Trap Avoidance in Local Search Using Pseudo-Conflict Learning

Duc Nghia Pham and **Thach-Thao Duong** and **Abdul Sattar**
Queensland Research Laboratory, NICTA Ltd., Australia and
Institute for Integrated and Intelligent Systems, Griffith University, Australia
{*duc-nghia.pham,thao.duong,abdul.sattar*}@*nicta.com.au*

Abstract

A key challenge in developing efficient local search solvers is to effectively minimise search stagnation (i.e. avoiding traps or local minima). A majority of the state-of-the-art local search solvers perform random and/or Novelty-based walks to overcome search stagnation. Although such strategies are effective in diversifying a search from its current local minimum, they do not actively prevent the search from visiting previously encountered local minima. In this paper, we propose a new preventative strategy to effectively minimise search stagnation using pseudo-conflict learning. We define a pseudo-conflict as a derived path from the search trajectory that leads to a local minimum. We then introduce a new variable selection scheme that penalises variables causing those pseudo-conflicts. Our experimental results show that the new preventative approach significantly improves the performance of local search solvers on a wide range of structured and random benchmarks.

Introduction

The satisfiability (SAT) problem is one of the best known and most well-studied problems in computer science, with many practical applications in domains such as theorem proving, hardware verification and planning. Generally, there are two main directions in designing algorithms to solve SAT problems: (i) systematic search based on the Davis-Putnam-Logemann-Loveland algorithm (Davis, Logemann, and Loveland 1962) and (ii) stochastic local search (SLS) evolving out of the GSAT algorithm (Selman, Levesque, and Mitchell 1992). As for SLS algorithms, there have been two successful but distinct avenues of development: the WalkSAT family of algorithms (McAllester, Selman, and Kautz 1997) and the various dynamic weighting approaches (e.g. (Morris 1993)). Recently, a new trend in developing efficient SLS algorithms has emerged and successfully combined the strengths of the earlier two approaches. Examples of such hybridisation algorithms include G^2WSAT (Li and Huang 2005), gNovelty$^+$ (Pham et al. 2008), TNM (Wei and Li 2009), and Sparrow (Balint and Fröhlich 2010). Machine learning techniques have also been used to automatically configure new SLS solvers based on heuristics from existing algorithms with encouraging results (e.g. SATenstein (KhudaBukhsh et al. 2009)).

Despite these significant improvements, SLS solvers are still unable to match the performance of systematic solvers on solving practical and structured SAT problems, as evident throughout previous SAT competitions.[1] One explanation for this mismatch is the inability of SLS solvers to effectively exploit domain specific properties (e.g. variable dependencies) in those SAT instances while systematic solvers can do using powerful unit propagation (UP) and conflict-driven clause-learning (CDCL) techniques. Due to the nature of SLS that greedily moves between complete assignments of variables, it is challenging to integrate tree-based techniques (such as UP and CDCL) within an SLS solver. There were attempts to integrate UP into SLS (e.g. UnitWalk (Hirsch and Kojevnikov 2005) and QingTing (Li, Stallmann, and Brglez 2003)), leading to improvements on solving structured SAT instances. However, such approach weakens the performance of SLS solvers on random instances and prevents researchers from utilising advanced SLS heuristics such as Novelty-based moves or weighting techniques.

In this paper, we aim to improve the performance of SLS on structured SAT instances by revisiting a fundamental challenge in the design of SLS algorithms: how to effectively minimise search stagnation (i.e. the number of times a local search gets stuck in local minima or traps). We speculate that as structured problems are more tightly constrained than random problems, it is easier for SLS solvers to be trapped in local minima and is harder for them to avoid or escape from those local minima. This observation is supported by our experiments on a wide range of SLS solvers over the simple ternary chain problem. A review of the state-of-the-art SLS SAT solvers reveals that the majority of them rely on random and/or Novelty-based walks to overcome search stagnation. Although such a strategy has proven to be effective in diversifying a search from its current local minimum, it does not actively prevent a search from visiting previously encountered local minima.

Consequently, we propose a new preventative approach to effectively minimise search stagnation using pseudo-conflict learning. We define a pseudo-conflict as a derived path from the search trajectory that leads to a local minimum. For each

[1] http://www.satcompetition.org

variable, we maintain the number of times it involved in a pseudo-conflict. We then use those frequencies to prioritise variables for selecting the next move and hence prevent the search from revisiting those local minima. To the best of our knowledge, this is a new approach that utilises variable weights for learning pseudo-conflicts although variable weights have been used before. In (Prestwich 2005), the flip time of a variable was used as its weight to prefer the search to the least recently flipped variable. TNM (Wei and Li 2009) exploited this variable weight concept to switch between two noise heuristics to escape from local minima. Alternatively, BGWalkSAT (Zhang, Rangan, and Looks 2003) calculated the frequency a literal appears in a near (optimal) solution (i.e. local minima) during a number of short runs. These pseudo-backbone frequencies were then used as literal weights to guide the search in the solving phase. Mazure, Sais, and Grégoire (1998) used a slightly different version of literal weights (i.e. the frequency a literal appears in a falsified clause) computed during an SLS phase to boost the performance of its main systematic solver. Cha and Iwama (1996) added resolvent clauses between pairs of falsified clauses in a local minimum to escape from it. This technique was later improved by (Fang and Ruml 2004; Shen and Zhang 2005). Audemard et al. (2010) ran a CDCL solver to assign values to all variables in a falsified clause when its main SLS solver reached a local minimum. It then resumed the SLS phase of which the partial assignment (provided by the CDCL solver) was fixed in order to prevent local search from revisiting that trap.

In the remainder of the paper, we firstly review existing heuristics to avoid and/or escape from local minima. We then provide a detailed description of our pseudo-conflict learning heuristic followed by an experimental evaluation on a wide range of structured and random benchmarks. Our results show that the new learning strategy significantly improves the performance of SLS solvers (in orders of magnitude) on those benchmarks. Finally, we present our conclusions and outline some directions for future research.

Existing Heuristics for Trap Avoidance in SLS

One of the key challenges in developing SLS solvers is to handle traps or local minima. Generally, an SLS solver starts with a random assignment of truth values to all variables in the given CNF formula. It then iteratively attempts to modify the current assignment until it finds an assignment satisfying all the CNF clauses or it is terminated. At each iteration, the search replaces the current assignment with one of its neighbours by flipping the value of one selected variable.[2] The aim here is to greedily move to a neighbour that minimises a predefined scoring function (e.g. the number of unsatisfied clauses). Given its nature, an SLS solver tends to get trapped into a local minimum where moving to any surrounding assignment cannot yield a better score than the current one. As a result, the performance of an SLS solver heavily depends on how effectively it can escape and avoid local minima.

[2]Although there is no restriction on how many variables can be flipped at each iteration, the majority of SLS SAT solvers elect to flip only one variable at each iteration.

A well-known strategy to escape from local minima is to perform a *random walk* whenever a local minimum is encountered. It is widely used by the WalkSAT family of algorithms (McAllester, Selman, and Kautz 1997). With a *walk* probability wp, WalkSAT/G and WalkSAT/B both perform a random walk move by flipping a random variable from a random clause. In addition, WalkSAT/SKC, if possible, always favours a *freebie* move over a random walk move. A freebie move is a move that does not falsify any satisfied clauses. Later, Hoos (2002) proved that occasionally performing random walks helps to reduce search stagnation.

With the introduction of G^2WSAT, Li and Huang (2005) replaced random walks with *Novelty walks* (i.e. using the Novelty algorithm to select a move) to effectively escape from local minima. Novelty (McAllester, Selman, and Kautz 1997), one of the most competitive WalkSAT variants, greedily selects the best scoring variable v from a random unsatisfied clause c, breaking ties by selecting the least recently flipped variable. If v is the most recently flipped variable, then the second best variable from clause c will be selected with a *noise* probability p. Since then, many state-of-the-art SLS solvers (such as gNovelty$^+$ (Pham et al. 2008) and TNM (Wei and Li 2009)) have employed this strategy but opted to use AdaptNovelty (Hoos 2002), an adaptive version of Novelty.

Although these strategies are effective in diversifying the search from its traps, they do not actively prevent the search from visiting previous traps. A simple strategy to avoid previous local minima is the *Tabu* heuristic (Taillard 1991). It keeps a list of k most recently flipped variables. By forbidding the search from selecting variables in this tabu list, it exhausts all the neighbour assignments and consequently escapes from the local minimum. Arguably, if the *tabu tenure* (i.e. the length of this tabu list) is long enough, tabu heuristic can also help the search avoid previous traps. However, the performance of a tabu-based algorithm significantly depends on setting the right tabu tenure, which varies vastly for different instances (even from the same problem).

A more effective approach to deal with local minima is to use weighting strategy. *Clause weighting* techniques (as used in SAPS (Hutter, Tompkins, and Hoos 2002), PAWS (Thornton et al. 2004) and gNovelty$^+$ (Pham et al. 2008)) associate a weight to each CNF clause. They then use a *weighted* scoring function (e.g. the sum of weights of all unsatisfied clauses) to guide the search. Whenever a local minimum is encountered, the weights of all currently unsatisfied clauses are increased. This alters the search landscape by gradually filling the gap and hence escape from that local minimum. As those new clause weights are also used to select subsequent moves, it prevents the search from getting stuck in previous known traps. In addition, it is widely believed and has been empirically shown that periodically reducing clause weights dramatically improves the performance of clause weighting algorithms as it forces the search to forget long-term memory of encountered traps (Hutter, Tompkins, and Hoos 2002; Thornton et al. 2004).

Alternatively, Prestwich (2005) used *variable weights* (i.e. the number of times a variable has been flipped) to avoid traps. His VW algorithm selects a move similarly to Walk-

SAT/SKC but breaks ties by selecting one with the smallest weight and breaks further ties randomly. The more advanced VW2 variant also performs continuous smoothing of variable weights similarly to clause weighting algorithms.

Recently, Balint and Fröhlich (2010) introduced a new trap escaping heuristic that combined the advantages of the clause weighted scoring scheme and a Novelty walk. The idea was to calculate a *probability distribution* of variables from a random unsatisfied clause. Their new algorithm, Sparrow, was based on gNovelty$^+$ and replaced the AdaptNovelty component in gNovelty$^+$ with the new probability distribution based variable selection mechanism. Sparrow2011 (Balint et al. 2011), the latest version of Sparrow, won two Gold medals in the 2011 SAT competition.

Experiments on the Ternary Chain Problem

In order to evaluate the effectiveness of existing strategies for trap avoidance, we conducted an experiment using seven different SLS solvers on a set of *ternary chain* instances. Table 1 lists those solvers together with the strategies they employ to escape and/or avoid traps. It is worth mentioned that TNM uses a variant of VW algorithm to switch between two variants of the Novelty strategy when encountering traps.

Solvers	random walk	Novelty walk	Tabu	variable weight	clause weight	prob. dist.
adaptG^{2}WSAT0 (Li, Wei, and Zhang 2007)	✓	✓				
TNM (Wei and Li 2009)	✓	✓ (two variants)		✓ (indirectly)		
RoTS (Taillard 1991)			✓			
PAWS (Thornton et al. 2004)	✓				✓	
VW2 (Prestwich 2005)	✓			✓		
gNovelty$^+$ (v1.2) (Pham et al. 2008)	✓	✓ (weighted)	✓		✓	
Sparrow2011 (Balint et al. 2011)			✓		✓	✓

Table 1: Trap avoidance strategies used by tested solvers.

Prestwich (2002) defined the ternary chain problem as:

$$(v_1) \wedge (v_2) \wedge (v_1 \wedge v_2 \rightarrow v_3) \wedge \ldots \wedge (v_{n-2} \wedge v_{n-1} \rightarrow v_n)$$

We selected this problem as it closely simulates the chains of variable dependencies, regularly found in highly-structured problems from real-world domains such as hardware/software verification and planning. The problem contains only a single solution (where all variables are true) and can be easily solved using UP. However, it poses many traps for SLS solvers and remains a challenge for them to solve. In fact, it was proved to be exponentially hard for random walk algorithms (Wei and Selman 2002) and is highly recommended for evaluating the performance of SLS solvers (Prestwich 2002; Wei and Selman 2002; Prestwich 2005).

Figure 1 plots the performance of tested solvers on small 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 experiment was conducted on a SunFire v20z cluster, which runs Fedora Core OS and is equipped with dual-core 2.6GHz AMD Opteron processors

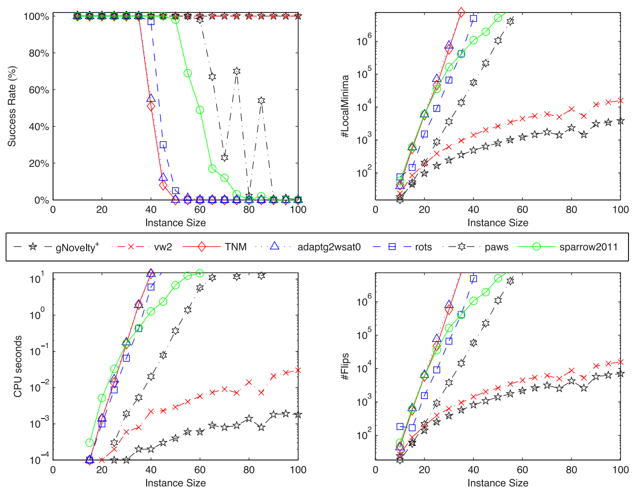

Figure 1: The performance of seven benchmarked solvers on small ternary chain instances. Each data point (except those in the success rate plot) is the median result over 100 runs.

and 4GB of RAM for each processor. As shown in Figure 1, adaptG^{2}WSAT0 and TNM (which mainly use Novelty walk strategy) are the worst performers, following by RoTS (which uses tabu strategy). None of them could solve instances with size greater than 50. Our experiment also confirms what Prestwich (2005) observed in his study that weighting techniques are more effective than other strategies in avoiding traps. Indeed, VW2 and gNovelty$^+$ successfully solved all tested instances with 100% success rate whilst Sparrow2011 (the worst performer among the four weighting solvers) managed to solve instances with size up to 75. Finally, our experiment shows that gNovelty$^+$ is the best performer among the seven benchmarked solvers.

Pseudo-Conflict Learning

Although VW2 came second in our previous experiment, its performance curves, especially the number of flips and the number of local minima encountered, closely resemble those of gNovelty$^+$ (as shown in Figure 1). Pairwise, both algorithms perform random walks occasionally. However, VW2 implements a lighter and simpler weighting heuristic than gNovelty$^+$. As previously described, VW2 always selects the least flipped variable to diversify the search to freshly new or poorly explored areas. In fact, Wei, Li, and Zhang (2008) pointed out that VW algorithms are more diversified than adaptG^{2}WSAT (Li, Wei, and Zhang 2007), which arguably is a non-weighted variant of gNovelty$^+$. This is confirmed by the results reported in Figure 1. In addition, our experiment also shows that VW2 performs much better than PAWS, which is arguably a variant of gNovelty$^+$ without Tabu (with the tenure = 1) and weighted Novelty walks. Therefore, it raises the question of whether we can combine the advantages of clause and variable weighting strategies to further minimise search stagnation.

The probability distribution based heuristic in Sparrow (Balint and Fröhlich 2010) can be viewed as a form of combining clause weighting and variable weighting to escape from local minima. Its probability distribution function

$$p(v_i) = \frac{p_s(v_i) * p_a(v_i)}{\sum_{v_i} p_s(v_i) * p_a(v_i)}$$

gives priority to variables v_i with higher clause weighted score $wscore(v_i)$ (as in $p_s(v_i) = {c_1}^{wscore(v_i)}$). It also uses $age(v_i)$, the last iteration that v_i was flipped, to assist the variable selection. If v_i was recently flipped, its probability will be discounted by $p_a(v_i) = (\frac{age(v_i)}{c_3})^{c_2}+1$. However, the probability of v_i being selected will be greatly enhanced if it was flipped long ago (i.e. its age $\geq c_3$). Note that c_1, c_2, and c_3 are parameters. Unfortunately, Sparrow2011 is the worst performer among the four benchmarked weighting solvers.

In this work, we want to capture the cause of a conflict in the current local minimum and then use it to prevent the local search from revisiting that trap. Currently, systematic solvers that use the CDCL mechanism are very effective in avoiding previous traps. This CDCL mechanism actively monitors the current assignment for conflicts. Whenever a conflict is detected, it will perform resolution to find the cause of that conflict. Such a cause is recorded and then used to prevent the search from falling into that trap again. Unfortunately, we cannot directly implement a CDCL scheme within an SLS solver due to its nature of operation.

Therefore in order to approximately record the cause of a local minimum, we introduce the *pseudo-conflict* concept. Whenever a local search hits a local minimum, we hypothesise that the assignment of the k most recently flipped variables is actually the cause of this conflict. Obviously, there is no trivial method to work out the exact value of k rather than guessing. Hence, we call such segment of the search trajectory that contains the last k flipped variables a pseudo-conflict. In our implementation, we opted to set a small value for k as we observed that small pseudo-conflicts were practically the best approximation of conflict causes.

Ideally, one would record all derived pseudo-conflicts to prevent the search from visiting a previously encountered trap. However, this may require a lot of memory to store them. In addition, the overhead of ensuring that a potential candidate variable, if flipped, will not lead to a known trap can make a significant negative impact on the solver's performance. Thus, we decided to adopt the variable weight concept to record pseudo-conflicts and guide the search.

Algorithm 1 outlines our pseudo-conflict learning (PCL) mechanism. We associate with each variable v_i a *frequency* weight to record the number of times that v_i is involved in a pseudo-conflict. Whenever a local minimum is encountered, the weights of the k most recently flipped variables are increased by 1 (line 1). In a subsequent variable selection, they are used to break ties, preferring the variable with the lowest weight. Like other weighting techniques, we also reduce those weights periodically to counter the long-term memory effect. Every time a time window T elapses, we reduce the weights of all weighted variables by half (lines 2-5).

Note that our PCL mechanism is significantly different to the Tabu heuristic. A Tabu solver memorises only the most recent pseudo-conflict (its size is equal to the Tabu tenure). Hence, if one sets the Tabu tenure large enough to cover a reasonable number of known traps, it is more likely that many good moves are also tabooed. It is also bad if the tenure is too small as the search only has a short-term memory of one or two most recent traps. In contrast, our PCL mechanism can memorise a reasonable amount of known traps without severely ruling out good candidate variables.

Algorithm 1: Pseudo-conflict learning

```
Input: the current iteration $it$, the pseudo-conflict tenure $k$, a time window $T$
1 for $i = 0$; $i < k$; $i$++ do frequency(search_history[it-i].variable)++;
2 if a time window $T$ has elapsed then
3     foreach variable $v_i$ do
4         if frequency($v_i$) $> 1$ then reduce frequency($v_i$) by half;
5         else frequency($v_i$) $\leftarrow 0$;
```

Integrating PCL within SLS Solvers

Here, we describe how our PCL mechanism is integrated into gNovelty$^+$ and Sparrow2011. The two new algorithms, gNovelty$^+$PCL and Sparrow2011-PCL, are outlined in Algorithms 2 and 3 respectively. The differences between PCL solvers and their original ones are also highlighted.

Algorithm 2: gNovelty$^+$PCL(Θ)

```
Input: A CNF formula $\Theta$
Output: A solution $\alpha$ (if found) or TIMEOUT
1  randomly generate an assignment $\alpha$;
2  while not timeout do
3      if $\alpha$ satisfies $\Theta$ then return $\alpha$ as the solution;
4      if within a walking probability $wp$ then
5          randomly select a variable $v$ that appears in an unsatisfied clause;
6      else
7          if there exist promising variables then
8              select the most promising variable $v$, breaking ties by selecting
               the least pseudo-conflicted variable;
9          else
10             perform pseudo-conflict learning;
11             update (and smooth within a probability $sp$) clause weights;
12             select the most improving variable $v$ from a random unsatisfied
               clause, breaking ties by selecting the least pseudo-conflicted
               variable;
13     update $\alpha$ with the flipped value of $v$;
14     update the search trajectory;
15 return TIMEOUT;
```

gNovelty$^+$PCL We firstly removed the Tabu heuristic out of gNovelty$^+$PCL as its benefit is now cover by our PCL mechanism. We then modified its gradient-based variable selection. When there are two or more promising variables with the same highest score, we will select the *least pseudo-conflicted* one (i.e. a variable with the lowest *frequency* weight) instead of selecting the least recently flipped variable (as gNovelty$^+$ does). The reason for this change is to favour a move that less likely leads to a known trap. We will also break ties when performing the weighted AdaptNovelty walk by favouring the least pseudo-conflicted variable.

Algorithm 3: Sparrow2011-PCL(Θ)

```
Input: A CNF formula $\Theta$
Output: A solution $\alpha$ (if found) or TIMEOUT
1  randomly generate an assignment $\alpha$;
2  while not timeout do
3      if $\alpha$ satisfies $\Theta$ then return $\alpha$ as the solution;
4      if there exist promising variables then
5          select the most promising variable $v$, breaking ties by selecting the
           least pseudo-conflicted variable;
6      else
7          perform pseudo-conflict learning;
8          update (and smooth within a probability $sp$) clause weights;
9          select the highest probability variable $v$ from a random unsatisfied
           clause using the new probability distribution;
10     update $\alpha$ with the flipped value of $v$;
11     update the search trajectory;
12 return TIMEOUT;
```

Sparrow2011-PCL We removed the Tabu heuristic from Sparrow2011-PCL and modified its gradient-based variable selection to favour the least pseudo-conflicted variable when there is a tie. We also replaced the $p_a(v_i)$ component in the probability function $p(v_i)$ with the new component $p_f(v_i) = (\frac{c5}{frequency(v_i)+1})^{c_4} + 1$. This favours variables involved in a small number of pseudo-conflicts ($\leq c_5$) and heavily penalises those highly leading to traps.

Experimental Results and Analysis

We firstly compared the performance of gNovelty$^+$PCL and Sparrow2011-PCL against other solvers on ternary chain instances. We further included large instances with size ranging from 100 to 1000 in steps of 50. All settings are kept the same as in our previous experiment. The new results are summarised in Figures 2 and 3. Each data point (except those in the success rate plots) is the median result over 100 runs. As shown in Figure 2, our PCL mechanism greatly enhances the performance of gNovelty$^+$ and Sparrow2011 up to an order of magnitude on three measures: the number of local minima encountered, the number of CPU seconds and the number of flips. Unfortunately, Sparrow2011-PCL was unable to solve all small instances with 100% success rate. However, gNovelty$^+$PCL clearly outperformed other solvers on larger instances (as shown in Figure 3).

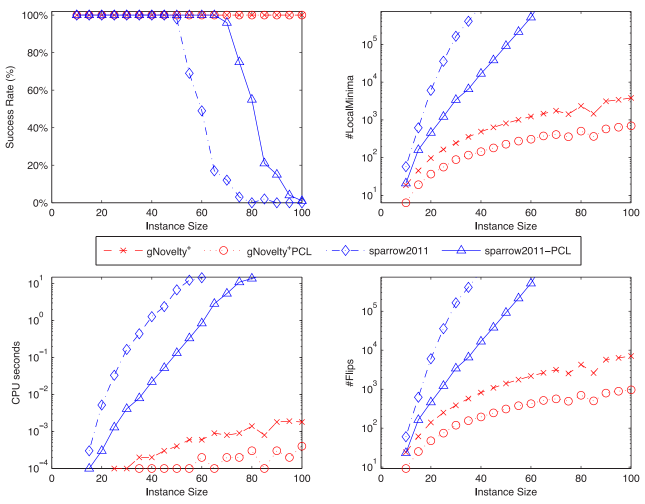

Figure 2: Small ternary chains.

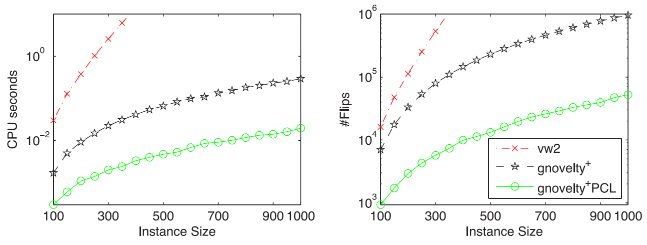

Figure 3: Large ternary chains: no omitted solver could solve any instance.

We then ran a new experiment on three real-world verification problem suites: (i) 100 sss-sat-1.0 superscalar microprocessors verification instances,[3] (ii) 39 cbmc bounded model checking instances, and (iii) 75 swv software verification instances generated by the CALYSTO checker.[4]

The experimental results are summarised in Table 2 and depicted in Figure 4. The parameter settings for PCL solvers are reported in Table 4. All solvers were run 50 times for each instance and each run was timed out after 600 seconds. The experiment was conducted on a cluster equipped with hexa-core CPUs (Intel Xeon X5650 @2.67GHz, 12MB L2 Cache) and 2GB RAM per core, running Red Hat Santiago OS. For each benchmark set in Table 2, we report (i) the percentage success rate; and (ii) the mean-of-minimum, the mean-of-mean and the mean-of-maximum solution-times in seconds. The percentage success rate of a benchmark is calculated as the percentage of solved instances in that benchmark set. An instance is considered solved if its success rate

[3]http://www.miroslav-velev.com/sat_benchmarks.html

[4]The cbmc and swv instances are *test* instances from http://people.cs.ubc.ca/~davet/papers/sat10-dave-instances.zip

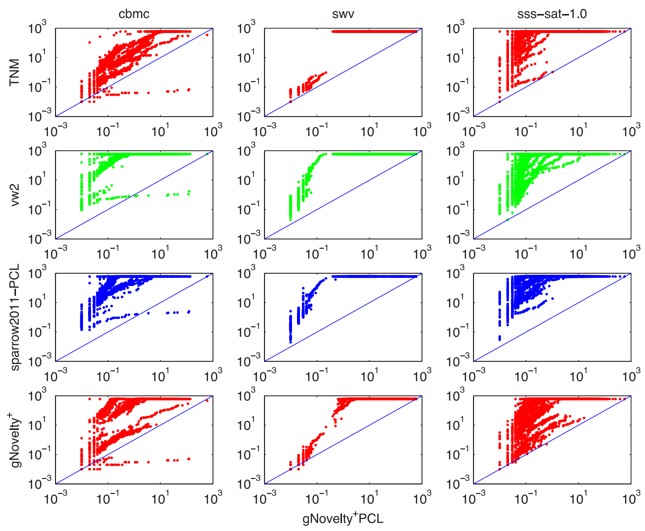

Figure 4: gNovelty$^+$PCL vs. others: CPU time performance

Instances	TNM		VW2		Sparrow2011		Sparrow2011-PCL		gNovelty$^+$		gNovelty$^+$PCL		minisat2.2		glucose2	
	%sr	#secs	%sr	#secs	%sr	#secs	%sr	#secs	%sr	#secs	%sr	#secs	%sr	#secs	%sr	#secs
cbmc (39)	92%	13.399 76.599 164.602	31%	362.911 439.128 498.058	51%	119.692 384.359 499.474	59%	158.482 311.671 461.666	85%	54.573 247.997 382.512	**100%**	**0.044** **2.378** **28.643**	100%	0.009	100%	0.011
swv (75)	23%	464.005 464.013 464.037	23%	464.133 466.002 474.762	21%	413.379 486.949 519.651	23%	461.103 465.217 472.802	25%	409.451 459.097 464.131	**48%**	**304.774** **326.605** **357.854**	100%	0.138	100%	0.143
sss-sat-1.0 (100)	18%	350.904 517.709 546.742	24%	318.728 481.357 523.310	7%	484.792 572.993 588.703	22%	324.935 498.615 531.340	50%	162.757 348.512 427.622	**100%**	**0.154** **2.370** **16.667**	100%	0.211	100%	0.216

Table 2: The performance of difference solvers on verification instances.

Instances	Sparrow2011		Sparrow2001-PCL		gNovelty$^+$		gNovelty$^+$PCL		minisat2.2		glucose2	
	%sr	#secs	%sr	#secs	%sr	#secs	%sr	#secs	%sr	#secs	%sr	#secs
application (8)	25%	461.824 515.799 543.695	25%	461.207 477.668 542.149	12%	464.149 527.431 527.503	**50%**	**229.016** **373.847** **479.386**	100%	14.560	100%	4.055
crafted (82)	56%	237.720 285.159 338.434	59%	217.799 270.204 307.305	62%	202.872 251.332 285.784	**84%**	**67.631** **114.586** **156.552**	51.21%	336.279	43.90%	362.970
large random (64)	20%	437.202 **509.487** **555.076**	19%	443.579 524.496 568.140	0%	576.600 n/a 600.000	**25%**	**313.537** 515.067 581.770	0%	n/a	0%	n/a
medium random (201)	**100%**	5.049 40.671 99.540	99%	**3.161** 31.429 87.231	75%	130.371 184.698 241.121	**100%**	3.851 **13.487** **46.103**	13.15%	512.980	17.46%	541.407

Table 3: The performance of difference solvers on SAT-2011 benchmark instances.

is at least 50%. The overall means are taken over the minimums, means and maximums time to solve each instance. Again, statistics on solution-time are only on successful runs and when success rate is at least 50%. Furthermore, we also included in Table 2 the results of two CDCL solvers: minisat2.2 and glucose2.

Problems	gNovelty$^+$PCL			Sparrow2011-PCL			
	k	T	sp	k	T	c_4	c_5
cbmc	25	100	35	25	300	1	16.0
smv	10	300	5	25	250	12	16.5
sss-sat-1.0	20	100	5	25	300	3	8.0
sat-2011	15	250	0	30	300	1	1.5

Table 4: Parameter settings for PCL solvers. The values were tuned mainly based on the impact on the average flips and not because of the overhead caused by Algorithm 1.

These results re-confirmed the benefits of our PCL mechanism on gNovelty$^+$ and Sparrow2011. gNovelty$^+$PCL solved the cbmc and sss-sat problems with 100% success. That is 100% improvement on sss-sat-1.0 as gNovelty$^+$ solved only 50% of this set. In addition, the CPU time performance of gNovelty$^+$PCL is up to many orders of magnitude better than its original. Although the improvement for Sparrow2011-PCL is not that great, it managed to solve 22% of sss-sat instances while its counterpart Sparrow2011 achieved only 7%. Overall, gNovelty$^+$PCL is the best performer and significantly outperformed other benchmarked solvers on structured verification problems (as shown in Table 2 and Figure 4).

Motivated by these results on verification problems, we conducted another experiment on all the benchmark instances used in the 2011 SAT competition to further evaluate the performance of PCL solvers against their original solvers on the traditional benchmark problems for SLS solvers. Each solver was run 10 times on each instance with a 600s limit per run. The experiment was conducted on the same hardware used in the previous verification test. The parameter settings for PCL solvers on SAT 2011 benchmarks are reported in Table 4. The new results are summarised in Table 3. Here we only report results on instances that can be solved by at least one SLS solver. Note that all statistics in Table 3 are computed in the similar manner to those in Table 2. Once again, our new results, as shown in Table 3, demonstrate the significant benefits of integrating PCL mechanism with gNovelty$^+$ on both random and structured instances.

Finally, it is worthy to note that constraint propagation has been successfully used in structure-based SLS solvers (Pham, Thornton, and Sattar 2007; Belov, Järvisalo, and Stachniak 2011). However, there is little report on how they perform on non-structured (especially random) instances. The structure-based solver in (Pham, Thornton, and Sattar 2007) performed significantly worse on bart instances from which it could not derive any structure information (see Table 3 in (Pham, Thornton, and Sattar 2007)). Based on the results reported in (Belov, Järvisalo, and Stachniak 2011), we found an interesting comparison: gNovelty$^+$PCL solved all of the 100 sss-sat-1 .0 instances (with a maximum CPU time per run is 16.667s) whilst d-crsat (the best in its structured-based SLS class) solved only 79 of these instances (with a 300s time limit per run). However, we do not suggest that gNovelty$^+$PCL will perform better than d-crsat on other structured problems. Of course, d-crsat will solve the ternary chain instances with ease.

Conclusion and Future Work

In conclusion, we experimentally showed that weighting strategies are much better than non-weighting techniques in minimising search stagnation. We also found that the simpler and lighter variable weighting scheme used in VW2 is not that worse in comparison with the more complicated weighting strategy employed in gNovelty$^+$. Based on this observation and further analysis, we introduced a new PCL mechanism to approximately learn the causes of conflicts for local search solvers. This idea was inspired by the CDCL scheme as well as the Tabu and variable weighting heuristics. We then showed how to integrate this new mechanism with gNovelty$^+$ and Sparrow2011. Finally we experimentally showed that gNovelty$^+$PCL performed significantly better than existing SLS solvers on both artificially generated and real-world benchmark instances. In future, we plan to further fine-tune the PCL mechanism as well as make these parameters to self-tune.

Acknowledgments

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

Audemard, G.; Lagniez, J.-M.; Mazure, B.; and Sais, L. 2010. Boosting local search thanks to CDCL. In *Proceedings of LPAR-10*, 474–488.

Balint, A., and Fröhlich, A. 2010. Improving stochastic local search for SAT with a new probability distribution. In *Proceedings of SAT-10*, 10–15.

Balint, A.; Fröhlich, A.; Tompkins, D. A.; and Hoos, H. H. 2011. Sparrow2011. In *Booklet of SAT-2011 Competition*.

Belov, A.; Järvisalo, M.; and Stachniak, Z. 2011. Depth-driven circuit-level stochastic local search for SAT. In *Proceedings of IJCAI-11*, 504–509.

Cha, B., and Iwama, K. 1996. Adding new clauses for faster local search. In *Proceedings of AAAI-96*, 332–337.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A machine program for theorem proving. *Communications of the ACM* 5(7):394–397.

Fang, H., and Ruml, W. 2004. Complete local search for propositional satisfiability. In *Proceedings of AAAI-04*, 161–166.

Hirsch, E. A., and Kojevnikov, A. 2005. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Math. and AI* 43(1):91–111.

Hoos, H. H. 2002. An adaptive noise mechanism for Walk-SAT. In *Proceedings of AAAI-02*, 635–660.

Hutter, F.; Tompkins, D. A.; and Hoos, H. H. 2002. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proceedings of CP-02*, 233–248.

KhudaBukhsh, A. R.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2009. SATenstein: Automatically building local search SAT solvers from components. In *Proceedings of IJCAI-93*, 517–524.

Li, C. M., and Huang, W. Q. 2005. Diversification and determinism in local search for satisfiability. In *Proceedings of SAT-05*, 158–172.

Li, X. Y.; Stallmann, M. F. M.; and Brglez, F. 2003. QingTing: A fast SAT solver using local search and efficient unit propagation. In *Proceedings of SAT-03*, 452–467.

Li, C. M.; Wei, W.; and Zhang, H. 2007. Combining adaptive noise and look-ahead in local search for SAT. In *Proceedings of SAT-07*, 121–133.

Mazure, B.; Sais, L.; and Grégoire, É. 1998. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence* 22(3-4):319–331.

McAllester, D. A.; Selman, B.; and Kautz, H. A. 1997. Evidence for invariants in local search. In *Proceedings of AAAI-97*, 321–326.

Morris, P. 1993. The Breakout method for escaping from local minima. In *Proceedings of AAAI-93*, 40–45.

Pham, D. N.; Thornton, J.; Gretton, C.; and Sattar, A. 2008. Combining adaptive and dynamic local search for satisfiability. *JSAT* 4(2-4):149–172.

Pham, D. N.; Thornton, J.; and Sattar, A. 2007. Building structure into local search for SAT. In *Proceedings of IJCAI-07*, 2359–2364.

Prestwich, S. 2002. SAT problems with chains of dependent variables. *Discrete Applied Mathematics* 3037:1–22.

Prestwich, S. 2005. Random walk continuously smoothed variable weights. In *Proceedings of SAT-05*, 203–215.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings of AAAI-92*, 440–446.

Shen, H., and Zhang, H. 2005. Another complete local search method for SAT. In *Proceedings of LPAR-05*, 595–605.

Taillard, E. D. 1991. Robust taboo search for the quadratic assignment problem. *Parallel Computing* 17(4-5):443–455.

Thornton, J. R.; Pham, D. N.; Bain, S.; and Ferreira Jr., V. 2004. Additive versus multiplicative clause weighting for SAT. In *Proceedings of AAAI-04*, 191–196.

Wei, W., and Li, C. M. 2009. Switching between two adaptive noise mechanisms in localsearch. In *Booklet of SAT-2009 Competition*.

Wei, W., and Selman, B. 2002. Accelerating random walks. In *Proceedings of CP-02*, 216–232.

Wei, W.; Li, C. M.; and Zhang, H. 2008. A switching criterion for intensification and diversification in local search for SAT. *JSAT* 4(2-4):219–237.

Zhang, W.; Rangan, A.; and Looks, M. 2003. Backbone guided local search for maximum satisfiability. In *Proceedings of IJCAI-93*, 1179–1186.