SAT competition) on all tested benchmarks. Finally, we summarise our conclusions and outline our plan for future work in Section 4.

2 TRAP PREVENTION STRATEGY

2.1 Motivation

Complex structured problems often contain many highly connected dependent variables and hence present many traps (or local minima) to a SLS solver. These traps are normally large plateaus in a deep valley, making it difficult for a SLS solver to escape from them. Therefore, it is highly desirable to adopt intelligent heuristics that can predict potential traps. Such smart heuristics could help SLS solvers tremendously in avoiding local minima.

There are many techniques in the literature for handling local minima. Examples include the variants of random walks and clause weighting. The random walk strategy from the WalkSAT family [7] has been the most popular treatment for search stagnation. However,

[1] Queensland Research Laboratory, NICTA Ltd and Institute for Integrated and Intelligent Systems, Griffith University, email : {thao.duong, ducnghia.pham, abdul.sattar} @ nicta.com.au
[2] http://www.satcompetition.org

they are reactively treatments once stagnation occurs. Clause weighting algorithms (e.g. PAWS [13], gNovelty$^+$ [8], Sparrow [1]) add weights to unsatisfied clauses and subsequently change the search landscape to escape from local minima. In contrast, the variable weighting scheme (e.g. VW2 [10]) favors the least flipped variable when breaking ties amongst variables with the same WalkSAT scores. An alternative method to weighting techniques is the Tabu heuristic [12]. This method prevents the search from returning back to recently visited areas by abstaining from selecting a predefined number of recently flipped variables.

Despite being quite successful in handling traps, existing approaches do not utilise an effective learning scheme to proactively help the search avoid future local minima. The random walk strategy simply relies on randomisation to get out of traps. Recently, modern state-of-the-art SLS solvers implement a (Adaptive)Novelty-based walk instead of a random walk to escape from local minima. By alternating between the best and second best scored variables based on a noise probability, this method makes a compromise between greediness and exploration. However, neither of these strategies is able to learn from the current or previous encountered traps. Amongst existing approaches, clause weighting schemes can be seen to learn about visited local minima. In particular, a clause with a high frequency of being unsatisfied at local minima is preferred to be solved in the next search steps. Inspired by that scheme, we aim to develop a move selection mechanism that gives priority to variables that regularly cause stagnations.

2.2 Trap prevention strategy

The purpose of our trap prevention approach is to intelligently assist local search algorithms in avoiding potential traps by exploiting information gathered from previous encountered local minima. In general, our trap prevention strategy operates in two phases: a learning phase and a prevention phase.

The first phase is invoked every time the search encounters a local minimum. Here, it will try to derive and learn as much useful information as possible from the cause of this stagnation. We assume that a local minimum is the result of some recent conflicted variable assignments made immediately before that local minimum. As opposed to conflict-driven clause-learning systematic search, it is challenging to derive the exact conflict that leads to this trap. Hence, we approximate the *cause-of-stagnation* (or *pseudo-conflict*) being responsible for that local minimum. In detail, we define the cause-of-stagnation as the sequence of k most recently flipped variables leading to that stagnation. The tenure k is used to impose the maximum size of this pseudo-conflict. It is worthy to note that the value of k should not be set too large to prevent learning incorrect information. We adopt the variable weighting approach [10] as a simple way to store and exploit the learnt cause-of-stagnation. We associate a $stagnation\_weight$ to each variable to represent the frequency that variable is involved in a cause-of-stagnation. Intuitively, flipping a variable with a high stagnation_weight value is more likely to lead a local minimum. Thus in the second phase, we use this intuition to proactively prevent the search from falling into old traps. In particular, we alter the move selection to give higher priority to variables with lower stagnation_weight values.

Algorithm 1 outlines our pseudo-conflict learning (PCL) approach to learn and update the stagnation_weight of each variable. Lines 2-9 illustrate how a cause-of-stagnation can be derived and learnt. Once the cause-of-stagnation has been derived, the stagnation_weight of each variable involved in that cause-of-stagnation is increased by 1 (line 10). Finally, Lines 11-19 demonstrate how to reduce the stagnation_weight of certain variables in order to appropriately discard outdated information.

Algorithm 1: PCL(k, s, T, tp, H, it, Q)

Input : the PCL tenure k, the learning strategy s, the time window size T, the naive smoothing probability tp, the history H of flipped variables, the current iteration it, and the queue Q of recent causes-of-stagnation.

```
1  cause_of_stagnation = ∅;
2  if s is 'static' then
3  |  for i = 0; i < k; i++ do cause_of_stagnation += {H[it − i]};
4  else                                              /* s is 'dynamic' */
5  |  most_recently_flipped_var = H[it];
6  |  cause_of_stagnation += {most_recently_flipped_var};
7  |  for i = 1; i < k; i++ do
8  |  |  if H[it − i] == most_recently_flipped_var then break;
9  |  |  cause_of_stagnation += {H[it − i]};
10 foreach var ∈ cause_of_stagnation do stagnation_weight[var]++;
11 if T > 0 then                          /* use time window smoothing */
12 |  if Q.size() == T then
13 |  |  oldest_cause_of_stagnation = Q.pop();
14 |  |  foreach var ∈ oldest_cause_of_stagnation do
   |  |  stagnation_weight[var]- -;
15 |  Q.push(cause_of_stagnation);
16 if tp > 0 then                              /* use naive smoothing */
17 |  if within the probability tp then
18 |  |  foreach var having its stagnation_weight[var] > 1 do
19 |  |  |  stagnation_weight[var]- -;
```

2.3 Learning the cause-of-stagnation

In Algorithm 1, we propose two different strategies to derive a cause-of-stagnation when encountering a local minimum: "static" or "dynamic". The first strategy (lines 2-3) presumes that all causes-of-stagnation have the same *static* fixed size which is the tenure k. As a result, deriving a static cause-of-stagnation simply requires keeping track of the last k flipped variables.

In contrast, our second strategy (lines 4-9) *dynamically* determines the size of a cause-of-stagnation according to search history. In detail, let lv be the latest flipped variable. A dynamic cause-of-stagnation is defined as a reversed sequence of the current search history up to the first re-occurrence of lv. Note that the size of a dynamic cause-of-stagnation is capped at k if no duplication of lv has been found. Notice that the use of the tenure k is inspired by Tabu search [12]. However, a Tabu search completely forbids tabooed variables from being selected. On the other hand, our approach allows a recently flipped variable (even the most recently flipped variable) to be selected as the next move if it has the lowest stagnation_weight.

2.4 Forgetting outdated causes-of-stagnation

As the search progresses and encounters several local minima, the collection of variable stagnation_weights may not correctly reflect the causes of stagnation. In other words, the stagnation_weights of many variables may contain information gathered from the derived causes of now obsolete traps. Thus, it is necessary to forget such outdated causes-of-stagnation by selectively reducing the stagnation_weights of certain variables. In Algorithm 1, we outline two different smoothing heuristics that can be used together or individually.

Our first smoothing heuristic is inspired by gNovelty$^+$ [8]. This *naive* smoothing heuristic reduces the stagnation_weights of all weighted variables (i.e. their stagnation_weights are greater than 1) by 1 (lines 16-19). In contrast, the second smoothing heuristic (lines 11-15) mimics a mechanism in which a window is shifted alongside

the search. This *time window* smoothing method aims to restrict the effect of the learnt information to within a short-term period. In other words, the derived information within the time window is accumulated and saved in the stagnation_weights while information gathered outside that time window is discarded. The time window size is specified by a parameter T. As outlined in Algorithm 1, we maintain a queue of T recent causes-of-stagnation. As the time window shifts, once a cause-of-stagnation is out of the specified window, information learnt from that cause-of-stagnation is discarded. Indeed, the stagnation_weights of variables involved in the recent obsolete cause-of-stagnation are decreased by 1 (line 14).

Algorithm 2: gNovelty$^+$PCL(k, s, T, tp, sp)

Input : A formula Θ, a random walk probability wp, a clause weight smoothing probability sp, the PCL tenure k, the learning strategy s, the time window size T, and the naive smoothing probability tp.
Output: Solution α (if found) or TIMEOUT

```
1  randomly generate a candidate solution α;
2  clear the queue Q of recent causes-of-stagnation;
3  clear the history H of flipped variables;
4  initialise stagnation_weight of all variables to 0;
5  set the iteration counter it = 0;
6  while not timeout do
7      if α satisfied the formula Θ then return α ;
8      if within the random walk probability wp then
9          randomly pick a variable v in an unsatisfied clause;
10     else
11         if there exist promising variables then
12             select the most promising variable v, break tie by selecting one
               with the least stagnation_weight, break further tie randomly;
13         else                          /* encounter a local minimum */
14             update (and smooth within probability sp) all clause_weights;
15             PCL (k, s, T, tp, H, it, Q);
16             perform an AdaptiveNovelty walk to select a variable v in an
               unsatisfied clause, break tie by selecting one with the least
               stagnation_weight, break further tie randomly;
17     update the candidate solution α with the flipped variable v;
18     update the search trajectory H[it] = v;
19     it++;
20 return TIMEOUT;
```

2.5 The new gNovelty$^+$PCL variants

In this section, we describe how our approach can be combined with gNovelty$^+$, a clause weighting SLS solver. The new gNovelty$^+$PCL algorithm is illustrated in Algorithm 2. Initially, all stagnation_weights are set to 0; and the search history H of flipped variables and the queue Q of recent causes-of-stagnation are emptied. At the beginning of the search, gNovelty$^+$PCL operates in the same manner as gNovelty$^+$, relying mainly on the clause_weights of variables to select the next move. Conceptually, a *promising* variable is one with an improving clause weighted score (i.e. if it is flipped the total weights of all unsatisfied clauses is reduced). For a detailed description of promising variables, please see [8]. Whenever a local minimum is encountered (i.e. there is no promising variable), clause weights are updated and smoothed if required. Our pseudo-conflict learning mechanism (Algorithm 1) is called to derive and learn the approximate cause-of-stagnation (line 15). Subsequently, gNovelty$^+$PCL utilises the learnt knowledge saved in stagnation_weights to select its moves. In detail, ties are broken by selecting the variables with the lowest stagnation_weight rather than choosing the least recently flipped variable. Further ties are broken by randomly selecting a variable. This change is reflected in lines 12 and 16 in Algorithm 2. Once the chosen variable is flipped, the search history H is also updated to keep track of the k recently flipped variables (line 18).

As illustrated in Algorithm 2, eight different variants gNovelty$^+$PCL can be implemented by varying the combination of learning strategies and smoothing heuristics. Note that either static or dynamic learning strategy can be specified by parameter s. The time window smoothing can be turned off by setting the window size $T = 0$ whilst the naive smoothing can be turned off by setting the probability $tp = 0$. The list of these variants together with their enabled options are described in Table 1.

Table 1. Eight different PCL variants

Learning strategy	Smoothing heuristic			
	none	naive	time window	time window + naive
static	Tp	Tps	Tw	Tws
dynamic	Tpc	Tpcs	Twc	Twcs

3 EXPERIMENTS

We firstly compare gNovelty+PCL with other SLS solvers on the ternary chain problem [9]. This problem has a unique solution and has been proven to be very hard for SLS solvers (as it contains many traps) [9, 17, 10]. This experiment enables us to evaluate how effective our approach is in avoiding traps. We then present a detailed study of 8 different variants of gNovelty+PCL on all the benchmark instances used in the 2011 SAT competition. In addition, we also added three other benchmark problems from hardware and software verification applications.

3.1 Experiments on ternary chains

The ternary chain problem with n variables is defined as follows:

$$(x_1) \wedge (x_2) \wedge (x_1 \wedge x_2 \rightarrow x_3) \wedge \ldots \wedge (x_{n-2} \wedge x_{n-1} \rightarrow x_n)$$

This artificial problem deliberately simulates chains of variable dependency found in structured problems. It is employed to study the diversification capability of local search algorithms in [9], due to the fact that this problem has only one solution where all variables are assigned to *true*. In other words, flipping any variable from *true* to *false* moves further away from the solution. This property makes it easy for local search algorithms to become trapped. For this reason, this problem has been highly recommended to evaluate the performance of SLS solvers [9, 17, 10].

In this experiment, we compared the performance of gNovelty$^+$PCL on ternary chain instances against that of existing popular SLS solvers: RoTS (Tabu search) [12], PAWS [13], VW2 [10], AdaptG^{2}WSAT0 [6], TNM [15], gNovelty$^+$ [8], and Sparrow2011 [2]. Figure 1 plots the performance of these solvers on ternary chain instances with size ranging from 10 to 100 in steps of 5. Each solver was run 100 times on each instance and each run was limited to 20 seconds. Default settings were used for all solvers. The gNovelty$^+$PCL version tested here is the Tp variant with static learning strategy ($k = 20$) and no smoothing. The experiment was conducted on the NICTA HPC cluster of quad-core Intel Xeon CPUs @2.0GHz.

As shown on Figure 1, VW2, gNovelty$^+$ and gNovelty$^+$PCL dramatically outperform other solvers. They are the only three solvers that can successfully solve all instances within the time limit. This

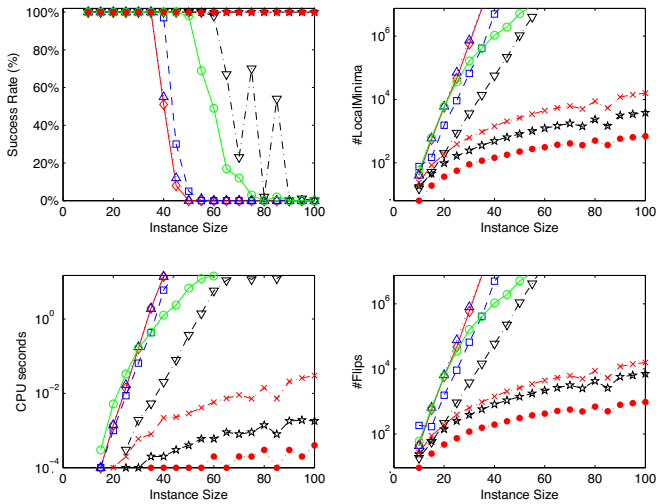

Figure 1. Experiments on ternary chain instances. The pairs of solvers and line markers used in this figure are as follows: RoTS - □, AdaptG^{2}WSAT0 - △, TNM - ◇, VW2 - ×, PAWS - ▽, gNovelty$^+$ - ★, Sparrow2011 - ○, and gNovelty$^+$PCL - ●.

again confirmed the observation from [10] that weighting schemes have the advantage of reducing stagnation. Amongst those three solvers, VW2 is the worst performer in terms of CPU times used, the number of flips performed and the number of local minima encountered. This is because VW2 relies only on its variable weighting scheme whilst gNovelty$^+$ (the second best) uses both clause weights and AdaptiveNovelty walks to handle traps. Nevertheless, our results clearly show the superiority of our PCL approach in avoiding traps. Indeed, the performance of gNovelty$^+$PCL is at least two times better than gNovelty$^+$ on all the evaluation criteria except the success rate where both achieved 100%.

3.2 Experiments on random and structured instances

In this study, we ran the eight different variants of gNovelty$^+$PCL (listed in Table 1) on a large benchmark suite comprised of random and structured instances. This aims to thoroughly examine the effectiveness of our PCL approach as well as evaluate the effects of different PCL options on gNovelty$^+$ across a wide range of different problem types. For this purpose, our benchmark suite comprises of three different real-world verification problems as well as all the random, crafted and application instances used in the 2011 SAT competition.

The parameter configuration for each gNovelty$^+$PCL variant was optimised by ParamILS [4], a local search optimisation tool for parameterised algorithms. For each variant, six independent training sections within {0.5; 1; 1.5; 2; 2.5; 3} days were performed. The resultant configurations were then validated to obtain the best settings reported in Table 2. Other settings for gNovelty$^+$PCL that were not listed here are kept the same as in gNovelty$^+$. The training and validation were conducted on the Griffith University's Gowonda HPC cluster equipped with Intel(R) Xeon(R) CPUs X5650 2.67GHz. For other solvers included in the following experiments, their best parameter settings (found in their submission to SAT competitions) are used unless otherwise stated.

Table 2. Parameters settings for gNovelty$^+$PCL variants

Benchmark set	Learning strategy	Tp/Tpc		Tw/Twc			Tps/Tpcs			Tw/Twc			
		k	sp	k	T	sp	k	tp	sp	k	T	tp	sp
cbmc	static	30	0.4	10	300	0.4	15	0.1	0.4	20	200	0.2	0.4
	dynamic	15	0.4	15	300	0.35	20	0.2	0.4	15	100	0.4	0.35
swv	static	10	0.05	15	250	0.05	20	0.3	0.2	30	300	0.4	0.05
	dynamic	30	0.05	10	300	0.05	30	0.15	0.1	30	150	0.1	0.05
sss-sat-1.0	static	10	0.05	20	100	0.5	30	0.25	0.05	15	300	0.05	0.05
	dynamic	15	0.05	10	100	0.05	10	0.05	0.05	10	150	0.05	0.05
SAT 2011	static	25	0	15	150	0	15	0.4	0	25	200	0.35	0
	dynamic	25	0	15	250	0	25	0.3	0	25	100	0.3	0.05

3.2.1 Experimental results: verification benchmarks

Our verification benchmark set consists of three real-world problem: cbmc, swv, and sss-sat-1.0. The first two problem sets are software verification instances comprised of (i) 39 cbmc instances generated by a bounded model checking tool, and (ii) 75 swv instances generated by the CALYSTO checker.[3] The third one contains 100 instances that encodes the verification of super-scalar microprocessors.[4] Note that even though these instances can be easily solved by systematic solver such as PicoSAT [3], they remain a remarkable challenge for SLS solvers. In addition to eight different gNovelty$^+$PCL variants, we also included in this experiment the following solvers: gNovelty$^+$ and VW2 (that can solve all ternary chain instances in the previous experiment); and TNM and Sparrow2011 (the respective winner of the 2009 and 2011 SAT competitions). For each instance, we ran each solver 50 times with a time limit of 600 seconds for each run. All experiments were conducted on the Griffith University's Gowonda HPC cluster equipped with Intel(R) Xeon(R) CPUs X5650 2.67GHz. The results are reported in Table 3.

In general, all gNovelty$^+$PCL variants performed consistently better than other solvers on these three problem sets. In particular, all variants achieved 100% success rate on the cbmc and sss-sat-1.0 problem sets. This is a significant achievement as other solvers were unable to solve all these instances. Indeed, their best results on these two problem set are 92% (by TNM) on cbmc and 50% (by gNovelty$^+$) on sss-sat-1.0. Furthermore, the success rate of each gNovelty$^+$PCL variant is about two times those of other solvers on the swv data set. In fact, it was discovered that 50% of swv instances cannot be solved consistently by SLS solvers [14]. In comparison amongst gNovelty$^+$PCL variants, those without naive smoothing generally gave better results. Moreover, gNovelty$^+$Tpc performed the best on the cbmc and swv sets; while gNovelty$^+$Tw performed the best on sss-sat-1.0.[5]

3.2.2 Experimental results: SAT-2011 benchmarks

In this study, we compared the performance of gNovelty$^+$PCL variants against its original gNovelty$^+$ on the random, crafted and application instances from the SAT-2011 benchmarks. We also included Sparrow2011, the best SLS solver for random SAT instances in the SAT-2011 competition.[6] For each instance, we ran each solver 10 times with a time limit of 600 seconds for each run. All experiments

[3] These instances are available at http://www.cs.uwaterloo.ca/~dtompkin/papers/sat10-dave-instances.zip
[4] Available at http://www.miroslav-velev.com/sat_benchmarks.html
[5] We used the success rate as the main evaluation criterion, breaking ties by using the CPU times.
[6] No SLS solver was ranked in the Top-3 for the crafted and application categories in the SAT-2011 competition

Table 3. Experimental results on verification benchmarks. For each problem set, the first row reports the success rate and the other rows report the average of the min, median, and max CPU times (in seconds) to solve an instance in that set. Here, an instance is considered consistently solvable by a solver if its successful rate is greater than or equal to 50%. gNovelty$^+$ was ran with its best settings for structured instances [8].

Problems	TNM	VW2	Sparrow2011	gNovelty$^+$	gNovelty$^+$Tp	gNovelty$^+$Tpc	gNovelty$^+$Tw	gNovelty$^+$Twc	gNovelty$^+$Tps	gNovelty$^+$Tpcs	gNovelty$^+$Tws	gNovelty$^+$Twcs
cbmc (39)	92%	31%	51%	85%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
	13.399	362.911	119.692	54.573	0.059	**0.046**	0.062	0.095	0.094	0.047	0.047	0.094
	76.599	439.128	384.359	247.997	1.453	**1.351**	1.842	2.258	1.954	1.782	1.393	2.724
	164.602	498.058	499.474	382.512	6.797	**6.457**	7.729	14.295	8.077	7.384	8.730	30.471
swv (75)	23%	23%	21%	25%	48%	**49%**	48%	48%	47%	47%	45%	48%
	464.005	464.133	413.379	409.451	304.693	304.618	**303.952**	304.774	292.073	292.919	299.504	299.995
	464.013	466.002	486.949	459.097	334.323	335.663	326.953	**326.605**	351.524	338.161	341.782	333.747
	464.037	474.762	519.651	464.131	360.534	365.093	360.447	**357.854**	373.035	367.581	371.332	361.988
sss-sat-1.0 (100)	18%	24%	7%	50%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
	350.904	318.728	484.792	162.757	0.195	0.200	0.179	**0.159**	0.185	0.170	0.161	0.186
	517.709	481.357	572.993	348.512	2.721	2.755	**2.095**	2.266	2.733	2.227	2.365	2.449
	546.742	523.310	588.703	427.622	20.285	19.037	18.216	15.516	17.498	**13.664**	17.408	16.327

were conducted on the Griffith University's Gowonda HPC cluster equipped with Intel(R) Xeon(R) CPUs X5650 2.67GHz. The results were summarised in Table 4 and depicted graphically in Figure 2. Here, we reported the results for instances that can be solved by all solvers in at least one run. Thus, the number of instances in Table 4 may not reflect the original number of instances from the SAT-2011 benchmark suite. In addition, because of the big difference in terms of size and difficulty amongst random instances, we divided the results of these instances into two groups: medium and large sized instances.

As illustrated in Figure 2, gNovelty$^+$ performed better than Sparrow2011 on application and crafted instances. However, it lost to Sparrow2011 on random instances. The detailed results in Table 4 confirmed our observations but also revealed that gNovelty$^+$ won only on crafted instances. In fact, its success rate was only half of that of Sparrow2011 on application instances. Nevertheless, our PCL approach significantly improved the performance of gNovelty$^+$ across the whole benchmark suite. Figure 2 showed that all gNovelty$^+$PCL variants greatly outperformed Sparrow2011 on all three problem categories. Indeed, gNovelty$^+$Tp achieved 88% and 85% success rates respectively on the application and crafted instances; whereas gNovelty$^+$ could only solve 12% and 62%, and Sparrow2011 only solved 25% and 56%. Table 4 also showed that gNovelty$^+$Tw solved medium sized random instances at least two times faster than Sparrow2011[7]. Although there was not much difference in the CPU time measures between the two solvers on large random instances, gNovelty$^+$Tw achieved a better success rate than Sparrow2011 on these instances. Note that gNovelty$^+$ was not able to consistently solve a single large random instance. Overall, there were always at least three variants of gNovelty$^+$PCL that performed better than Sparrow2011 in terms of success rate. In terms of CPU times, the majority of gNovelty$^+$PCL variants were much faster than Sparrow2011.

Comparing amongst eight gNovelty$^+$PCL variants, gNovelty$^+$Tp performed the best on structured instances (combining both application and crafted sets). However, gNovelty$^+$Tw was the best performer on crafted as well as both medium and large random instances according to Table 4. With regard to smoothing effects, our results do not support the use of naive smoothing together with static learning on structured problems. When combined with dynamic learning naive smoothing variants were slightly faster on structured problems than their corresponding variants without naive smoothing (although

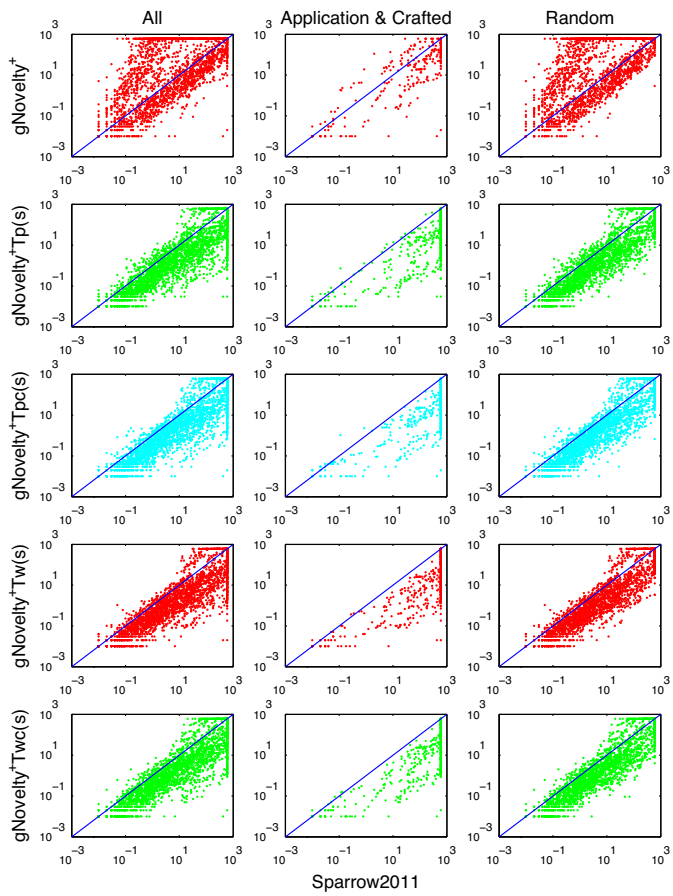

Figure 2. A CPU time comparison of gNovelty$^+$PCL variants and gNovelty$^+$ against Sparrow2011 on the SAT-2011 benchmarks.

each pair of variants achieved a similar success rate). On random instances, Table 4 supported the use of either naive or time window smoothing heuristics. In fact, the best results on random instances were achieved with time window smoothing. However combining naive with time window smoothing on random instances makes it worse (e.g. gNovelty$^+$Tws and gNovelty$^+$Twcs both performed

[7] Both solvers solved all the medium sized random instances.

Table 4. Experimental results on the SAT-2011 benchmarks. For each problem set, we report the success rate and the average of the min, median, and max CPU times (in seconds) to solve an instance in that set. Here, an instance is considered consistently solvable by a solver if its successful rate is greater than or equal to 50%. gNovelty$^+$ was ran with its best settings for structured instances [8].

Problems	Sparrow2011		gNovelty$^+$		gNovelty$^+$Tp		gNovelty$^+$Tpc		gNovelty$^+$Tw		gNovelty$^+$Twc		gNovelty$^+$Tps		gNovelty$^+$Tpcs		gNovelty$^+$Tws		gNovelty$^+$Twcs	
Application		461.824		464.149		**46.671**		44.099		262.256		229.016		149.478		125.454		170.588		310.411
(8)	25%	515.799	12%	527.431	**88%**	**186.717**	75%	236.130	50%	385.124	50%	373.847	38%	409.214	50%	392.688	50%	367.528	50%	324.933
		543.695		527.503		**435.348**		403.632		481.179		479.386		465.744		468.523		474.327		456.077
Crafted		237.720		202.872		62.826		72.894		**51.703**		67.631		58.480		63.692		51.286		87.756
(82)	56%	285.159	62%	251.332	**85%**	113.645	84%	123.203	**85%**	**109.709**	84%	114.586	83%	123.859	84%	120.483	84%	122.356	73%	174.103
		338.434		285.784		156.681		157.404		**150.945**		156.552		161.427		165.790		155.360		199.108
Random		5.049		130.371		4.970		3.397		1.512		3.851		3.999		2.234		1.407		**1.376**
Medium	**100%**	40.671	75%	184.698	96%	38.726	97%	35.386	**100%**	14.864	**100%**	**13.487**	**100%**	14.934	**100%**	17.717	98%	24.391	**100%**	13.893
(201)		99.540		241.121		100.166		100.289		**40.142**		46.103		46.929		53.706		71.666		54.821
Random		437.202		576.600		439.747		439.215		325.823		**313.537**		330.508		332.950		382.080		401.903
Large	20%	509.487	0%	600.000	9%	560.437	12%	560.193	**28%**	**504.191**	25%	515.067	27%	509.195	20%	528.470	17%	540.270	16%	545.306
(64)		**555.076**		600.000		594.281		591.126		584.112		581.770		583.585		580.986		591.395		593.068

poorly than to gNovelty$^+$Tw and gNovelty$^+$Twc). With regard to learning strategies, there was a winner for each smoothing option on a particular problem set. However, there was no overall clear winner between the two learning strategies.

4 CONCLUSION AND FUTURE WORK

In conclusion, our experiments showed that across all benchmark instances proactively avoiding local minima was substantially better than reactively escaping from them. Indeed, gNovelty+PCL solved more than twice the number of instances that could be solved by its original, gNovelty+. In addition, gNovelty+PCL was usually more than an order of magnitude faster than gNovelty+. Similar conclusions also hold when comparing the performance of gNovelty+PCL variants against other SLS solvers, including Sparrow2011 - the best solver for random SAT instances in the 2011 SAT competition.

Our experiments revealed that there was not much difference in the performance using either the static or dynamic learning strategies. This observation was true for all variants of smoothing heuristics on both the structured and random instances. Therefore, we recommend the use of static pseudo-conflict learning strategy to avoid unnecessary overheads. With respect to retaining the learnt knowledge, our results showed that gNovelty+PCL variants (that kept a long-term memory of pseudo-conflicts) outperformed other variants (that performed smoothing and only memorised recent pseudo-conflicts) on structured SAT-2011 instances. However, the observation was reversed for random instances. One reason for this reverse effect is that random instances are less constrained than structured instances, and hence there is virtually no connection between two pseudo-conflicts that are far apart. Therefore, it is better to maintain that knowledge for a short time when solving random instances.

Encouraged by these results, we plan to explore alternative ways to learn different information during a local search in order to predict and prevent SLS SAT solvers from local minima. In addition, we intend to investigate the potential of automatically adjusting our parameters during the search. In the near future, we plan to thoroughly evaluate the impact of our heuristics over a wider range of SLS solvers.

ACKNOWLEDGEMENTS

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

REFERENCES

[1] Adrian Balint and Andreas Fröhlich, 'Improving stochastic local search for SAT with a new probability distribution', in *SAT*, pp. 10–15, (2010).
[2] Adrian Balint, Andreas Fröhlich, Dave A.D. Tompkins, and Holger H. Hoos, 'Sparrow2011', in *Booklet of SAT-2011 Competition*, (2011).
[3] Armin Biere, 'PicoSAT essentials', *JSAT*, **4**(2-4), 75–97, (2008).
[4] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle, 'ParamILS: An automatic algorithm configuration framework', *J. Artif. Intell. Res. (JAIR)*, **36**, 267–306, (2009).
[5] Chu Min Li and Wen Qi Huang, 'Diversification and determinism in local search for satisfiability', in *SAT*, pp. 158–172, (2005).
[6] Chu Min Li, Wanxia Wei, and Harry Zhang, 'Combining adaptive noise and look-ahead in local search for SAT', in *SAT*, pp. 121–133, (2007).
[7] David A. McAllester, Bart Selman, and Henry A. Kautz, 'Evidence for invariants in local search', in *AAAI*, pp. 321–326, (1997).
[8] Duc Nghia Pham, John Thornton, Charles Gretton, and Abdul Sattar, 'Combining adaptive and dynamic local search for satisfiability', *JSAT*, **4**(2-4), 149–172, (2008).
[9] Steven Prestwich, 'SAT problems with chains of dependent variables', *Discrete Applied Mathematics*, **3037**, 1–22, (2002).
[10] Steven Prestwich, 'Random walk continuously smoothed variable weights', in *SAT*, pp. 203–215, (2005).
[11] Bart Selman, Hector J. Levesque, and David G. Mitchell, 'A new method for solving hard satisfiability problems', in *AAAI*, pp. 440–446, (1992).
[12] Éric D. Taillard, 'Robust taboo search for the quadratic assignment problem', *Parallel Computing*, **17**(4-5), 443–455, (1991).
[13] John R. Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr., 'Additive versus multiplicative clause weighting for SAT', in *AAAI*, pp. 191–196, (2004).
[14] Dave A. D. Tompkins, Adrian Balint, and Holger H. Hoos, 'Captain Jack: New variable selection heuristics in local search for SAT', in *SAT*, pp. 302–316, (2011).
[15] Wanxia Wei and Chu Min Li, 'Switching between two adaptive noise mechanisms in local search', in *Booklet of the 2009 SAT Competition*, (2009).
[16] Wanxia Wei, Chu Min Li, and Harry Zhang, 'Switching among non-weighting, clause weighting, and variable weighting in local search for SAT', in *CP*, pp. 313–326, (2008).
[17] Wei Wei and Bart Selman, 'Accelerating random walks', in *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP-02)*, pp. 216–232, (2002).