# Combining Adaptive and Dynamic Local Search for Satisfiability

**Duc Nghia Pham**                                    duc-nghia.pham@nicta.com.au
*SAFE Program,*
*Queensland Research Laboratory,*
*NICTA Ltd., Australia*

**John Thornton**                                        john.thornton@nicta.com.au
*Institute for Integrated and Intelligent Systems,*
*Griffith University, Australia*

**Charles Gretton**                                  charles.gretton@nicta.com.au
*SAFE Program,*
*Queensland Research Laboratory,*
*NICTA Ltd., Australia*

**Abdul Sattar**                                        abdul.sattar@nicta.com.au
*Institute for Integrated and Intelligent Systems,*
*Griffith University, Australia*

## Abstract

In this paper we describe a stochastic local search (SLS) procedure for finding models of satisfiable propositional formulae. This new algorithm, gNovelty$^+$, draws on the features of two other WalkSAT family algorithms: AdaptNovelty$^+$ and G$^2$WSAT, while also successfully employing a hybrid clause weighting heuristic based on the features of two dynamic local search (DLS) algorithms: PAWS and (R)SAPS.

gNovelty$^+$ was a Gold Medal winner in the random category of the 2007 SAT competition. In this paper we present a detailed description of the algorithm and extend the SAT competition results via an empirical study of the effects of problem structure, parameter tuning and resolution preprocessors on the performance of gNovelty$^+$. The study compares gNovelty$^+$ with three of the most representative WalkSAT-based solvers: AdaptG$^2$WSAT0, G$^2$WSAT and AdaptNovelty$^+$, and two of the most representative DLS solvers: RSAPS and PAWS. Our new results augment the SAT competition results and show that gNovelty$^+$ is also highly competitive in the domain of solving *structured* satisfiability problems in comparison with other SLS techniques.

KEYWORDS: *SAT-solver, local search, clause weighting, adaptive heuristic*

*Submitted October 2007; revised January 2008; published May 2008*

## 1. Introduction

The satisfiability (SAT) problem is one of the best known and well-studied problems in computer science, with many practical applications in domains such as theorem proving, hardware verification and planning. The techniques used to solve SAT problems can be divided into two main areas: complete search techniques based on the well-known Davis-Putnam-

Logemann-Loveland (DPLL) algorithm [3] and stochastic local search (SLS) techniques evolving out of Selman and Kautz's 1992 GSAT algorithm [18]. As for SLS techniques, there have been two successful but distinct avenues of development: the WalkSAT family of algorithms [14] and the various dynamic local search (DLS) clause weighting approaches (e.g. [15]).

Since the early 1990s, the state-of-the-art in SAT solving has moved forward from only being able to solve problems containing hundreds of variables to the routine solution of problems with millions of variables. One of the key reasons for this success has been the keen competition between researchers and the public availability of the source code of the best techniques. Nowadays the SAT community organises regular competitions on large sets of benchmark problems and awards prizes to the best performing algorithms in different problem categories. In this paper we introduce the current 2007 SAT competition[1.] Gold Medal winner in the satisfiable random problem category: gNovelty$^+$.

gNovelty$^+$ evolved from a careful analysis of the SLS solvers that participated in the 2005 SAT competition and was initially designed only to compete on random SAT problems. It draws on the strengths of two WalkSAT variants which respectively came first and second in the random category of the 2005 SAT competition: R+AdaptNovelty$^+$ [1] and G$^2$WSAT [12]. In addition, gNovelty$^+$ connects the two branches of SLS (WalkSAT and DLS) by effectively exploiting a hybrid clause weighting heuristic based on ideas taken from the two main approaches to clause weighting DLS algorithms: additive weighting (e.g. PAWS [21]) and multiplicative weighting (e.g. (R)SAPS [11]).

In the remainder of the paper we describe in more detail the techniques used in G$^2$WSAT, R+AdaptNovelty$^+$, PAWS and (R)SAPS before discussing the strengths and weaknesses of these solvers based on the results from the 2005 SAT competition and our own study. We then provide a full explanation of the execution of gNovelty$^+$ followed by an experimental evaluation on a range of random and structured problems. As the performance of gNovelty$^+$ on random problems is now a matter of public record,[2.] this evaluation examines the performance of gNovelty$^+$ on a broad benchmark set of structured problems, testing the effects of parameter tuning and resolution preprocessing in comparison with a range of state-of-the-art SLS solvers. Finally, we present our conclusions and outline some directions for future research.

## 2. Preliminaries

In this section, we briefly describe and summarise the key techniques used in four SLS solvers that represent the state-of-the-art in the two main streams of SLS development: the WalkSAT family and clause weighting DLS solvers.

### 2.1 AdaptNovelty$^+$

During the mid-1990s, Novelty [14] was considered to be one of the most competitive techniques in the WalkSAT family. Starting from a random truth assignment to the problem variables, Novelty repeatedly changes single variable assignments (i.e. it makes a *flip* move)

---

1. http://www.satcompetition.org

2. More detailed results from the competition are available at http://www.satcompetition.org/

until a solution is found. The cost of flipping a variable (i.e. flipping the assignment of that variable) is defined as the number of unsatisfied clauses after $x$ is flipped. In more detail, at each search step Novelty greedily selects the best variable $x$ from a random unsatisfied clause $c$ such that flipping $x$ leads to the minimal number of unsatisfied clauses. If there is more than one variable with the same flip cost, the least recently flipped variable will be selected. In addition, if $x$ is the most recently flipped variable, then the second best variable from clause $c$ will be selected with a fixed *noise* probability $p$. This flip selection procedure is outlined in lines 10-13 of Algorithm 1.

Although Novelty generally achieves better results than other WalkSAT variants introduced during its time [14], due to its deterministic variable selection[3.] it may loop indefinitely and fail to return a solution even where one exists [7, 12]. We refer the reader to [7] for an example instance that is satisfiable but for which Novelty is unable to find a solution regardless of the noise parameter setting. Hoos [7] solved this problem by adding a random walk behaviour (lines 7-9 in Algorithm 1) to the Novelty procedure. The resulting Novelty$^+$ algorithm randomly flips a variable from a randomised unsatisfied clause $c$ with a *walk* probability $wp$ and behaves exactly as Novelty otherwise.

---

**Algorithm 1** AdaptNovelty$^+$(F, $wp$=0.01)

---

1: randomly generate an assignment A;
2: **while** not timeout **do**
3:     **if** A satisfies F **then**
4:         **return** A as the solution;
5:     **else**
6:         randomly select an unsatisfied clause $c$;
7:         **if** within a walking probability $wp$ **then**
8:            *randomly* select a variable $x$ in $c$;
9:         **else**
10:           *greedily* select the best variable $x$ in $c$, breaking ties by selecting the least recently flipped promising variable;
11:           **if** $x$ is the most recently flipped variable in $c$ AND within a noise probability $p$ **then**
12:             re-select $x$ as the second best variable;
13:           **end if**
14:         **end if**
15:         update A with the flipped value of $x$;
16:         adaptively adjust the noise probability $p$;
17:     **end if**
18: **end while**
19: **return** 'no solution found';

---

It was shown that the performance of every WalkSAT variant (including Novelty and Novelty$^+$) critically depends on the setting of the noise parameter $p$ which, in turn, controls the level of greediness of the search [8, 14]. This means that without extensive empirical tuning, the average case performance of a WalkSAT algorithm is quite poor. Hoos [8] addressed this problem by proposing an adaptive version of WalkSAT that dynamically adjusts the noise value based on the automatic detection of search stagnation. This AdaptNovelty$^+$ version of Novelty$^+$ (outlined in Algorithm 1) starts with $p = 0$ (i.e. the solver is completely greedy in selecting the next move). If the search enters a stagnation stage (i.e. it encounters

---

3. Novelty only selects the next move from the two best variables of a randomly selected unsatisfied clause.

a local minimum where none of the considered moves yields fewer unsatisfied clauses than the current assignment), then the noise value is gradually increased to allow the selection of non-greedy moves that will allow the search to overcome its stagnation. Once the local minimum is escaped, the noise value is reduced to again make the search more greedy. Hoos [8] demonstrated experimentally that this adaptive noise mechanism is effective both with Novelty$^+$ and the other WalkSAT variants.

## 2.2 G$^2$WSAT

More recently Li and Huang [12] proposed a new heuristic to solve the problem of determinism in Novelty (discussed in the previous section). Rather than using a Novelty$^+$-type random walk [7], they opted for a solution based on the timestamping of variables to make the selection process more diversified. The resulting Novelty$^{++}$ heuristic (lines 9-14 in Algorithm 2) selects the least recently flipped variable from a randomly selected clause $c$ for the next move with a *diversification* probability $dp$, otherwise it performs as Novelty. Li and Huang [12] further improved Novelty$^{++}$ by combining the greedy heuristic in GSAT [18] with a variant of tabu search [6] as follows: during the search, all variables that, if flipped, do not strictly minimise the objective function are considered tabu (i.e. they cannot be selected for flipping during the greedy phase). Once a variable $x$ is flipped, only those variables that become *promising* as a consequence of flipping $x$ (i.e. that will strictly improve the objective function if flipped) will lose their tabu status and become available for greedy variable selection. The resulting G$^2$WSAT solver (outlined in Algorithm 2) always selects the most promising non-tabu variable for the next move, if such variable is available. If there is more than one variable with the best score, G$^2$WSAT selects the least recently flipped one, and if the search hits a local minimum, G$^2$WSAT disregards the tabu list and performs as Novelty$^{++}$ until it escapes.

---

**Algorithm 2** G$^2$WSAT(F, $dp$, $p$)

---

1: randomly generate an assignment A;
2: **while** not timeout **do**
3:   **if** A satisfies F **then**
4:     **return** A as the solution;
5:   **else**
6:     **if** there exist *promising* variables **then**
7:       *greedily* select the most *non-tabu* promising variable $x$, breaking ties by selecting the least recently flipped promising variable;
8:     **else**
9:       randomly select an unsatisfied clause $c$;
10:      **if** within a diversification probability $dp$ **then**
11:        select the least recently flipped variable $x$ in $c$;
12:      **else**
13:        select a variable $x$ in $c$ according to the Novelty heuristic;
14:      **end if**
15:     **end if**
16:     update A with the flipped value of $x$;
17:     update the tabu list;
18:   **end if**
19: **end while**
20: **return** 'no solution found';

---

## 2.3 (R)SAPS

As opposed to the previously discussed SLS algorithms (that use a count of unsatisfied clauses as the search objective function) DLS algorithms associate weights with clauses of a given formula and use the sum of weights of unsatisfied clauses as the objective function for the selection of the next move. Typically, clause weights are initialised to 1 and are dynamically adjusted during the search to help in avoiding or escaping local minima. Depending on how clause weights are updated, DLS solvers can be divided into two main categories: *multiplicative* weighting and *additive* weighting. Algorithm 3 sketches out the basics of the Scaling and Probabilistic Smoothing (SAPS) algorithm [11], which is arguably the current best DLS solver in the multiplicative category. At each search step, SAPS greedily attempts to flip the most promising variable that strictly improves the weighted objective function. If no promising variable exists, SAPS randomly selects a variable for the next move with walk probability $wp$. Otherwise, with probability $(1 - wp)$, SAPS multiplies the weights of all unsatisfied clauses by a factor $\alpha > 1$ and consequently directs future search to traverse an assignment that will satisfy currently unsatisfied clauses. After updating weights, with smooth probability $sp$ clause weights are probabilistically smoothed and reduced to the average clause weight by a factor $\rho$. This smoothing phase helps the search forget the earlier weighting decisions, as these past effects are generally no longer helpful to escape future local minima.

---

**Algorithm 3** (R)SAPS(F, $wp$=0.01, $sp$, $\alpha$=1.3, $\rho$=0.8)

---

1:  initialise the weight of each clause to 1;
2:  randomly generate an assignment A;
3:  **while** not timeout **do**
4:      **if** A satisfies F **then**
5:          **return** A as the solution;
6:      **else**
7:          **if** there exist *promising* variables **then**
8:              *greedily* select a promising variable $x$ that occurs in an unsatisfied clauses, breaking ties by randomly selecting;
9:          **else if** within a walk probability $wp$ **then**
10:             randomly select a variable $x$;
11:         **end if**
12:         **if** $x$ has been selected **then**
13:             update A with the flipped value of $x$;
14:         **else**
15:             scale the weights of unsatisfied clauses by a factor $\alpha$;
16:             with probability $sp$ smooth the weights of all clauses by a factor $\rho$;
17:         **end if**
18:         **if** in *reactive* mode **then**
19:             adaptively adjust the smooth probability $sp$;
20:         **end if**
21:     **end if**
22: **end while**
23: **return** 'no solution found';

---

SAPS has four parameters and its performance critically depends on finding the right settings for these parameters. Hutter, Tompkins & Hoos [11] attempted to dynamically adjust the value of the smooth probability $sp$ using the same approach as AdaptNovelty$^+$

[7], while holding the other three parameters ($wp$, $\alpha$ and $\rho$) fixed. Their experimental study showed that the new RSAPS solver can achieve similar and sometimes better results in comparison to SAPS [11]. However, the other parameters in RSAPS, especially $\rho$, still need to be manually tuned in order to achieve optimal performance [9, 20].

## 2.4 PAWS

Recently, Thornton *et al.* [21] were the first to closely investigate the performance difference between additive and multiplicative weighting DLS solvers. Part of this study included the development of the Pure Additive Weighting Scheme (PAWS), which is now one of the best DLS algorithms in the additive weighting category. The basics of PAWS are outlined in Algorithm 4. Instead of performing a random walk when no promising variable exists as SAPS does, PAWS randomly selects and flips a *flat-move* variable with a fixed flat-move probability $fp = 0.15$.[4] Otherwise, with probability $(1 - fp)$, the weights of all unsatisfied clauses are increased by 1. After a fixed number $w_{inc}$ of weight increases, PAWS deterministically reduces the weights of all weighted clauses by 1. The experimental results conducted in [21, 20] demonstrated the overall superiority of PAWS over SAPS for solving large and difficult problems.

---

**Algorithm 4** PAWS($\mathsf{F}$, $fp$=0.15, $w_{inc}$)

---

1: initialise the weight of each clause to 1;
2: randomly generate an assignment $\mathsf{A}$;
3: **while** not timeout **do**
4:     **if** $\mathsf{A}$ satisfies $\mathsf{F}$ **then**
5:         **return** $\mathsf{A}$ as the solution;
6:     **else**
7:         **if** there exist *promising* variables **then**
8:             *greedily* select a promising variable $x$, breaking ties by randomly selecting;
9:         **else if** there exist *flat-move* variables AND within a flat-move probability $fp$ **then**
10:            randomly select a flat-move variable $x$;
11:        **end if**
12:        **if** $x$ has been selected **then**
13:            update $\mathsf{A}$ with the flipped value of $x$;
14:        **else**
15:            increase the weights of unsatisfied clauses by 1;
16:            **if** has updated weights for $w_{inc}$ times **then**
17:                reduce the weights of all weighted clauses by 1;
18:            **end if**
19:        **end if**
20:    **end if**
21: **end while**
22: **return** 'no solution found';

---

---

4. A flat-move variable is one that, if flipped, will cause no change to the objective function.

## 3. gNovelty$^+$: An 'Overall' Solver for Random Problems

### 3.1 Observations from the 2005 SAT Competition

The initial development of gNovelty$^+$ focussed on preparing for the 2007 SAT competition. This meant concentrating on the random problem category, where SLS solvers have traditionally outperformed complete solvers. Consequently we paid considerable attention to the best performing techniques from this category in the 2005 SAT competition: R+AdaptNovelty$^+$, G$^2$WSAT and R+PAWS.[5.] Table 1 summarises the performance of these solvers on random SAT instances in the first phase of the 2005 SAT competition. Note that R+AdaptNovelty$^+$ and R+PAWS are variants of AdaptNovelty$^+$ and PAWS, respectively, where resolution is used to preprocess the input problem before the main solver is called.

**Table 1.** The number of random instances solved by R+AdaptNovelty$^+$, G$^2$WSAT and R+PAWS in the first phase of the 2005 SAT competition.

| Solvers | Large Size Problems | | | Medium Size Problems | | |
|---|---|---|---|---|---|---|
| | 3-SAT | 5-SAT | 7-SAT | 3-SAT | 5-SAT | 7-SAT |
| R+AdaptNovelty$^+$ | 22 | 32 | 19 | 35 | 35 | 35 |
| G$^2$WSAT | 37 | 2 | 14 | 35 | 35 | 35 |
| R+PAWS | 33 | 1 | 12 | 35 | 35 | 35 |

From Table 1, it is clear that R+AdaptNovelty$^+$ was able to win the 2005 competition because of its superior performance on the large 5-SAT and 7-SAT instances. As the resolution preprocessor employed by R+AdaptNovelty$^+$ (and also R+PAWS) only operates on clauses of length $\leq 3$ in the input and only adds resolvent clauses of length $\leq 3$ to the problem, this competition winning performance must be credited to the AdaptNovelty$^+$ heuristic rather than to the effects of resolution.

The large 3-SAT instance results in Table 1 clearly show that R+AdaptNovelty$^+$ was outperformed by G$^2$WSAT and R+PAWS. As AdaptNovelty$^+$ limits its variable selection to a *single* randomly selected unsatisfied clause while G$^2$WSAT and PAWS pick the most promising variable from *all* unsatisfied clauses, we conjectured that the superior performance of G$^2$WSAT and R+PAWS on 3-SAT was due to this more aggressive greediness.

However, when considering the SAT competition results we should bear in mind that each solver was only run once on each instance. This means that the random effects of different starting positions could have distorted the underlying average performance of each algorithm. In order to verify our observations, we therefore conducted our own experiments in which each solver was run 100 times per instance to minimise any starting position effects. We used the original AdaptNovelty$^+$ and PAWS algorithms[6.] to eliminate any advantage these solvers may have obtained from the resolution preprocessor on the 3-SAT instances. Figure 1 plots the head-to-head comparisons of these solvers on 12 3-SAT instances, 12 5-SAT instances and 10 7-SAT instances randomly selected from the large benchmark set used in

---

5. R+PAWS was ranked third in the first phase of the 2005 SAT competition. However, due to the competition rule that authors can have only one solver competing in the final phase, R+PAWS was withdrawn from the final phase of the competition as it was submitted by the same authors of R+AdaptNovelty$^+$.

6. The $w_{inc}$ parameter of PAWS in this experiment was set to 10 as it was in the competition.
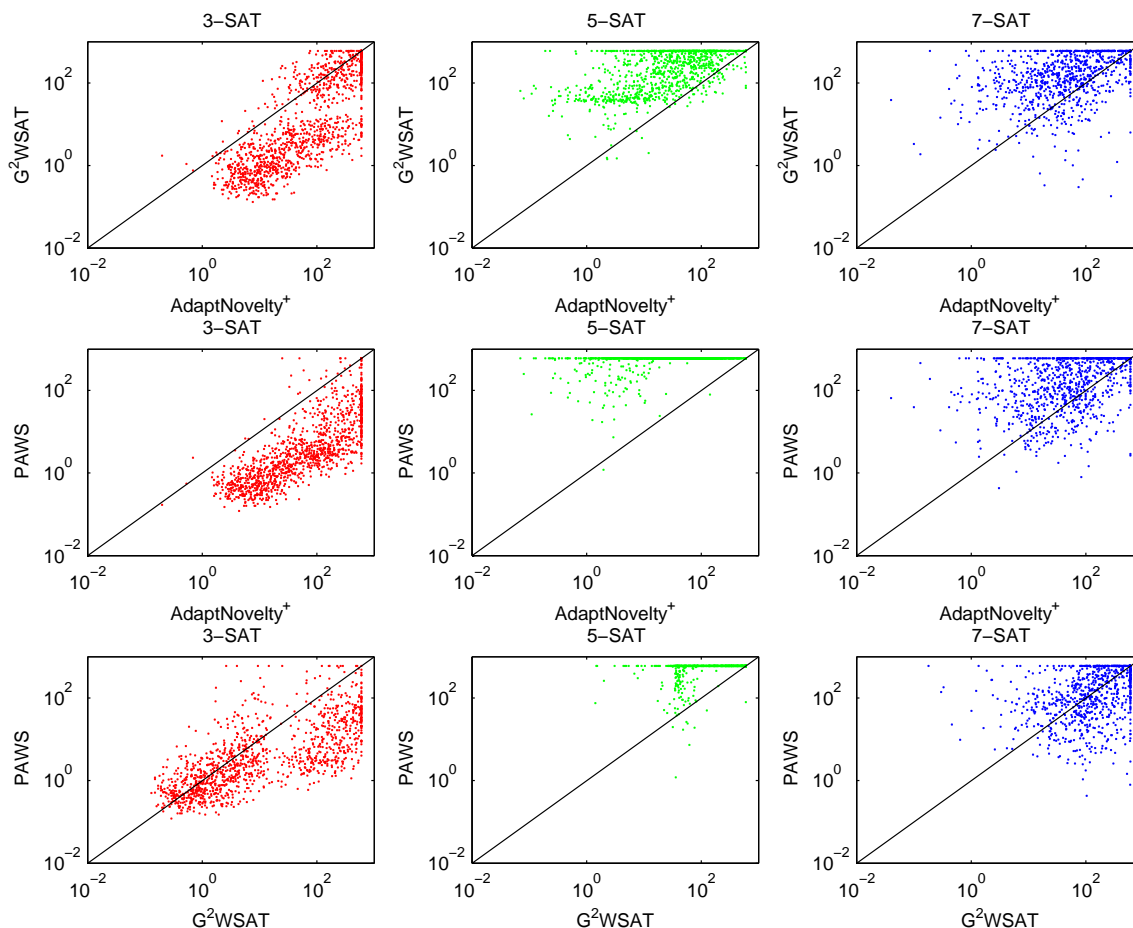
**Figure 1.** Head-up comparison between AdaptNovelty$^+$, G$^2$WSAT and PAWS on selected random 3-SAT, 5-SAT and 7-SAT instances from the 2005 SAT competition.

the 2005 competition. All experiments (including those presented in subsequent sections) were performed on cluster of 16 computers, each with a single AMD Opteron 252 2.6GHz processor with 2GB of RAM, and each run was timed out at 600 seconds. More detailed results are reported in Table 2.

These results confirm our conjectures that a more greedy heuristic (e.g. G$^2$WSAT or PAWS) performs better on random 3-SAT instances while a less greedy approach such as AdaptNovelty$^+$ is better on random 5-SAT and 7-SAT instances. The results also show that PAWS without resolution preprocessing outperforms G$^2$WSAT on 3-SAT instances. This result is consistent with the findings in [1] where resolution preprocessing was shown to harm the performance of local search solvers on random problems. The outstanding performance of PAWS further suggests that clause weighting provides useful guidance for random 3-SAT instances.

## 3.2 The Design of gNovelty$^+$

---

**Algorithm 5** gNovelty$^+$(F, $wp$=0.01, $sp$, $p$)

---

1: initialise the weight of each clause to 1;
2: randomly generate an assignment A;
3: **while** not timeout **do**
4:     **if** A satisfies F **then**
5:         **return**  A as the solution;
6:     **else**
7:         **if** within a walking probability $wp$ **then**
8:           randomly select a variable $x$ that appears in an unsatisfied clause;
9:         **else if** there exist *promising* variables **then**
10:           *greedily* select a *non-tabu* promising variable $x$, breaking ties by selecting the least recently flipped promising variable;
11:         **else**
12:           *greedily* select the most promising variable $x$ from a random unsatisfied clause $c$, breaking ties by selecting the least recently flipped promising variable;
13:           **if** $x$ is the most recently flipped variable in $c$ AND within a noise probability $p$ **then**
14:             re-select $x$ as the second most promising variable;
15:           **end if**
16:           update the weights of unsatisfied clauses;
17:           with probability $sp$ smooth the weights of all *weighted* clauses;
18:         **end if**
19:         update A with the flipped value of $x$;
20:         update the tabu list;
21:         adaptively adjust the noise probability $p$;
22:     **end if**
23: **end while**
24: **return**  'no solution found';

---

On the basis of the preceding observations, we developed a new overall solver for random problems, called gNovelty$^+$. We based this solver on G$^2$WSAT as it provides a good framework for combining the strengths of the three solvers. We first replaced the Novelty$^{++}$ heuristic in G$^2$WSAT with the AdaptNovelty$^+$ heuristic to enhance performance on the 5-SAT and 7-SAT instances. We then moved the random walk step inherited from AdaptNovelty$^+$ to the top of the solver to provide a better balance between diversification and greediness. Finally, we integrated the additive clause weighting scheme from PAWS into gNovelty$^+$. We selected the additive scheme as it is computationally cheaper and provides better guidance than its multiplicative counterpart. As shown in Table 2, RSAPS (which implements multiplicative weighting) performs significantly worse on random instances. However, we replaced the deterministic weight smoothing phase from PAWS with a linear version of the probabilistic weight smoothing phase from SAPS. This gave us more flexibility in controlling the greediness of gNovelty$^+$ which proved to be useful in our experimental study.

The basics of gNovelty$^+$ are sketched out in Algorithm 5. It starts with a full random assignment of values to all variables of the input problem and initialises all clause weights to one. At each search step, gNovelty$^+$ performs a random walk with a walk probability $wp$

fixed to 0.01.[7.] With probability $(1 - wp)$, gNovelty$^+$ selects the most promising *non-tabu* variable that is also the least recently flipped, based on a *weighted* objective function that aims to minimise the sum of weights of all unsatisfied clauses. If no such promising variable exists, the next variable is selected using a heuristic based on AdaptNovelty that again uses the *weighted* objective function. After an AdaptNovelty step, gNovelty$^+$ increases the weights of all currently unsatisfied clauses by 1. At the same time, with a *smoothing* probability $sp$, gNovelty$^+$ will reduce the weight of all *weighted* clauses by 1.[8.] It is also worth noting that gNovelty$^+$ initialises and updates its tabu list of promising variables in the same manner as G$^2$WSAT with the following exception: all variables that become promising during the weight updating phase are removed from the tabu list. In addition, gNovelty$^+$ only uses the tabu list when doing greedy variable selection and disregards the list when it performs a random walk or an AdaptNovelty step.

We manually tuned the parameter $sp$ of gNovelty$^+$ on the small random 3-SAT, 5-SAT and 7-SAT instances by varying its value from 0 to 1 in steps of 0.1. It should be noted that setting $sp = 0$ will stop gNovelty$^+$ from performing its probabilistic weight smoothing phase, while setting $sp = 1$ will effectively turn off all gNovelty$^+$'s clause weighting phases. It turned out that $sp = 0.4$ is the best setting for gNovelty$^+$ on the random 3-SAT instances, while $sp = 1$ was the best setting for the random 5-SAT and 7-SAT instances. We therefore ran gNovelty$^+$ with these two $sp$ settings on the 34 random problems reported in Figure 1 to evaluate its performance against its three predecessors. The detailed performance of these two versions are reported in Table 2. The previously reported results of AdaptNovelty$^+$, G$^2$WSAT and PAWS are also included for comparison purposes. To give an idea of the relative performance of a multiplicative weighting algorithm, we included the results for RSAPS in Table 2 as well.

Overall these random problem results show that the performance of gNovelty$^+$ closely reflects the relative performance of the predecessor algorithms on which it is based. Firstly, on the 3-SAT instances where PAWS dominates AdaptNovelty$^+$ and G$^2$WSAT, it is also the case that gNovelty$^+$ with weight ($sp = 0.4$) dominates its counterpart gNovelty$^+$ without weight ($sp = 1.0$). Conversely, on the 5-SAT and 7-SAT results, where AdaptNovelty$^+$ strongly dominates G$^2$WSAT and PAWS, the gNovelty$^+$ version *without* weight ($sp = 1.0$) performs significantly better than gNovelty$^+$ with weight ($sp = 0.4$).

In addition, if we compare the best version of gNovelty$^+$ against the best version of its predecessors (i.e. gNovelty$^+$($sp = 0.4$) versus PAWS on the 3-SAT instances and gNovelty$^+$($sp = 1.0$) versus AdaptNovelty$^+$ on the 5-SAT and 7-SAT instances), the results show that gNovelty$^+$ is at least as good and often better than its counterparts when the problems become bigger and harder. More specifically, gNovelty$^+$ dominates all other solvers on the bigger 5-SAT problems (k5-v600 and k5-v800 instances) and is dominant interchangeably with PAWS on the larger 3-SAT k3-v6000 and k3-v8000 instances and AdaptNovelty$^+$ on the 7-SAT k7-v140 and k7-v160 instances. The runtime distributions (RTDs) in Figure 2 further confirm that gNovelty$^+$ has achieved our goal of becoming the best overall solver across the three random problem categories.

---

7. Hoos [7] empirically showed that setting $wp$ to 0.01 is enough to make an SLS solver become "probabilistically approximately complete".

8. A clause is weighted if its weight is greater than 1.

**Table 2.** Results on random k-SAT instances shown in the form: $\frac{median}{mean}$. Best results are marked with ★ and flip counts are reported in thousands. On problems where a solver was timed out for some runs, we report the percentage of success of that solver instead of its CPU time and flip count.

| Instances | G²WSAT | | AdaptNovelty+ | | PAWS | | RSAPS | | gNovelty+ (sp=0.4) | | gNovelty+ (sp=1.0) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs |
| k3-v4000-1672 | 450/769 | 0.500/0.796 | 8,027/9,097 | 5.590/6.388 | 296★/466★ | 0.430★/0.571★ | 0% success | | 693/1,012 | 0.865/1.247 | 18,498/25,357 | 25.700/36.303 |
| k3-v4000-1674 | 982/1,641 | 1.035/1.655 | 13,769/15,525 | 9.780/10.988 | 458★/705★ | 0.615★/0.853★ | 0% success | | 1,316/1,816 | 1.560/2.161 | 36,988/56,273 | 52.190/79.284 |
| k3-v4000-1680 | 560/714 | 0.550/0.689 | 8,301/11,314 | 5.855/7.902 | 326★/493★ | 0.455★/0.586★ | 0% success | | 859/1,152 | 1.025/1.362 | 18,835/31,849 | 26.105/44.762 |
| k3-v4000-1681 | 1,246/3,253 | 1.240/2.911 | 20,085/24,202 | 14.170/17.131 | 667★/1,203★ | 0.980★/1.452★ | 0% success | | 1,953/2,388 | 2.445/2.937 | 94% success | |
| k3-v6000-1682 | 1,813/5,088 | 2.350/5.842 | 34,912/35,309 | 27.850/28.277 | 755★/1,247★ | 1.300★/1.792★ | 0% success | | 3,059/3,661 | 4.720/5.513 | 88% success | |
| k3-v6000-1683 | 90% success | | 59% success | | 96% success | | 0% success | | 29,138★/36,750★ | 41.770★/51.945★ | 2% success | |
| k3-v6000-1684 | 4,675/31,960 | 5.875/34.043 | 81,385/94,014 | 65.530/75.126 | 1,550★/3,773★ | 2.740★/4.860★ | 0% success | | 5,997/8,966 | 8.380/12.423 | 48% success | |
| k3-v6000-1686 | 44,944/92,358 | 46.090/92.432 | 98% success | | 99% success | | 0% success | | 13,221★/19,550★ | 19.545★/28.457★ | 7% success | |
| k3-v8000-1693 | 5,813/29,910 | 8.195/36.031 | 111,619/126,067 | 102.240/115.953 | 1,645★/5,025★ | 3.255★/7.306★ | 0% success | | 10,029/14,937 | 17.915/26.749 | 18% success | |
| k3-v8000-1694 | 62% success | | 24% success | | 93% success | | 0% success | | 62,294★/77,401★ | 104.160★/126.752★ | 0% success | |
| k3-v8000-1698 | 64,578/92,255 | 72.910/101.316 | 99% success | | 2,391★/10,251★ | 4.930★/14.429★ | 0% success | | 17,239/26,729 | 30.220/45.820 | 3% success | |
| k3-v8000-1701 | 83% success | | 54% success | | 98% success | | 0% success | | 47,290★/58,711★ | 77.040★/95.475★ | 1% success | |
| k5-v500-1533 | 26,926/34,975 | 88.675/114.784 | 8,598/10,069 | 16.295/19.023★ | 6% success | | 0% success | | 2% success | | 3,317★/4,949★ | 13.115★/19.543 |
| k5-v500-1537 | 11,899/12,922 | 40.920/44.209 | 1,608/2,883 | 3.005★/5.406★ | 35% success | | 1% success | | 17% success | | 1,133★/1,506★ | 4.295/5.757 |
| k5-v500-1540 | 87% success | | 19,040/26,131 | 36.180★/49.562★ | 0% success | | 0% success | | 2% success | | 9,673★/13,817★ | 38.455/54.948 |
| k5-v500-1541 | 11,285/12,064 | 38.920/41.344 | 1,326/1,883 | 2.540/3.598★ | 62% success | | 1% success | | 27% success | | 652★/995★ | 2.520★/3.868 |
| k5-v600-1542 | 59% success | | 25,069/41,388 | 49.295/81.377 | 1% success | | 0% success | | 0% success | | 9,813★/13,311★ | 41.375★/56.013★ |
| k5-v600-1544 | 45% success | | 31,190/43,091 | 61.120/84.402 | 0% success | | 0% success | | 0% success | | 13,561★/16,206★ | 56.655★/67.709★ |
| k5-v600-1547 | 42% success | | 24,427/43,789 | 47.560★/85.215 | 0% success | | 0% success | | 0% success | | 13,894★/20,203★ | 58.295/84.856★ |
| k5-v600-1550 | 77% success | | 22,280/32,758 | 43.540/64.161 | 0% success | | 0% success | | 0% success | | 8,919★/11,992★ | 37.255★/50.094★ |
| k5-v700-1552 | 42% success | | 97% success | | 0% success | | 0% success | | 0% success | | 12,073★/17,648★ | 53.555★/77.710★ |
| k5-v700-1557 | 39% success | | 36,796/51,272 | 73.570/102.555 | 0% success | | 0% success | | 0% success | | 9,350★/11,692★ | 40.925★/51.131★ |
| k5-v700-1558 | 55% success | | 23,394/34,032 | 46.675/68.051 | 0% success | | 0% success | | 0% success | | 7,207★/10,130★ | 31.095★/43.839★ |
| k5-v700-1561 | 5% success | | 59% success | | 0% success | | 0% success | | 0% success | | 79% success★ | |
| k7-v120-1583 | 99% success | | 3,989/6,407 | 20.440★/32.861 | 7,643/13,564 | 49.415/87.850 | 66% success | | 99% success | | 3,347★/4,560★ | 21.935★/29.881★ |
| k7-v120-1584 | 97% success | | 11,652/15,935 | 59.920/81.898 | 94% success | | 37% success | | 90% success | | 6,100★/10,387★ | 39.620★/67.414★ |
| k7-v120-1587 | 4,013/6,204 | 28.340/43.656 | 1,935/2,709 | 9.925/13.932 | 5,851/7,779 | 36.055/47.738 | 81% success | | 7,515/12,752 | 47.995/81.470 | 1,391★/2,065★ | 8.940★/13.231★ |
| k7-v120-1591 | 11,546/13,707 | 80.915/95.852 | 6,632/8,914 | 33.835/45.474 | 7,184/12,256 | 44.460/75.892 | 61% success | | 98% success | | 4,830★/7,085★ | 28.580★/41.964★ |
| k7-v140-1592 | 97% success | | 4,631★/8,545★ | 24.495★/45.177★ | 88% success | | 28% success | | 71% success | | 6,992/9,334 | 48.195/64.418 |
| k7-v140-1597 | 58% success | | 95% success | | 48% success | | 8% success | | 37% success | | 96% success★ | |
| k7-v140-1599 | 85% success | | 12,905★/18,652★ | 69.960★/101.019★ | 71% success | | 19% success | | 50% success | | 99% success | |
| k7-v140-1601 | 59% success | | 88% success★ | | 55% success | | 12% success | | 32% success | | 81% success | |
| k7-v160-1604 | 27% success | | 56% success | | 18% success | | 2% success | | 5% success | | 60% success★ | |
| k7-v160-1606 | 7% success | | 37% success★ | | 12% success | | 3% success | | 3% success | | 22% success | |

    Given the above results, we entered gNovelty+ into the 2007 SAT competition and set it to automatically adjust the value of its parameter *sp* depending on the input problem size. If gNovelty+ detects that the input formula is a random 3-SAT instance, it will run with a smooth probability of $sp = 0.4$. Otherwise, it will reset *sp* back to 1.0. On this basis, gNovelty+ was able to win the Gold Medal for the Random SAT category of the
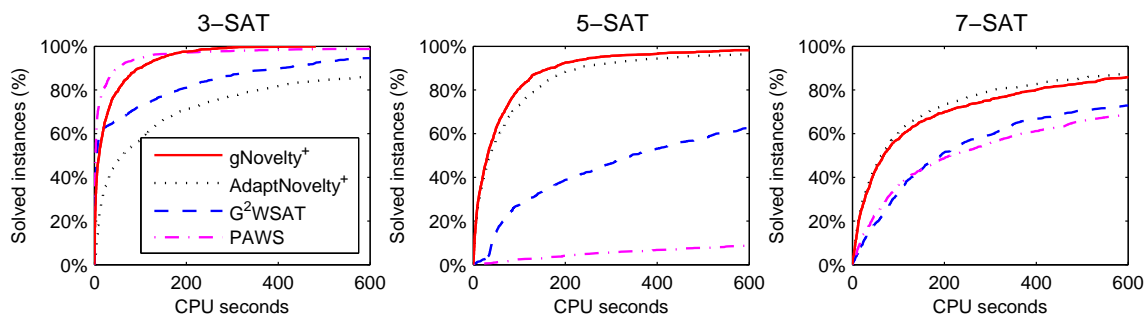
**Figure 2.** Runtime distribution of 4 solvers on random instances. The smooth probability of gNovelty$^+$ is set to 0.4 for 3-SAT instances and 1.0 for 5-SAT and 7-SAT instances.

competition.[9]

In the remainder of the paper, we focus on answering the question whether gNovelty$^+$ (an algorithm designed specifically for random problems) has a wider field of useful application. To answer this, we devised an extensive experimental study to test gNovelty$^+$ in comparison with other state-of-the-art SLS SAT solvers and across a range of benchmark structured problems.

## 4. Experimental Setup and Benchmark Sets

As the performance of gNovelty$^+$ in the SAT random category is already a matter of public record,[10] we based our experimental study on a range of structured benchmark problems that have been used in previous SLS comparison studies.[11] Our problem test set comprises of four circuit synthesis formula problems (2bitadd_11, 2bitadd_12, 3bitadd_31 and 3bitadd_32), three all-interval series problems (ais10 to ais14), two blocksworld planning problems (bw_large.c and bw_large.d), four Beijing scheduling problems (enddr2-1, enddr2-8, ewddr2-1 and ewddr2-8), two "flat" graph colouring problems (flat200-med and flat200-har), four large DIMACS graph colouring problems (g125.17 to g250.29), two logistics planning problems (logistics.c and logistics.d), five 16-bit parity function learning problems (par16-1-c to par16-5-c), and five hard quasi-group problems (qg1-08 to qg7-13).

As gNovelty$^+$ combines the strengths of solvers from the WalkSAT series and DLS algorithms, for comparison purposes we selected algorithms from each of the four possible categories, i.e. manual WalkSAT (G$^2$WSAT [12]), adaptive WalkSAT (AdaptNovelty$^+$ [8]), manual clause weighting (PAWS [20]) and adaptive clause weighting (RSAPS [11]). In addition, we included AdaptG$^2$WSAT0 [13], an adaptive version of G$^2$WSAT, as it came second in the random SAT category of the 2007 SAT competition. It should be noted that these algorithms have consistently dominated other local search techniques in the recent

---

9. http://www.satcompetition.org

10. See http://www.cril.univ-artois.fr/SAT07/slides-contest07.pdf

11. See http://www.satlib.org

SAT competitions (where the majority of modern SAT solvers developed by the research community have competed). We therefore consider them to be a fair representation of the state-of-the-art. While other SAT solvers have been developed that may also have proved competitive (e.g. commercial solvers), the lack of availability of their source code has precluded their inclusion in the current work.

For this experimental study, we manually tuned the parameters of PAWS, G$^2$WSAT and gNovelty$^+$ to obtain optimal performance for each category of the problem set.[12] These settings are shown in Table 3 (note, only one parameter setting per algorithm was allowed for each of the eight problem categories). Here we not only manipulated the gNovelty$^+$ $sp$ parameter but on some categories we also manually tuned the noise parameter of its Novelty component. For G$^2$WSAT we used the optimal settings for the noise and $dp$ parameters published in [12, 13], and for PAWS we tuned the $w_{inc}$ parameter.

**Table 3.** Optimal parameter settings for each problem category.

| Method | Parameter | Problem Category | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | bitadd | ais | bw_large | e*ddr | flat200 | g | logistics | par16 | qg |
| gNovelty$^+$ | $p$ | adapted | adapted | 0.08 | adapted | adapted | 0.10 | adapted | 0.05 | 0.02 |
| | $sp$ | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.10 | 0.00 |
| G$^2$WSAT | $p$ | 0.50 | 0.20 | 0.20 | 0.40 | 0.50 | 0.30 | 0.20 | 0.50 | 0.40 |
| | $dp$ | 0.05 | 0.05 | 0.00 | 0.45 | 0.06 | 0.01 | 0.05 | 0.01 | 0.03 |
| PAWS | $w_{inc}$ | 9 | 52 | 4 | 59 | 74 | 4 | 100 | 40 | 10 |

## 5. Structured Problem Results

Table 4 shows the results obtained after manually tuning gNovelty$^+$, G$^2$WSAT and PAWS in comparison to the default adaptive behaviour of AdaptNovelty$^+$, AdaptG$^2$WSAT0 and RSAPS. Here the results for the best performing algorithm on each problem are shown in bold, with all results reporting the mean and median of 100 runs of each algorithm on each instance (each run was timed out after 600 seconds). In order to have a fair comparison, we disabled the unit propagation preprocessor used in G$^2$WSAT and AdaptG$^2$WSAT0 in the two studies presented in this section. The results of all solvers in association with different preprocessors are discussed in later sections.

A brief overview shows that gNovelty$^+$ has the best results for all bitadd, ais, bw_large, e*ddr and logistics problems. In addition, it has the best results on the three hardest quasigroup problems (RSAPS won on two other instances) and is about equal with G$^2$WSAT on the flat graph colouring problems. Of the other algorithms, PAWS is the best for the parity problems, G$^2$WSAT is the best for the two harder large graph instances while PAWS and RSAPS each won on one easier instance. On this basis gNovelty$^+$ emerges as the best algorithm both in terms of the number of problems (19) and the number of problem classes (6) in which it dominates.

An even clearer picture emerges when we look at the overall proportion of runs that completed within 600 seconds. Here, gNovelty$^+$ achieves a 99.90% success rate compared

---

12. The other three solvers (AdaptNovelty$^+$, RSAPS and AdaptG$^2$WSAT0) can automatically adapt the values of their parameters during the search.

**Table 4.** Optimally tuned results on structured problems shown in the form: $\frac{median}{mean}$. Best results are marked with ★ and flip counts are reported in thousands. On problems where a solver was timed out for some runs, we report the percentage of success of that solver instead of its CPU time and flip count.

| Instances | gNovelty+ | | AdaptG2WSAT0 | | G2WSAT | | AdaptNovelty+ | | RSAPS | | PAWS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs |
| 2bitadd_11 | 0.928/0.947 | 0.000★/0.001★ | 2.458/2.348 | 0.000/0.003 | 0.510★/0.677★ | 0.000/0.002 | 1.592/1.693 | 0.000/0.001 | 374/577 | 0.230/0.368 | 86% success | |
| 2bitadd_12 | 0.576/0.625★ | 0.000★/0.001★ | 1.716/1.882 | 0.000/0.003 | 0.394★/200 | 0.000/0.136 | 1.521/1.554 | 0.000/0.001 | 59/111 | 0.040/0.071 | 50,062/127,135 | 30.105/75.140 |
| 3bitadd_31 | 18★/20★ | 0.060★/0.068★ | 549/817 | 1.120/1.625 | 0% success | | 151/163 | 0.440/0.467 | 0% success | | 0% success | |
| 3bitadd_32 | 15★/16★ | 0.060★/0.058★ | 262/318 | 0.535/0.652 | 0% success | | 131/135 | 0.450/0.466 | 0% success | | 0% success | |
| ais10 | 8.116★/12★ | 0.010★/0.012★ | 135/200 | 0.120/0.174 | 66/88 | 0.065/0.086 | 1,291/1,937 | 1.135/1.671 | 13/18 | 0.020/0.020 | 13/20 | 0.015/0.022 |
| ais12 | 48★/64★ | 0.060★/0.080★ | 3,127/4,320 | 3.445/4.754 | 879/1,428 | 1.045/1.698 | 24,315/35,292 | 28.140/40.833 | 103/151 | 0.135/0.202 | 104/192 | 0.150/0.262 |
| ais14 | 334★/414★ | 0.435★/0.537★ | 13,455/23,628 | 15.850/27.803 | 3,008/10,800 | 3.870/13.667 | 81% success | | 629/892 | 0.885/1.265 | 1,267/1,677 | 1.825/2.455 |
| bw_large.c | 800★/1,277★ | 1.440★/2.222★ | 3,317/4,930 | 3.990/5.902 | 1,991/3,297 | 2.025/3.261 | 6,350/9,355 | 6.550/9.699 | 3,606/5,361 | 10.720/15.921 | 991/1,604 | 1.460/2.268 |
| bw_large.d | 937★/1,131★ | 2.645★/3.098★ | 10,600/15,714 | 21.550/30.726 | 2,891/4,296 | 4.700/6.763 | 21,462/30,616 | 37.055/55.237 | 70% success | | 1,029/1,371 | 2.770/3.508 |
| enddr2-1 | 39★/44★ | 0.190★/0.196★ | 2,404/2,510 | 3.980/4.110 | 223/322 | 0.915/1.187 | 5,785/7,040 | 9.075/11.187 | 61/72 | 0.505/0.534 | 47/58 | 0.500/0.519 |
| enddr2-8 | 30★/34★ | 0.170★/0.180★ | 2,088/2,164 | 3.655/3.715 | 134/158 | 0.585/0.643 | 4,287/5,173 | 6.905/8.284 | 49/54 | 0.495/0.509 | 41/44 | 0.500/0.505 |
| ewddr2-1 | 32★/34★ | 0.190★/0.192★ | 1,939/2,079 | 3.435/3.686 | 114/134 | 0.570/0.613 | 4,660/6,193 | 7.970/10.352 | 48/54 | 0.550/0.550 | 45/47 | 0.550/0.549 |
| ewddr2-8 | 30★/32★ | 0.200★/0.198★ | 1,757/1,828 | 3.365/3.454 | 90/103 | 0.535/0.564 | 4,959/6,515 | 8.690/10.959 | 48/50 | 0.600/0.602 | 43/44 | 0.580/0.584 |
| flat200-med | 164/241 | 0.070/0.099 | 182/242 | 0.065/0.087 | 133★/169★ | 0.050★/0.065★ | 260/392 | 0.085/0.131 | 287/377 | 0.150/0.196 | 248/348 | 0.135/0.185 |
| flat200-har | 2,576/4,037 | 1.040★/1.638★ | 3,418/5,476 | 1.210/1.950 | 4,050/16,903 | 1.475/5.967 | 17,879/22,572 | 6.055/7.580 | 3,143/4,200 | 1.595/2.149 | 2,403★/3,344★ | 1.280/1.761 |
| g125.17 | 687/1,066 | 3.065/4.792 | 804/1,161 | 3.175/4.546 | 528/747 | 2.230★/2.978★ | 981/1,264 | 4.110/5.408 | 2% success | | 492★/694★ | 2.310/3.415 |
| g125.18 | 13/15 | 0.090/0.098 | 52/51 | 0.170/0.169 | 7.718★/9.872★ | 0.090/0.100 | 35/36 | 0.100/0.100 | 1,057/1,787 | 4.995/8.379 | 11/13 | 0.080★/0.084★ |
| g250.15 | 2,585/2,668 | 0.110/0.112 | 2,889/2,909 | 0.210/0.208 | 2,381/2,410 | 0.545/0.545 | 3,311/4,033 | 0.110/0.118 | 2,208★/2,219★ | 0.080★/0.082★ | 2,239/2,247 | 0.090/0.086 |
| g250.29 | 638/704 | 12.725/14.956 | 716/765 | 8.235/9.147 | 247★/297★ | 5.455★/6.259★ | 755/895 | 9.960/11.979 | 0% success | | 263/320 | 8.070/9.237 |
| logistics.c | 6.332★/6.873★ | 0.010★/0.007★ | 2,550/3,631 | 1.385/1.996 | 52/65 | 0.040/0.045 | 122/152 | 0.070/0.091 | 6.811/7.814 | 0.010/0.008 | 10/12 | 0.010/0.012 |
| logistics.d | 27/32★ | 0.040★/0.042★ | 111/137 | 0.090/0.109 | 86/107 | 0.090/0.102 | 170/196 | 0.100/0.114 | 23★/33 | 0.040/0.051 | 30/42 | 0.050/0.063 |
| par16-1-c | 6,943/9,621 | 2.920/4.064 | 13,759/18,975 | 5.300/7.402 | 5,689/118,873 | 2.370/48.676 | 17,186/32,062 | 5.870/10.978 | 73% success | | 1,557★/2,470★ | 0.860★/1.368★ |
| par16-2-c | 28,291/38,826 | 11.975/16.493 | 129,544/190,040 | 51.655/75.896 | 96% success | | 159,405/260,240 | 53.415/87.516 | 39% success | | 3,675★/4,805★ | 2.020★/2.668★ |
| par16-3-c | 18,136/27,626 | 7.750/11.787 | 40,233/53,219 | 16.285/21.554 | 46,134/57,719 | 19.330/24.175 | 66,942/102,783 | 23.075/35.486 | 42% success | | 2,613★/4,106★ | 1.455★/2.313★ |
| par16-4-c | 11,146/16,938 | 4.810/7.216 | 23,984/39,982 | 9.445/15.905 | 55,693/156,597 | 23.060/64.789 | 80,419/112,780 | 27.905/38.674 | 69% success | | 1,035★/2,183★ | 0.570★/1.211★ |
| par16-5-c | 11,830/17,545 | 5.020/7.436 | 23,480/37,596 | 9.505/15.329 | 69% success | | 90,820/126,322 | 31.225/43.841 | 41% success | | 3,169★/4,092★ | 1.740★/2.239★ |
| qg1-08 | 647★/920★ | 15.645★/22.476★ | 99% success | | 30% success | | 99% success | | 59% success | | 80% success | |
| qg2-08 | 2,545★/3,295★ | 51.120★/69.991★ | 52% success | | 3% success | | 43% success | | 36% success | | 20% success | |
| qg5-11 | 99% success | | 0% success | | 0% success | | 1% success | | 2,287★/3,288★ | 24.575★/35.405★ | 22% success | |
| qg6-09 | 726/3,090 | 2.235/9.322 | 5% success | | 1% success | | 14% success | | 29★/44★ | 0.110★/0.173★ | 833/1,263 | 3.285/4.995 |
| qg7-13 | 98% success★ | | 0% success | | 0% success | | 0% success | | 3% success | | 0% success | |

with 88.90% for AdaptG2WSAT0, 88.32% for AdaptNovelty+, 84.13% for PAWS, 77.39% for G2WSAT and 72.06% for RSAPS. This observation is reinforced in the RTDs on the left-hand of Figure 3 where the gNovelty+ curve dominates over the entire time range.

Overall, gNovelty+ not only outperforms the other techniques in the greatest number of problem classes, it is within an order of magnitude of the best performing algorithms in all remaining cases. It is this robust average case performance (that gNovelty+ also demonstrated in the SAT competition) that argues strongly for its usefulness as a general purpose solver.
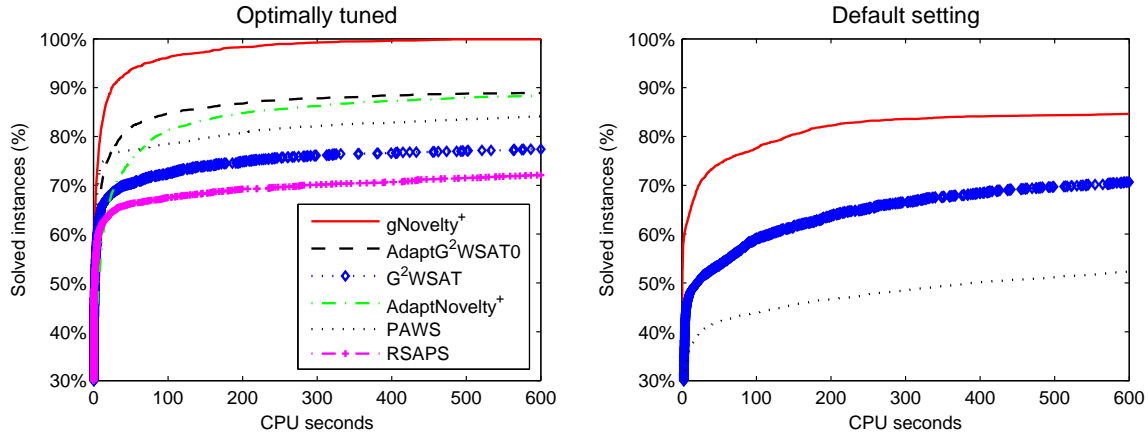
**Figure 3.** Run-time distributions over the complete data set.

However, if such robust behaviour depends critically on manually tuned parameter settings then the case for gNovelty$^+$ must weaken. To evaluate this we tested gNovelty$^+$ on the same problem set with a default $sp$ value of 0 (meaning clause weights are increased in each local minimum but never decreased) and with the noise parameter $p$ adaptively adjusted during the search.[13.] These results and the results of the default parameter values for G$^2$WSAT ($dp = 0.05$ and $p = 0.5$) and PAWS ($w_{inc} = 10$) are shown in Table 5. To give an idea of the relative performance of these default setting algorithms against the other three adaptive ones, the results of AdaptNovelty$^+$, AdaptG$^2$WSAT0 and RSAPS from Table 4 are also reported again in Table 5.

In this second comparison, gNovelty$^+$ remains the champion both in terms of the number of problem classes (bitadd, ais, bw_large, e*ddr, logistics and qg) and the number of instances (19). Table 5 also shows that the performance of gNovelty$^+$, G$^2$WSAT and PAWS (especially the later two) is substantially reduced without parameter tuning, with AdaptG$^2$WSAT0 taking over from PAWS as the winner on all parity problems and beating G$^2$WSAT on the two harder large graph instances. AdaptNovelty$^+$ further dominates on the other large graph instances previously won by PAWS. Consequently, AdaptG$^2$WSAT0 now has the best overall success rate of 88.90% followed by AdaptNovelty$^+$ at 88.32%, the default valued gNovelty$^+$ at 82.23%, RSAPS at (72.06%), with G$^2$WSAT (70.68%) and PAWS (52.32%) coming last (this is also illustrated in the RTDs in Figure 3). Looking in more detail, we can see that the main negative impact of a fixed parameter on gNovelty$^+$ has come from its failure on the parity problems. Similarly, AdaptG$^2$WSAT0 and AdaptNovelty$^+$ fail mainly on the quasi-group problems. If we put these two data sets aside, then the default gNovelty$^+$ shows a clear advantage over AdaptG$^2$WSAT0 and AdaptNovelty$^+$, dominating on five of the remaining seven problem classes.

---

13. Although gNovelty$^+$'s noise parameter was also adjusted in Table 3, performance was not greatly improved, with the main benefits coming from adjusting $sp$.

**Table 5.** Default parameter setting results on structured problems shown in the form: $\frac{median}{mean}$. Best results are marked with ★ and flip counts are reported in thousands. On problems where a solver was timed out for some runs, we report the percentage of success of that solver instead of its CPU time and flip count.

| Instances | gNovelty+ | | AdaptG²WSAT0 | | G²WSAT | | AdaptNovelty+ | | RSAPS | | PAWS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs |
| 2bitadd_11 | 0.928 / 0.947 | 0.000★ / 0.001★ | 2.458 / 2.348 | 0.000 / 0.003 | 0.510★ / 0.677★ | 0.000 / 0.002 | 1.592 / 1.693 | 0.000 / 0.001 | 374 / 577 | 0.230 / 0.368 | 86% success | |
| 2bitadd_12 | 0.576 / 0.625★ | 0.000★ / 0.001★ | 1.716 / 1.882 | 0.000 / 0.003 | 0.394★ / 200 | 0.000 / 0.136 | 1.521 / 1.554 | 0.000 / 0.001 | 59 / 111 | 0.040 / 0.071 | 50,062 / 127,135 | 30.105 / 75.140 |
| 3bitadd_31 | 18★ / 20★ | 0.060★ / 0.068★ | 549 / 817 | 1.120 / 1.625 | 0% success | | 151 / 163 | 0.440 / 0.467 | 0% success | | 0% success | |
| 3bitadd_32 | 15★ / 16★ | 0.060★ / 0.058★ | 262 / 318 | 0.535 / 0.652 | 0% success | | 131 / 135 | 0.450 / 0.466 | 0% success | | 0% success | |
| ais10 | 8.116★ / 12★ | 0.010★ / 0.012★ | 135 / 200 | 0.120 / 0.174 | 140 / 187 | 0.140 / 0.184 | 1,291 / 1,937 | 1.135 / 1.671 | 13 / 18 | 0.020 / 0.020 | 68 / 95 | 0.080 / 0.110 |
| ais12 | 48★ / 64★ | 0.060★ / 0.080★ | 3,127 / 4,320 | 3.445 / 4.754 | 5,128 / 8,732 | 6.245 / 10.673 | 24,315 / 35,292 | 28.140 / 40.833 | 103 / 151 | 0.135 / 0.202 | 917 / 1,424 | 1.410 / 2.166 |
| ais14 | 334★ / 414★ | 0.435★ / 0.537★ | 13,455 / 23,628 | 15.850 / 27.803 | 95% success | | 81% success | | 629 / 892 | 0.885 / 1.265 | 4,205 / 7,108 | 6.795 / 11.530 |
| bw_large.c | 832★ / 1,131★ | 1.485★ / 2.044★ | 3,317 / 4,930 | 3.990 / 5.902 | 49% success | | 6,350 / 9,355 | 6.550 / 9.699 | 3,606 / 5,361 | 10.720 / 15.921 | 7,238 / 9,440 | 17.310 / 22.493 |
| bw_large.d | 4,802★ / 5,499★ | 14.515★ / 16.738★ | 10,600 / 15,714 | 21.550 / 30.726 | 0% success | | 21,462 / 30,616 | 37.055 / 55.237 | 70% success | | 39% success | |
| enddr2-1 | 39★ / 44★ | 0.190★ / 0.196★ | 2,404 / 2,510 | 3.980 / 4.110 | 1,552 / 1,805 | 4.020 / 4.565 | 5,785 / 7,040 | 9.075 / 11.187 | 61 / 72 | 0.505 / 0.534 | 20% success | |
| enddr2-8 | 30★ / 34★ | 0.170★ / 0.180★ | 2,088 / 2,164 | 3.655 / 3.715 | 1,315 / 1,383 | 3.460 / 3.530 | 4,287 / 5,173 | 6.905 / 8.284 | 49 / 54 | 0.495 / 0.509 | 29% success | |
| ewddr2-1 | 32★ / 34★ | 0.190★ / 0.192★ | 1,939 / 2,079 | 3.435 / 3.686 | 1,165 / 1,231 | 3.095 / 3.225 | 4,660 / 6,193 | 7.970 / 10.352 | 48 / 54 | 0.550 / 0.550 | 28% success | |
| ewddr2-8 | 30★ / 32★ | 0.200★ / 0.198★ | 1,757 / 1,828 | 3.365 / 3.454 | 1,131 / 1,164 | 3.205 / 3.276 | 4,959 / 6,515 | 8.690 / 10.959 | 48 / 50 | 0.600 / 0.602 | 38% success | |
| flat200-med | 164 / 241 | 0.070 / 0.099 | 182 / 242 | 0.065 / 0.087 | 140★ / 193 | 0.050★ / 0.073★ | 260 / 392 | 0.085 / 0.131 | 287 / 377 | 0.150 / 0.196 | 145 / 181★ | 0.080 / 0.073 |
| flat200-har | 2,576★ / 4,037★ | 1.040★ / 1.638★ | 3,418 / 5,476 | 1.210 / 1.950 | 4,268 / 13,931 | 1.570 / 4.921 | 17,879 / 22,572 | 6.055 / 7.580 | 3,143 / 4,200 | 1.595 / 2.149 | 5,923 / 8,168 | 3.115 / 4.304 |
| g125.17 | 3,525 / 4,229 | 15.890 / 18.808 | 804★ / 1,161★ | 3.175★ / 4.546★ | 99% success | | 981 / 1,264 | 4.110 / 5.408 | 2% success | | 6% success | |
| g125.18 | 185 / 185 | 0.970 / 0.951 | 52 / 51 | 0.170 / 0.169 | 11★ / 15★ | 0.130 / 0.144 | 35 / 36 | 0.100★ / 0.100★ | 1,057 / 1,787 | 4.995 / 8.379 | 19 / 25 | 0.150 / 0.195 |
| g250.15 | 2.253 / 2.283 | 0.080 / 0.084 | 2.889 / 2.909 | 0.210 / 0.208 | 2.473 / 2.488 | 0.410 / 0.407 | 3.311 / 4.033 | 0.110 / 0.118 | 2.208★ / 2.219★ | 0.080★ / 0.082★ | 2.211 / 2.236 | 0.110 / 0.113 |
| g250.29 | 4,984 / 5,008 | 136.745 / 141.147 | 716★ / 765★ | 8.235★ / 9.147★ | 43% success | | 755 / 895 | 9.960 / 11.979 | 0% success | | 0% success | |
| logistics.c | 6.332★ / 6.873★ | 0.010★ / 0.007★ | 2,550 / 3,631 | 1.385 / 1.996 | 44 / 55 | 0.030 / 0.041 | 122 / 152 | 0.070 / 0.091 | 6.811 / 7.814 | 0.010 / 0.008 | 118 / 158 | 0.100 / 0.128 |
| logistics.d | 27 / 32★ | 0.040★ / 0.042★ | 111 / 137 | 0.090 / 0.109 | 1,775 / 2,946 | 1.520 / 2.560 | 170 / 196 | 0.100 / 0.114 | 23★ / 33 | 0.040 / 0.051 | 275 / 386 | 0.275 / 0.375 |
| par16-1-c | 20% success | | 13,759★ / 18,975★ | 5.300★ / 7.402★ | 99% success | | 17,186 / 32,062 | 5.870 / 10.978 | 73% success | | 23% success | |
| par16-2-c | 10% success | | 129,544★ / 190,040★ | 51.655★ / 75.896★ | 98% success | | 159,405 / 260,240 | 53.415 / 87.516 | 39% success | | 1% success | |
| par16-3-c | 5% success | | 40,233★ / 53,219★ | 16.285★ / 21.554★ | 62,745 / 97,922 | 26.600 / 41.142 | 66,942 / 102,783 | 23.075 / 35.486 | 42% success | | 5% success | |
| par16-4-c | 9% success | | 23,984★ / 39,982★ | 9.445★ / 15.905★ | 99,333 / 158,001 | 41.290 / 65.417 | 80,419 / 112,780 | 27.905 / 38.674 | 69% success | | 16% success | |
| par16-5-c | 5% success | | 23,480★ / 37,596★ | 9.505★ / 15.329★ | 73% success | | 90,820 / 126,329 | 31.225 / 43.841 | 41% success | | 3% success | |
| qg1-08 | 853★ / 1,134★ | 18.900★ / 24.939★ | 99% success | | 14% success | | 99% success | | 59% success | | 83% success | |
| qg2-08 | 3,155★ / 4,093★ | 68.675★ / 91.038★ | 52% success | | 4% success | | 43% success | | 36% success | | 23% success | |
| qg5-11 | 5,012 / 6,863 | 39.115 / 50.497 | 0% success | | 0% success | | 1% success | | 2,287★ / 3,288★ | 24.575★ / 35.405★ | 22% success | |
| qg6-09 | 343 / 2,281 | 1.155 / 7.319 | 5% success | | 17% success | | 14% success | | 29★ / 44★ | 0.110★ / 0.173★ | 968 / 1,229 | 4.035 / 5.102 |
| qg7-13 | 74% success★ | | 0% success | | 0% success | | 0% success | | 3% success | | 0% success | |

## 6. Results with Pre-processing Enhancement

Although preprocessing has a generally negative effect on SLS solvers when solving random problems, it is now well understood that it can produce significant benefits on structured problems [16]. For this reason we decided to test the effects of the two most promising techniques, HyPre [2] and SatELite [5], on the performance of gNovelty+ and its competitors. We also included a simple UnitProp preprocessor as it is cheaper to compute and has been used by G²WSAT and AdaptG²WSAT0. In detail, these preprocessors simplify an input formula before passing the reduced formula to a particular solver as follows:

**UnitProp** simply applies the well-known unit propagation procedure [17] to the input formula until saturation.

**HyPre** [2] focuses on reasoning with binary clauses by implementing the HypBinRes procedure, a restricted version of hyper-resolution [17] that only runs on binary clauses. It also uses the implication graph concept and the HypBinRes rule to infer new binary clauses and avoid the space explosion of computing a full transitive closure. In addition, HyPre incrementally applies unit and equality reductions to infer more binary clauses and hence improve its performance.

**SatELite** [5] uses the (self-)subsumption rule and information about functionally dependent variables to further improve the simplification power of the Variable Elimination Resolution (VER) procedure [4] (a process to eliminate a variable $x$ by replacing all clauses containing $x$ and $\bar{x}$ with their resolvents). Like its predecessor, NiVER [19], SatELite implements the VER process only if there is no increase in the number of literals after variable elimination.

We combined the three preprocessors with each of the default-valued algorithms reported in the previous section, and tested these combinations on all problems that were able to be simplified by a particular preprocessor. These results are summarised in the following sections. Each combination was run 100 times on each instance and each run was timed out after 600 seconds.

## 6.1 Results on UnitProp-simplified Problems

**Table 6.** Default parameter setting results on structured problems preprocessed with UnitProp-agation preprocessor: $\frac{median}{mean}$. Best results are marked with ★ and flip counts are reported in thousands. On problems where a solver was timed out for some runs, we report the percentage of success of that solver instead of its CPU time and flip count. The time taken to preprocess a problem instance is included in the CPU time of each solver. Results on the problems where the preprocessor makes no change to the CNF formulae are omitted.

| Instances | gNovelty$^+$ | | AdaptG$^2$WSAT0 | | G$^2$WSAT | | AdaptNovelty$^+$ | | RSAPS | | PAWS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs |
| enddr2-1 | 22★ / 26★ | 0.120★ / 0.127★ | 206 / 428 | 0.465 / 0.776 | 555 / 676 | 1.135 / 1.386 | 346 / 546 | 0.665 / 1.012 | 31 / 41 | 0.255 / 0.277 | 27% success | |
| enddr2-8 | 18★ / 20★ | 0.100★ / 0.103★ | 142 / 165 | 0.365 / 0.398 | 183 / 203 | 0.490 / 0.531 | 307 / 365 | 0.590 / 0.688 | 26 / 32 | 0.240 / 0.245 | 24% success | |
| ewddr2-1 | 19★ / 19★ | 0.100★ / 0.103★ | 134 / 190 | 0.350 / 0.428 | 202 / 261 | 0.520 / 0.617 | 217 / 268 | 0.430 / 0.517 | 24 / 28 | 0.240 / 0.244 | 44% success | |
| ewddr2-8 | 14★ / 15★ | 0.090★ / 0.089★ | 118 / 123 | 0.310 / 0.318 | 81 / 98 | 0.300 / 0.326 | 205 / 225 | 0.390 / 0.422 | 21 / 23 | 0.220 / 0.223 | 40% success | |
| qg1-08 | 597 / 757 | 2.865 / 3.661 | 307★ / 445★ | 1.045★ / 1.517★ | 1,244 / 1,553 | 5.470 / 6.867 | 467 / 675 | 1.280 / 1.861 | 2,280 / 3,285 | 9.240 / 13.302 | 583 / 873 | 2.160 / 3.213 |
| qg2-08 | 1,278★ / 1,679★ | 6.515 / 8.658 | 1,436 / 2,026 | 5.095★ / 7.180★ | 8,126 / 9,988 | 36.605 / 44.804 | 2,288 / 3,019 | 6.420 / 8.488 | 6,370 / 8,453 | 26.235 / 34.857 | 2,436 / 3,109 | 9.725 / 12.848 |
| qg5-11 | 65 / 89★ | 0.350 / 0.470★ | 52% success | | 1,076 / 3,332 | 4.800 / 12.053 | 4,189 / 12,531 | 10.305 / 28.065 | 62★ / 102 | 0.320★ / 0.518 | 75 / 115 | 0.395 / 0.612 |
| qg6-09 | 3.343 / 4.335 | 0.010 / 0.009 | 1,146 / 2,664 | 1.370 / 3.143 | 8.285 / 16 | 0.020 / 0.029 | 416 / 630 | 0.565 / 0.870 | 2,955 / 3,282 | 0.010 / 0.007 | 2.277★ / 3.105★ | 0.010★ / 0.007★ |
| qg7-13 | 580★ / 741★ | 4.835★ / 5.789★ | 22% success | | 41% success | | 73% success | | 4,155 / 5,318 | 36.455 / 46.256 | 5,200 / 7,557 | 45.170 / 66.118 |

The results in Table 6 show that UnitProp only had an effect on the e*ddr and qg problems and that gNovelty$^+$ remains the dominant algorithm on these simplified instances. Specifically, gNovelty$^+$ had the best time performance on all 4 of the e*ddr problems and

2 of the 5 qg problems, with AdaptG$^2$WSAT0 and PAWS dominating on the remaining qg problems.

UnitProp had a beneficial effect for all algorithms on these problems (compared to the non-preprocessed results of Table 5 and graphed in Figure 4), producing significant improvements for AdaptNovelty$^+$ and the G$^2$WSAT algorithms on the e*ddr problems and across the board improvements on the qg problems. Overall, the benefits of UnitProp for gNovelty$^+$ are less dramatic than for other techniques. However, this can be explained by the fact that gNovelty$^+$ was already doing well on these problems (without preprocessing), and that margin for improvement was consequently smaller.

## 6.2 Results on HyPre-simplified Problems

Table 7 shows that HyPre was able to simplify all the problems in our test set with the exception of the two flat graph colouring problems. Again gNovelty$^+$ remains dominant, having the best performance on all the bitadd, ais, bw_large, e*ddr and logistics problems and 2 of the 5 qg problems. Of the other algorithms, AdaptG$^2$WSAT0 remained dominant on all the parity and 2 of the 4 large graph colouring problems, while improving over its non-preprocessed performance to win on 2 of the qg problems. Finally, AdaptNovelty$^+$ also improved over its non-preprocessed performance to win on 2 large graph colouring problems and PAWS improved to win on qg5-11.

As with UnitProp, the HyPre simplified formulae have been generally easier to solve than the original problems. However, the overhead of HyPre has outweighed these benefits on those problems that can already be solved relatively quickly without preprocessing. For example, the small improvements in the number of flips for gNovelty$^+$ on the e*ddr problems has been obtained at the cost of a more than 10 times increase in execution time (see Table 5 and the comparative graphs in Figure 4).

## 6.3 Results on SatELite-simplified Problems

The SatELite results in Table 8 show a similar pattern to the HyPre results, with gNovelty$^+$ dominating the bitadd, ais, bw_large, e*ddr and logistics problems and 3 of the 5 qg problems. This time, however, AdaptNovelty$^+$ clearly dominated the parity problems (achieving the best results of all the methods and preprocessing combinations tried on this problem class) and further dominated on 3 of the 4 large graph colouring problems and 1 of the flat graph colouring problems. This made AdaptNovelty$^+$ the second best performing SatElite-enhanced algorithm (behind gNovelty$^+$).

SatELite had the widest range of application of the three preprocessing techniques and was able to simplify all 31 problem instances. However, like HyPre, despite generally improving the flip rates of most algorithms on most problems, the overhead of using SatELite caused a deterioration in time performance on many instances. This is shown more clearly in Figure 4 where SatELite consistently appears as one of the worse options for any algorithm on the bitadd and large graph (g) problems.

**Table 7.** Default parameter setting results on structured problems preprocessed with HyPre preprocessor: $\frac{median}{mean}$. Best results are marked with ★ and flip counts are reported in thousands. The time taken to preprocess a problem instance is included in the CPU time of each solver. On problems where a solver was timed out for some runs, we report the percentage of success of that solver instead of its CPU time and flip count. Results on the problems where the preprocessor makes no change to the CNF formulae are omitted.

| Instances | gNovelty$^+$ | | AdaptG$^2$WSAT0 | | G$^2$WSAT | | AdaptNovelty$^+$ | | RSAPS | | PAWS | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs | #flips | #secs |
| 2bitadd_11 | 0.260★ / 0.275★ | 0.000★ / 0.000★ | 1.867 / 1.960 | 0.000 / 0.004 | 0.466 / 0.603 | 0.000 / 0.003 | 3.679 / 4.415 | 0.000 / 0.002 | 0.273 / 0.286 | 0.000 / 0.000 | 0.462 / 0.948 | 0.000 / 0.001 |
| 2bitadd_12 | 0.228★ / 0.231★ | 0.000★ / 0.000★ | 1.514 / 1.692 | 0.000 / 0.004 | 0.392 / 400 | 0.000 / 0.306 | 2.967 / 3.255 | 0.000 / 0.002 | 0.252 / 0.252 | 0.000 / 0.000 | 0.290 / 0.470 | 0.000 / 0.001 |
| 3bitadd_31 | 12★ / 12★ | 0.430★ / 0.433★ | 35 / 37 | 0.490 / 0.490 | 16 / 17 | 0.500 / 0.502 | 369 / 413 | 0.815 / 0.866 | 18 / 20 | 0.450 / 0.454 | 0% success | |
| 3bitadd_32 | 9.457★ / 9.428★ | 0.440★ / 0.440★ | 30 / 31 | 0.500 / 0.497 | 13 / 113 | 0.510 / 0.832 | 255 / 276 | 0.690 / 0.711 | 13 / 13 | 0.450 / 0.450 | 7% success | |
| ais10 | 11★ / 13★ | 0.010★ / 0.011★ | 92 / 131 | 0.070 / 0.100 | 85 / 116 | 0.080 / 0.097 | 1,017 / 1,524 | 0.710 / 1.060 | 13 / 17 | 0.010 / 0.015 | 28 / 42 | 0.030 / 0.039 |
| ais12 | 64★ / 77★ | 0.080★ / 0.087★ | 1,182 / 1,568 | 1.100 / 1.467 | 1,375 / 1,864 | 1.440 / 1.934 | 12,923 / 18,165 | 11.840 / 16.480 | 98 / 141 | 0.110 / 0.157 | 320 / 516 | 0.380 / 0.617 |
| ais14 | 372★ / 449★ | 0.445★ / 0.533★ | 15,312 / 21,691 | 16.840 / 24.058 | 94% success | | 85% success | | 669 / 863 | 0.835 / 1.092 | 3,875 / 6,373 | 5.620 / 9.241 |
| bw_large.c | 38★ / 58 | 1.440★ / 1.460★ | 1,831 / 2,600 | 2.925 / 3.595 | 1,441 / 2,210 | 2.960 / 3.750 | 4,716 / 6,993 | 5.480 / 7.503 | 43 / 56★ | 1.450 / 1.466 | 74 / 103 | 1.490 / 1.529 |
| bw_large.d | 508★ / 725★ | 11.890★ / 12.449★ | 22,061 / 25,059 | 57.255 / 61.824 | 53% success | | 96% success | | 2,218 / 3,799 | 19.985 / 26.924 | 1,302 / 1,842 | 16.255 / 18.665 |
| enddr2-1 | 31★ / 35★ | 6.010★ / 6.016★ | 1,400 / 1,475 | 8.040 / 8.145 | 963 / 1,065 | 7.650 / 7.817 | 4,614 / 5,747 | 12.445 / 14.091 | 48 / 61 | 6.150 / 6.183 | 57% success | |
| enddr2-8 | 21★ / 22★ | 5.820★ / 5.825★ | 450 / 531 | 6.610 / 6.716 | 416 / 427 | 6.640 / 6.662 | 571 / 864 | 6.615 / 6.999 | 33 / 38 | 5.950 / 5.965 | 48% success | |
| ewddr2-1 | 21★ / 23★ | 5.440★ / 5.437★ | 635 / 703 | 6.435 / 6.527 | 458 / 498 | 6.380 / 6.447 | 1,990 / 3,117 | 8.105 / 9.667 | 31 / 35 | 5.570 / 5.578 | 52% success | |
| ewddr2-8 | 15★ / 17★ | 5.020★ / 5.022★ | 300 / 347 | 5.590 / 5.631 | 180 / 218 | 5.440 / 5.501 | 364 / 502 | 5.465 / 5.641 | 26 / 31 | 5.150 / 5.151 | 56% success | |
| g125.17 | 3,546 / 4,512 | 13.085 / 16.252 | 717★ / 988★ | 1.875★ / 2.520★ | 16,308 / 20,545 | 62.780 / 77.955 | 931 / 1,488 | 2.555 / 4.135 | 1% success | | 8% success | |
| g125.18 | 160 / 173 | 1.070 / 1.107 | 48 / 49 | 0.230 / 0.230 | 9.607★ / 13★ | 0.200 / 0.213 | 37 / 38 | 0.170★ / 0.170★ | 1,306 / 1,911 | 6.420 / 9.370 | 19 / 23 | 0.205 / 0.226 |
| g250.15 | 2.261 / 2.291 | 0.530 / 0.529 | 2,885 / 2,933 | 0.680 / 0.677 | 2,410 / 2,475 | 0.770 / 0.773 | 3,217 / 3,599 | 0.520★ / 0.526★ | 2,182★ / 2,200★ | 0.530 / 0.530 | 2,216 / 2,225 | 0.540 / 0.536 |
| g250.29 | 4,796 / 4,910 | 73.060 / 74.602 | 748★ / 866★ | 6.400★ / 7.388★ | 67% success | | 778 / 892 | 8.825 / 10.354 | 0% success | | 0% success | |
| logistics.c | 1.347★ / 1.486 | 0.020★ / 0.022★ | 8.077 / 9.357 | 0.030 / 0.031 | 2.898 / 3.306 | 0.030 / 0.027 | 6.687 / 7.715 | 0.020 / 0.025 | 1,369 / 1,478★ | 0.020 / 0.022 | 3.242 / 5.152 | 0.020 / 0.025 |
| logistics.d | 3.726★ / 3.884★ | 0.850★ / 0.851★ | 27 / 29 | 0.900 / 0.900 | 6.785 / 7.123 | 0.890 / 0.889 | 34 / 37 | 0.870 / 0.867 | 3,780 / 3,915 | 0.850 / 0.854 | 4.472 / 7.067 | 0.860 / 0.857 |
| par16-1-c | 16% success | | 14,787★ / 20,756★ | 6.005★ / 8.453★ | 99% success | | 18,151 / 29,863 | 6.375 / 10.512 | 86% success | | 16% success | |
| par16-2-c | 4% success | | 83,478★ / 122,795★ | 33.030★ / 49.337★ | 95% success | | 124,928 / 176,082 | 44.130 / 61.248 | 54% success | | 1% success | |
| par16-3-c | 5% success | | 36,517★ / 42,956★ | 14.395★ / 17.183★ | 64,224 / 87,618 | 27.460 / 37.512 | 63,339 / 92,704 | 22.715 / 33.117 | 70% success | | 4% success | |
| par16-4-c | 6% success | | 18,058★ / 31,792★ | 7.480★ / 13.064★ | 76,621 / 147,878 | 32.830 / 62.198 | 32,438 / 55,159 | 11.345 / 19.297 | 79% success | | 7% success | |
| par16-5-c | 3% success | | 25,234★ / 42,119★ | 10.215★ / 17.261★ | 71% success | | 52,581 / 75,997 | 18.140 / 26.487 | 68% success | | 0% success | |
| qg1-08 | 553 / 673 | 2.950 / 3.532 | 366★ / 483★ | 1.610★ / 2.019★ | 1,096 / 1,582 | 5.335 / 7.459 | 472 / 813 | 1.620 / 2.529 | 1,743 / 2,495 | 7.185 / 10.133 | 576 / 813 | 2.425 / 3.311 |
| qg2-08 | 1,437 / 1,796 | 7.460 / 9.346 | 868★ / 1,326★ | 3.500★ / 5.121★ | 5,489 / 7,129 | 25.520 / 33.435 | 1,841 / 2,557 | 5.540 / 7.463 | 6,894 / 8,319 | 28.685 / 34.478 | 1,688 / 2,287 | 6.940 / 9.336 |
| qg5-11 | 20 / 28 | 0.320 / 0.352 | 1,559 / 2,913 | 3.745 / 6.807 | 51 / 88 | 0.480 / 0.645 | 570 / 1,558 | 1.615 / 3.923 | 20 / 27 | 0.335 / 0.359 | 15★ / 22★ | 0.315★ / 0.351★ |
| qg6-09 | 0.001★ / 0.001★ | 0.050★ / 0.050★ | 0.001 / 0.001 | 0.050 / 0.052 | 0.001 / 0.001 | 0.050 / 0.051 | 0.001 / 0.009 | 0.050 / 0.050 | 0.001 / 0.002 | 0.050 / 0.050 | 0.001 / 0.002 | 0.050 / 0.050 |
| qg7-13 | 250★ / 355★ | 2.310★ / 3.178★ | 95% success | | 975 / 1,299 | 5.960 / 8.057 | 99% success | | 404 / 688 | 3.700 / 5.858 | 407 / 589 | 3.650 / 5.166 |

## 6.4 Evaluation of Preprocessing

Overall it is not immediately clear whether preprocessing is a useful general purpose addition for our algorithms. Of the three techniques, only UnitProp has a consistently positive effect, even though this is limited to two problem classes (e*ddr and qg). Although both SatELite and HyPre have positive effects on certain problems for certain algorithms, neither of them is able to provide an overall improvement for all tested solvers across the whole benchmark set. For instance, HyPre is generally helpful on the qg problems and SatELite is helpful on the flat graph colouring problems. But arrayed against these gains are the unpredictable

**Table 8.** Default parameter setting results on structured problems preprocessed with SatELite preprocessor: $\frac{median}{mean}$. Best results are marked with ★ and flip counts are reported in thousands. The time taken to preprocess a problem instance is included in the CPU time of each solver. On problems where a solver was timed out for some runs, we report the percentage of success of that solver instead of its CPU time and flip count.

| Instances | gNovelty+ #flips | gNovelty+ #secs | AdaptG²WSAT0 #flips | AdaptG²WSAT0 #secs | G²WSAT #flips | G²WSAT #secs | AdaptNovelty+ #flips | AdaptNovelty+ #secs | RSAPS #flips | RSAPS #secs | PAWS #flips | PAWS #secs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2bitadd_11 | 1.116 / 1.330 | 0.042★ / 0.043★ | 7.848 / 16 | 0.052 / 0.051 | 0.679★ / 0.993★ | 0.042 / 0.044 | 1.853 / 3.950 | 0.042 / 0.044 | 20 / 32 | 0.062 / 0.066 | 5,332 / 8,425 | 3.802 / 6.075 |
| 2bitadd_12 | 0.686 / 0.772★ | 0.049★ / 0.050★ | 4.646 / 10 | 0.059 / 0.056 | 0.421★ / 500 | 0.049 / 0.389 | 1.511 / 1.569 | 0.049 / 0.050 | 5.179 / 11 | 0.049 / 0.057 | 950 / 2,864 | 0.724 / 2.069 |
| 3bitadd_31 | 32★ / 41★ | 2.228★ / 2.257★ | 0% success | | 0% success | | 249 / 298 | 2.803 / 2.887 | 0% success | | 0% success | |
| 3bitadd_32 | 17★ / 18★ | 2.351★ / 2.359★ | 1% success | | 0% success | | 147 / 149 | 2.811 / 2.811 | 0% success | | 0% success | |
| ais10 | 8.504★ / 10★ | 0.044★ / 0.050★ | 12 / 15 | 0.054 / 0.059 | 10 / 13 | 0.054 / 0.056 | 18 / 27 | 0.064 / 0.077 | 11 / 14 | 0.054 / 0.061 | 23 / 37 | 0.084 / 0.117 |
| ais12 | 62★ / 73★ | 0.168★ / 0.191★ | 78 / 127 | 0.203 / 0.292 | 108 / 138 | 0.268 / 0.331 | 138 / 187 | 0.348 / 0.451 | 77 / 96 | 0.253 / 0.299 | 324 / 381 | 1.013 / 1.199 |
| ais14 | 447 / 572★ | 1.010★ / 1.272★ | 3,104 / 4,795 | 5.925 / 9.119 | 1,267 / 1,832 | 2.675 / 3.858 | 689 / 1,008 | 1.510 / 2.181 | 417★ / 696 | 1.290 / 2.098 | 9% success | |
| bw_large.c | 960★ / 1,113★ | 2.175★ / 2.414★ | 4,100 / 5,697 | 5.220 / 7.218 | 39% success | | 5,475 / 8,335 | 6.975 / 10.557 | 4,431 / 6,050 | 11.685 / 16.166 | 5,428 / 9,368 | 13.975 / 24.912 |
| bw_large.d | 4,479 / 5,690★ | 13.510 / 16.524★ | 5,604 / 8,965 | 11.405★ / 17.520 | 0% success | | 21,199 / 25,744 | 40.260 / 48.444 | 61% success | | 36% success | |
| enddr2-1 | 12 / 16★ | 1.043★ / 1.049★ | 104 / 120 | 1.198 / 1.213 | 41 / 51 | 1.173 / 1.186 | 36,885 / 55,561 | 44.478 / 66.173 | 12★ / 18 | 1.053 / 1.066 | 53% success | |
| enddr2-8 | 10★ / 12★ | 1.071★ / 1.071★ | 101 / 107 | 1.191 / 1.201 | 31 / 38 | 1.181 / 1.192 | 19,861 / 26,945 | 23.706 / 31.876 | 10 / 13 | 1.081 / 1.090 | 53% success | |
| ewddr2-1 | 6.833 / 8.426 | 1.108★ / 1.112★ | 56 / 58 | 1.198 / 1.198 | 12 / 15 | 1.198 / 1.203 | 3,773 / 8,515 | 5.288 / 10.578 | 6.041★ / 7.381★ | 1.118 / 1.122 | 75% success | |
| ewddr2-8 | 5.734★ / 6.554★ | 1.149★ / 1.146★ | 55 / 57 | 1.239 / 1.241 | 10 / 12 | 1.229 / 1.232 | 1,410 / 4,468 | 2.719 / 6.056 | 5.973 / 6.713 | 1.159 / 1.157 | 69% success | |
| flat200-med | 106★ / 147★ | 0.114 / 0.135 | 195 / 298 | 0.149 / 0.193 | 266 / 416 | 0.189 / 0.269 | 125 / 163 | 0.109★ / 0.126★ | 398 / 621 | 0.329 / 0.489 | 114 / 160 | 0.129 / 0.168 |
| flat200-har | 1,180★ / 1,822★ | 0.729★ / 1.105★ | 2,306 / 3,881 | 1.154 / 1.891 | 2,886 / 4,399 | 1.544 / 2.329 | 1,674 / 2,347 | 0.809 / 1.117 | 3,408 / 4,695 | 2.409 / 3.309 | 1,465 / 2,014 | 1.124 / 1.516 |
| g125.17 | 89% success | | 0% success | | 0% success | | 1,190★ / 1,759★ | 16.811★ / 23.931★ | 0% success | | 9% success | |
| g125.18 | 277 / 284 | 8.409 / 8.511 | 1,103 / 1,590 | 19.384 / 27.250 | 56 / 81 | 3.674 / 4.391 | 35★ / 36★ | 2.389★ / 2.403★ | 1,545 / 2,249 | 29.744 / 44.530 | 91% success | |
| g250.15 | 2.806 / 2.935 | 8.928 / 8.934 | 357 / 375 | 23.833 / 24.305 | 3.465 / 3.551 | 9.518 / 9.521 | 4.032 / 4.181 | 8.938 / 8.945 | 2.631★ / 2.676★ | 8.888★ / 8.892★ | 98% success | |
| g250.29 | 66% success | | 0% success | | 0% success | | 827★ / 958★ | 68.974★ / 75.745★ | 0% success | | 0% success | |
| logistics.c | 2.910 / 3.293 | 0.273★ / 0.276★ | 72 / 93 | 0.323 / 0.332 | 9.260 / 11 | 0.283 / 0.285 | 22 / 28 | 0.293 / 0.291 | 2.538★ / 2.982★ | 0.273 / 0.276 | 7.479 / 11 | 0.283 / 0.283 |
| logistics.d | 14 / 16 | 0.552★ / 0.557★ | 380 / 513 | 0.857 / 0.973 | 707 / 942 | 1.292 / 1.535 | 130 / 172 | 0.637 / 0.669 | 12★ / 16★ | 0.552 / 0.559 | 31 / 57 | 0.582 / 0.606 |
| par16-1-c | 81% success | | 8,211 / 13,494 | 4.339 / 7.152 | 11,284 / 16,358 | 6.494 / 9.037 | 5,870★ / 8,113★ | 2.794★ / 3.848★ | 99% success | | 115,457 / 146,796 | 60.474 / 77.891 |
| par16-2-c | 35% success | | 40,270 / 68,949 | 21.563 / 36.520 | 33,095 / 47,439 | 18.468 / 26.128 | 32,693★ / 44,368★ | 15.663★ / 21.372★ | 90% success | | 48% success | |
| par16-3-c | 42% success | | 17,316★ / 23,186 | 9.365 / 12.566★ | 18,549 / 23,013★ | 10.820 / 13.027 | 18,588 / 25,850 | 9.240★ / 12.791 | 97% success | | 88% success | |
| par16-4-c | 47% success | | 14,469★ / 20,584★ | 7.581 / 10.873 | 23,399 / 31,199 | 13.131 / 17.307 | 15,549 / 22,520 | 7.566★ / 10.864★ | 99% success | | 69,676 / 105,256 | 38.696 / 57.880 |
| par16-5-c | 40% success | | 14,239 / 18,460★ | 7.715 / 10.017 | 19,506 / 33,987 | 11.025 / 18.486 | 11,341★ / 19,486 | 5.425★ / 9.299★ | 93% success | | 84% success | |
| qg1-08 | 366 / 522 | 2.381 / 3.207 | 283★ / 413★ | 1.371★ / 1.806★ | 1,154 / 1,687 | 5.236 / 7.412 | 498 / 662 | 1.741 / 2.164 | 1,822 / 2,671 | 7.566 / 10.970 | 455 / 721 | 2.056 / 3.005 |
| qg2-08 | 1,370★ / 1,738★ | 7.388 / 9.243 | 1,494 / 2,037 | 5.788★ / 7.670★ | 8,349 / 10,682 | 37.903 / 48.264 | 2,462 / 3,111 | 7.598 / 9.500 | 4,517 / 8,122 | 19.483 / 34.707 | 2,142 / 3,307 | 9.133 / 14.147 |
| qg5-11 | 71 / 83★ | 0.505★ / 0.573★ | 48% success | | 1,092 / 1,991 | 4.645 / 7.081 | 5,488 / 11,791 | 12.475 / 26.332 | 67★ / 108 | 0.535 / 0.750 | 76 / 123 | 0.570 / 0.807 |
| qg6-09 | 2.732★ / 3.856 | 0.062★ / 0.060★ | 1,237 / 2,523 | 1.537 / 3.069 | 8.046 / 15 | 0.072 / 0.079 | 367 / 720 | 0.567 / 1.048 | 2.870 / 3.546★ | 0.062 / 0.060 | 2.925 / 4.241 | 0.062 / 0.062 |
| qg7-13 | 658★ / 823★ | 5.254★ / 6.669★ | 23% success | | 30% success | | 76% success | | 3,601 / 5,099 | 35.324 / 44.504 | 4,481 / 6,718 | 37.524 / 56.072 |

worsening effects of the more complex preprocessors on other problem classes. For instance, consider the negative effect of SatELite on AdaptG²WSAT0 on the bitadd and the large graph colouring problems.

If we take the entire picture presented in Figure 4 two observations emerge. Firstly, gNovelty+ achieves the best overall performance regardless of the preprocessor used, and secondly, of the preprocessors, only UnitProp is able to improve the overall performance of

gNovelty$^+$. Therefore, our final recommendation from the preprocessor study would be to use gNovelty$^+$ in conjunction with UnitProp.
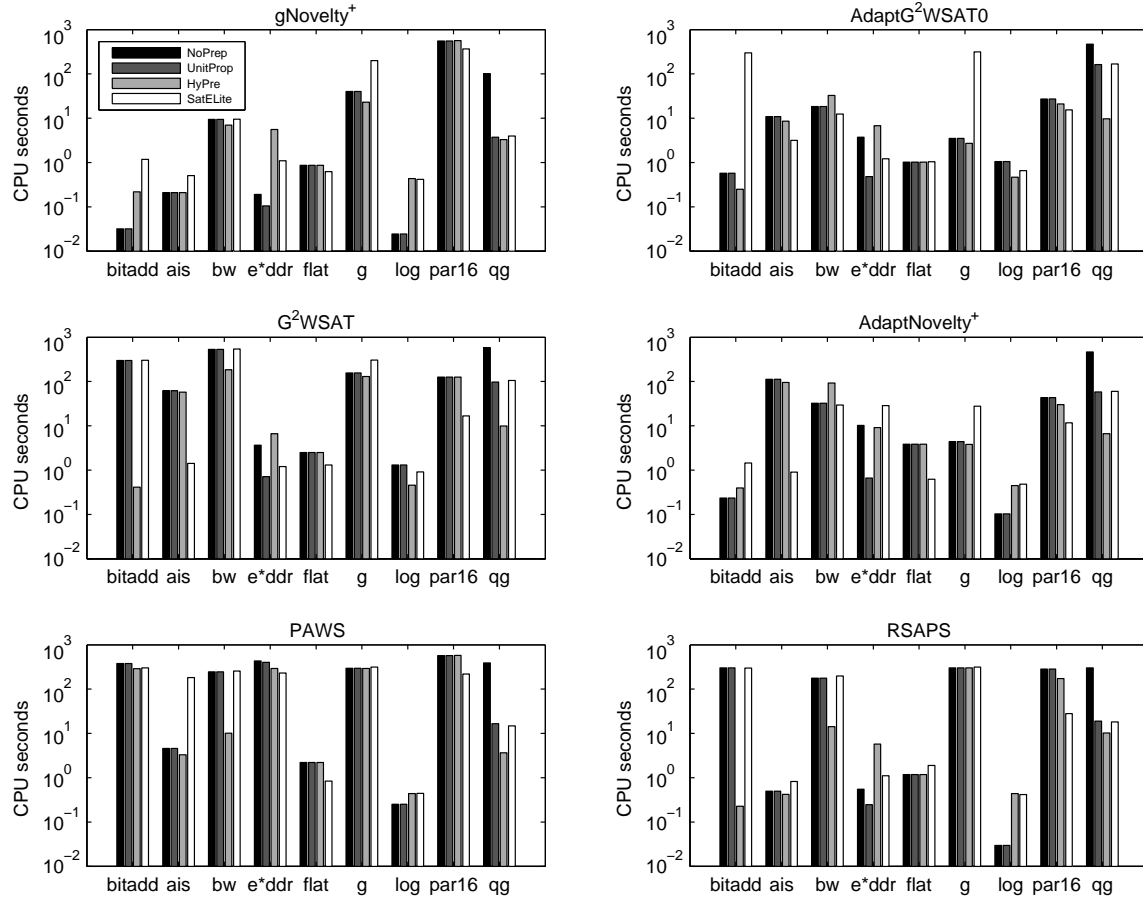


**Figure 4.** Comparing the performance of solvers (default settings) on the whole benchmark set with NoPrep, UnitProp, HyPre and SatELite. Data is mean CPU time (logarithmic scale).

## 7. Discussion and Conclusions

The experimental evidence of this paper and the 2007 SAT competition demonstrates that gNovelty$^+$ is a highly competitive algorithm for random SAT problems. In addition, these results show that gNovelty$^+$, with parameter tuning, can dominate several of the previously best performing SLS algorithms on a range of structured problems. If parameter tuning is ruled out (as it would be in most real-world problem scenarios), then gNovelty$^+$ still performs well, and only lost to its closest rival, AdaptG$^2$WSAT0, on one structured problem class.

Once again, as with PAWS and SAPS, the addition of a clause weighting heuristic to gNovelty$^+$ has required the addition of a sensitive weight decay parameter to get competitive

results. Nevertheless, the situation with gNovelty$^+$'s parameter does differ from SAPS and PAWS in that highly competitive performance can be obtained from a relatively small set of parameter values (i.e. 0.0, 0.1, 0.4 and 1.0). In contrast, SAPS and PAWS require much finer distinctions in parameter values to get even acceptable results [20]. This smaller set of values means that the process of tuning the smoothing parameter $sp$ of gNovelty$^+$ is considerably simpler than for other clause weight techniques. More importantly, the robust behaviour of gNovelty$^+$ indicates that it may be easier to devise an automatic adapting mechanism for $sp$. To date, procedures for automatically adapting weight decay parameters have not produced the fastest algorithms.[14] In future work, it therefore appears promising to try and develop a simple heuristic that will effectively adapt $sp$ in the structured problem domain.

Finally we examined the effects of preprocessing on the performance of the algorithms used in the study. Here we found that two of the best known modern preprocessing techniques (HyPre and SatELite) produced mixed results and had an overall negative impact on execution time across the whole problem set. These results appear to go against other work in [16] that found HyPre and SatELite to be generally beneficial for local search on SAT. However, in the current study many of the problems were solved quickly relative to the overhead of using the more complex preprocessors. If we only consider flip rates then HyPre and SatELite did show a generally positive effect. This means that for problems where execution times become large relative to the overhead of preprocessing, we would expect both HyPre and SatELite to show greater improvements. Nevertheless, within the confines of the current study, the simpler UnitProp preprocessing method (in conjunction with gNovelty$^+$) had the best overall results: even though UnitProp only had positive effects on two problem classes this was balanced by the fact that its overhead on other problems was relatively insignificant.

In conclusion, we have introduced gNovelty$^+$, a new hybrid SLS solver that won the random SAT category in the 2007 SAT competition. We have extended the SAT results and shown that gNovelty$^+$ is also effective in solving structured SAT problems. In fact, gNovelty$^+$ has not only outperformed five of the strongest current SLS SAT solvers, it has also demonstrated significant robustness in solving a wide range of diverse problems. In achieving this performance, we have highlighted gNovelty$^+$'s partial dependence on the setting of its $sp$ smoothing parameter. This leads us to recommend that future work should concentrate on the automatic adaptation of this parameter.

## Acknowledgements

---

14. Although machine learning techniques that are trained on test sets of existing instances and then applied to unseen instances have proved useful for setting SAPS and Novelty parameters [10]

## References

[1] Anbulagan, Duc Nghia Pham, John Slaney, and Abdul Sattar. Old resolution meets modern SLS. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 354–359, 2005.

[2] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT-03)*, **2919** of *Lecture Notes in Computer Science (LNCS)*, pages 341–355, 2003.

[3] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, **5**(7):394–397, 1962.

[4] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, **7**:201–215, 1960.

[5] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT-05)*, **3569** of *Lecture Notes in Computer Science (LNCS)*, pages 61–75, 2005.

[6] Fred Glover. Tabu search - part 1. *ORSA Journal on Computing*, **1**(3):190–206, 1989.

[7] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 661–666, 1999.

[8] Holger H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 635–660, 2002.

[9] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, 2005.

[10] Frank Hutter, Youssef Hamadi, Holger H. Hoos, and Kevin Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP-06)*, **4204** of *Lecture Notes in Computer Science (LNCS)*, pages 213–228, 2006.

[11] Frank Hutter, Dave A.D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP-02)*, **2470** of *Lecture Notes in Computer Science (LNCS)*, pages 233–248, 2002.

[12] Chu Min Li and Wen Qi Huang. Diversification and determinism in local search for satisfiability. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT-05)*, **3569** of *Lecture Notes in Computer Science (LNCS)*, pages 158–172, 2005.

[13] Chu Min Li, Wanxia Wei, and Harry Zhang. Combining adaptive noise and look-ahead in local search for SAT. In *Proceedings of the Third International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS-06)*, pages 2–16, 2006.

[14] David A. McAllester, Bart Selman, and Henry A. Kautz. Evidence for invariants in local search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 321–326, 1997.

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

[16] Duc Nghia Pham. *Modelling and Exploiting Structures in Solving Propositional Satisfiability Problems.* PhD thesis, Griffith University, Queensland, Australia, 2006.

[17] John Alan Robinson. Automated deduction with hyper-resolution. *Journal of Computer Mathematics*, **1**(3):227–234, 1965.

[18] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.

[19] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER: Non increasing variable elimination resolution for preprocessing SAT instances. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT-04)*, **3542** of *Lecture Notes in Computer Science (LNCS)*, pages 276–291, 2004.

[20] John R. Thornton. Clause weighting local search for SAT. *Journal of Automated Reasoning*, **35**(1-3):97–142, 2005.

[21] John R. Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-04)*, pages 191–196, 2004.