Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence

# Weight-Enhanced Diversification in Stochastic Local Search for Satisfiability

**Thach-Thao Duong** and **Duc Nghia Pham** and **Abdul Sattar** and **M.A. Hakim Newton**
Institute for Integrated and Intelligent Systems, Griffith University, Australia and NICTA Australia
{t.duong,d.pham,a.sattar,hakim.newton}@griffith.edu.au

## Abstract

Intensification and diversification are the key factors that control the performance of stochastic local search in satisfiability (SAT). Recently, Novelty Walk has become a popular method for improving diversification of the search and so has been integrated in many well-known SAT solvers such as TNM and gNovelty$^+$. In this paper, we introduce new heuristics to improve the effectiveness of Novelty Walk in terms of reducing search stagnation. In particular, we use weights (based on statistical information collected during the search) to focus the diversification phase onto specific areas of interest. With a given probability, we select the most frequently unsatisfied clause instead of a totally random one as Novelty Walk does. Amongst all the variables appearing in the selected clause, we then select the least flipped variable for the next move. Our experimental results show that the new weight-enhanced diversification method significantly improves the performance of gNovelty$^+$ and thus outperforms other local search SAT solvers on a wide range of structured and random satisfiability benchmarks.

## 1 Introduction

Stochastic Local Search (SLS) for Satisfiability (SAT) has obtained significant progress in recent years through the development of many powerful solvers such as G$^2$WSAT [Li and Huang, 2005], VW2 [Prestwich, 2005], gNovelty$^+$[Pham *et al.*, 2008], TNM [Wei and Li, 2009] , Sparrow [Balint and Fröhlich, 2010], and CCASat [Cai and Su, 2012]. However, their success has been restricted mostly to randomly generated SAT instances. The results from recent SAT competitions[1] reveal that SLS algorithms are unable to match the performance of systematic solvers on large structured problems. These problems are normally highly constrained than randomly generated problems. Consequently, it becomes harder to navigate the search space due to the large numbers of local minima. In other words, SLS solvers can easily get trapped in a local minimum and takes a long time to escape from it. As a result, the search diversification (i.e. its ability to cover different areas in the search space) on these problems is significantly reduced.

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

A typical approach to improve diversification in local minima is to perform a Random Walk [McAllester *et al.*, 1997]: randomly select an unsatisfied clause and then flip a random variable in that clause. Recently, Novelty Walk (that uses the Novelty[McAllester *et al.*, 1997] heuristic to select the variable) has become a popular method for improving diversification and so has been integrated in many well-known SAT solvers such as TNM and gNovelty$^+$. It is worthy to note that the way Novelty selects a variable from a randomly selected false clause is different to Random Walk. By ranking all variables in the clause w.r.t a scoring function, it flips the best variable most of the time but will occasionally flip the second best variable with some noise probability.

In this paper, we introduce new heuristics to improve the effectiveness of Novelty Walk in terms of reducing search stagnation. These heuristics are used to select (i) an unsatisfied clause and then (ii) a variable to be flipped thereof. We use weights (based on statistical information collected during the search) to focus the diversification phase onto specific areas of interest. Our interests in this case are long-term unsatisfied clauses and variables that have been flipped only few times. With a given probability $\beta$, we select the most frequently unsatisfied clause instead of a totally random one as Novelty does. Amongst all the variables appearing in the selected clause, we then select the least flipped variable for the next move. By doing this, we intend to explore diverse areas of the search space.

We applied our weight-enhanced diversification heuristics on the ternary chain problem to examine their diversification and trap-escaping capability. We also evaluated our heuristics on benchmarks that are real world industrial problems and have complex structures. Moreover, we compared the performance of our resulting solver with other well-known solvers on the benchmarks used in SAT 2011 competition. Our experimental results show that our new method significantly improves the performance of gNovelty$^+$ and thus outperforms other local search SAT solvers on both structured and random satisfiability benchmarks.

The rest of the paper is organised as follows: Section 2 reviews stochastic local search for SAT, Section 3 describes our weight-enhanced diversification algorithm, Section 4 presents our experimental results and analyses, and lastly Section 5 concludes the paper.

## 2 Stochastic Local Search for Satisfiability

Most SLS-based SAT solvers operate in two modes: intensification (or greedy) mode and diversification (or trap-escaping) mode. In the intensification mode, they choose the variable with the highest score to get as close to the solution as possible w.r.t. the scoring function. Thus, selecting the right scoring function to be used in the intensification mode is a key performance-controlling factor for SLS-based solvers. Typically, there are two types of scoring functions: non-weighted or weighted versions.

A popular non-weighted scoring function is based on the simple count of currently unsatisfied clauses [McAllester *et al.*, 1997]:

$$\mathrm{score}(v) = \sum_{c} (\mathrm{Cls}(c) - \mathrm{Cls}^{v}(c))$$

where $\mathrm{Cls}(c)$ indicates whether a clause $c$ is false or not under the current assignment. In order words, $\mathrm{Cls}(c) = 1$ if $c$ is currently unsatisfied, otherwise $\mathrm{Cls}(c) = 0$. Similarly, $\mathrm{Cls}^{v}(c)$ shows whether clause $c$ remains false if variable $v$ is flipped. Clearly, the score of flipping a variable $v$ is defined as the decrease in the number of unsatisfied clauses after $v$ is flipped. This scoring scheme is used by many SLS solvers such as GSAT [Selman *et al.*, 1992], $G^2$WSAT [Li and Huang, 2005], TNM [Wei and Li, 2009].

Under a weighted scheme, a scoring function can be defined as

$$\mathrm{wscore}(v) = \sum_{c} \mathrm{Wgh}(c) \times (\mathrm{Cls}(c) - \mathrm{Cls}^{v}(c))$$

where $\mathrm{Wgh}(c)$ is the weight of a clause $c$. Examples of modern SLS solvers that use clause-weighted scoring functions are gNovelty$^+$ [Pham *et al.*, 2008], Sparrow [Balint and Fröhlich, 2010], and CCASat [Cai and Su, 2012]).

Alternatively, a weighted scoring function can also be simply defined as

$$\mathrm{vscore(v)} = \mathrm{VarWgh}(v)$$

where the weight $\mathrm{VarWgh}(v)$ of a variable $v$ is normally defined as the number of times $v$ has been flipped. VW2 [Prestwich, 2005] is the first SLS solver that relies solely on this variable-weighted score to prioritise and select variables.

The diversification mode is activated when a SLS solver reaches a local minimum, i.e. an area where the scoring function becomes meaningless for filtering good potential moves. Therefore, the search cannot proceed in a greedy fashion at a local minimum. Random Walk and Novelty Walk [McAllester *et al.*, 1997] are the two popular choices to help SLS solvers escape from local minima. Additionally, there are other diversification boosting methods such as variable weighting [Prestwich, 2005] or clause weighting [Thornton *et al.*, 2004; Pham *et al.*, 2008], which have been used mostly in variable selection.

The Hybrid [Wei *et al.*, 2008] and TNM [Wei and Li, 2009] algorithms exploit variable weights to regulate two different noise heuristics, but do not directly use these weights in the selection of variables during the diversification phase. Sparrow, the winner of the SAT 2011 competition, uses its own probabilistic scoring function based on wscore. CCASat [Cai and Su, 2012], the winner of the 2012 SAT Challenge, uses the configuration checking concept to restrict the greedy selection of variables. Recently, [Pham *et al.*, 2012; Duong *et al.*, 2012; 2012] introduced the new pseudo-conflict concept to prevent the search from visiting previously encountered local minima.

### 2.1 gNovelty$^+$

In Algorithm 1, we provide the pseudo-code of gNovelty$^+$ [Pham *et al.*, 2008] to illustrate how a SLS solver switches from its intensification mode to its diversification mode and vice versa. gNovelty$^+$ is a hybrid of PAWS [Thornton *et al.*, 2004] and $G^2$WSAT [Li and Huang, 2005].

**Algorithm 1:** gNovelty$^+$($\Theta$, $sp$)

```
Input  : Formula $\Theta$, smoothing probability $sp$
Output: Solution $\alpha$ (if found) or TIMEOUT
1  randomly generate a candidate solution $\alpha$;
2  initiate all clause weights $\mathrm{Wgh}(c) = 1$;
3  while not timetout do
4      if $\alpha$ satisfies the formula $\Theta$ then return $\alpha$ ;
5      ;
6      if within the random walk probability $wp = 0.01$ then
7          perform a Random Walk;
8      else
9          if there exits promising variables then
10             select a variable $v$ with the highest $\mathrm{wscore}(v)$, breaking ties by
               selecting the least recently flipped one;
11         else
12             perform WeightedNovelty($p$) and adjust its noise $p$;
13             update and smooth (with the probability $sp$) clause weights;
14         end
15     end
16     update the candidate solution $\alpha$;
17 end
18 return TIMEOUT;
```

In its greedy mode, gNovelty$^+$ selects the most *promising* variable $v$ to flip [Pham *et al.*, 2008].[2] Initially, the weights of all clauses are set to 1. If no promising variable can be found, the weights of all unsatisfied clauses will be increased by 1 (Line 13). Within a smoothing probability $sp$, the weights of all *weighted clauses* $c$ (i.e. whose $\mathrm{Wgh}(c)$ is greater than 1) are decreased by 1 (Line 13) [Pham *et al.*, 2008]. Using this clause-weighted scoring scheme, gNovelty$^+$ always aims to select the variable $v$ that after flipped will lead to the maximum reduction in the weighted sum of all false clauses. In other words, gNovelty$^+$ tends to satisfy clauses that have remained unsatisfied for a long time.

When no promising variable exists, gNovelty$^+$ switches to its diversification mode. In this mode, it invokes WeightedNovelty, a weighted version of Novelty, to help lessen its greediness in selecting variables. As sketched out in Algorithm 2, the set of candidate variables is limited to those that appear in a randomly selected clause $c$. Note that in Line 3, WeightedNovelty uses the weighted score $\mathrm{wscore}(v)$ to rank all variables $v$ in clause $c$ instead of the normal $\mathrm{score}(v)$ score

[2]A promising variable has its $\mathrm{wscore} > 0$.

**Algorithm 2:** WeightedNovelty($p$)

```
1  c = randomly select an unsatisfied clause;
2  for all variables in clause c do
3      find the best and second best variables based on wscore, breaking ties in
       favour of the least recently flipped variable;
4  end
5  if within probability p then
6      return the second best variable;
7  else return the best variable;
8  ;
```

as the original Novelty does. With probability $p$, WeightedNovelty will select the second best variable within clause $c$ rather than returning the best one. Note that gNovelty$^+$ only adjusts the clause weights after WeightedNovelty completes its cycle. Once promising variables become available, gNovelty$^+$ will switch back to its intensification mode.

## 3 Weight-Enhanced Diversification

In most SLS solvers, clause weights are only used for computing wscore and ranking variables. To the best of our knowledge, no SLS solver has used clause weights for selecting clauses in the diversification mode. In fact, their diversification method (which is usually either Random Walk or Novelty Walk) normally selects an unsatisfied clause randomly. Nevertheless, a clause weight records no-good information about how often that clause has been false at local minima. Similarly, a variable weight stores no-good information about how often a variable has been flipped. In its diversification phase, VW1 uses variable weights to break ties beside its scoring function [Prestwich, 2005]. The tie-breaking prefers the variable having the least weight (i.e. the least often flipped variable). In contrast, VW2 uses variable weights to adjust its dynamic scoring function rather than to break ties. This study is motivated by the desire to investigate the use of weighting-based enhancement in the diversification phase. Firstly, we use clause weights to select clauses in the trap escaping phase. Secondly, we use variable weights to break ties in the scoring function. Below we introduce our weight-enhanced heuristics to improve the diversification of SLS solvers, especially gNovelty$^+$.

### 3.1 Clause-Weighting Enhancement

In the diversification phase, our new heuristic greedily selects an unsatisfied clause based on the clause weights. With probability $\beta$, it selects an unsatisfied clauses randomly. Otherwise, with probability (1-$\beta$), it selects an unsatisfied clause with the maximum clause weight. The aim of selecting a clause with the highest weight is to make an attempt to satisfy the clause that is most often falsified at a local minimum. Satisfying such a clause may help the search escape from the current trap. The positive aspect of using the diversification noise $\beta$ is the flexibility in switching back and forth between greediness and randomness. This allows the solver to occasionally move away from being too greedy.

The new NoveltyGC is outlined in Algorithm 3. Notice that Line 1 in Algorithm 2 has been replaced by Lines 1-4 in Algorithm 3. Once the false clause $c$ is selected, the algorithm will compute the $\mathrm{wscore}(v)$ to find the best and second

**Algorithm 3:** NoveltyGC($\beta, p$)

```
1  if within probability β then
2      c = randomly select an unsatisfied clause;
3  else c = select an unsatisfied clause with the highest clause weight Wgh(c);
4  ;
5  for all variables in clause c do
6      find the best and second best variables based on wscore, breaking ties
       based on a diversification criterion;
7  end
8  if within probability p then
9      return the second best variable;
10 else return the best variable;
11 ;
```

best variables w.r.t. the score. If two variables have the same score, normally ties are broken by selecting the least recently flipped variable (Line 3 in Algorithm 2). In the next section, we will describe other diversification criteria to break such ties (Line 6 in Algorithm 3).

### 3.2 Variable-Weighting Enhancement

As mentioned above, Novelty uses variable age as its diversification criterion. A variable age is computed as the number of flips since that variable was last flipped [McAllester *et al.*, 1997]. In other words, the older the variable, the more likely it will be selected. Many SLS solvers (e.g. gNovelty$^+$, PAWS, CCASat, and Sparrow) use variable ages to break ties in their diversification phase.

In this study, we use variable weights as an alternative diversification criterion. In this case, ties are broken by selecting the variable $v$ with less weight $\mathrm{VarWgh}(v)$, which records the number of times $v$ has been flipped. By doing so, the algorithm prefers to direct the search to the area that involves the least frequently flipped variables. The use of this kind of variable weights has reportedly improved diversification more than variable ages do [Prestwich, 2005].

Finally, we also investigate the use of both variable weights and variable ages in a combined fashion: break ties by selecting the variable with smaller variable weight, then break further ties by preferring the older variable. It was reported that a selective combination of variable properties (e.g. variable age, different types of scores) can boost the performance of local search solvers that originally use only a single variable property [Tompkins and Hoos, 2010]. However, the drawback of these solvers is that they require appropriate configuration of many parameters.

### 3.3 Implementation

In order to study the effect of our new weight-enhanced heuristics, we integrated these new changes into gNovelty$^+$. The new algorithm gNovelty$^+$GC (where GC stands for 'greedy clause selection') is illustrated in Algorithm 4. The obvious difference is gNovelty$^+$GC now uses NoveltyGC (Line 12) instead of the old WeightedNovelty heuristic (as in Line 12 in Algorithm 1). In addition, we altered gNovelty$^+$GC in Line 10 so that it will break ties in the intensification phase using the same diversification criterion as NoveltyGCdoes in the diversification phase.

In summary, there are three different diversification criteria: variable age, variable weight, and their combination. In

**Algorithm 4:** gNovelty$^+$GC($\Theta$, $\beta$, $sp$)

```
Input  : Formula $\Theta$, diversification probability $\beta$, smoothing probability $sp$
Output: Solution $\alpha$ (if found) or TIMEOUT
randomly generate a candidate solution $\alpha$;
initiate all clause weights $\mathrm{Wgh}(c) = 1$;
while not timetout do
    if $\alpha$ satisfies the formula $\Theta$ then return $\alpha$ ;
    ;
    if within the random walk probability $wp = 0.01$ then
        perform a Random Walk;
    else
        if there exits promising variables then
            select a variable $v$ with the highest $\mathrm{wscore}(v)$, breaking ties
            based on a diversification criterion;
        else
            perform NoveltyGC($\beta, p$) and adjust its noise $p$;
            update and smooth (with the probability $sp$) clause weights;
        end
    end
    update the candidate solution $\alpha$;
end
return TIMEOUT;
```

order to distinguish the three variants of gNovelty$^+$GC, we name them respectively as gNovelty$^+$GC$_a$, gNovelty$^+$GC$_w$, and gNovelty$^+$GC$_{wa}$ based on the diversification criterion being used. However, for space constraints and convenience, we may also denote them by GC$_a$, GC$_w$, and GC$_{wa}$ in the result tables or figures.

## 4 Experimental Results

We first evaluated the gNovelty$^+$GC variants on ternary chain instances to study their diversification capability over other known algorithms. Afterwards, they were tested on verification benchmarks from real world problems. Finally, to compare with the state of the art results, experiments were run with the SAT 2011 Competition benchmarks and the solvers TNM, VW2, EagleUP [Gableske and Heule, 2011], sattime2011 [Li and Li, 2012], Sparrow and CCASat. Note that Sparrow and CCASat are respectively the winners of SAT competitions in 2011 and 2012 on the random track. Experiments were conducted on Griffith University Gowonda HPC Cluster equipped with a number of Intel(R) Xeon(R) CPUs X5650 @2.67GHz.

### 4.1 Experiments on Ternary Chains

Ternary chain is an artificial problem that deliberately simulates chains of variable dependencies in structured problems. It is employed to study the diversification capability of local search algorithms in [Prestwich, 2002; Wei and Selman, 2002]. A ternary chain has only one solution where all variables are true. Therefore, flipping any variable from true to false is likely to move away from the solution. For this characteristic, local search algorithms will easily become trapped. The formulation of a ternary chain with N variables is defined as follows:

$$(x_1)\,(x_2)\,(x_1 \wedge x_2 \rightarrow x_3)\ \ldots\ (x_{N-2} \wedge x_{N-1} \rightarrow x_N)$$

In this experiment, we compare gNovelty$^+$GC($\beta = 0.5$) against gNovelty$^+$and other popular solvers such as PAWS, VW2, Sparrow and CCASat. The results are depicted in Figure 1. Figure 1(a) shows the success rate of solvers on ternary chains of size [10,100]. Figures 1(b) and 1(c) present the run times (in seconds) taken to solve ternary chains of size [10,100] and [100,1000], respectively. Each solver was ran 100 times on each instance with a timeout of 20 seconds.

As seen in Figure 1, it was apparent that VW2, gNovelty$^+$ and gNovelty$^+$GC performed significantly better than other solvers. gNovelty$^+$ and VW2 solved all ternary chains successfully due to variable weighting and clause weighting. It is very clear from Figure 1 that gNovelty$^+$GC$_a$ is actually no different with gNovelty$^+$. However, gNovelty$^+$GC$_w$, which takes advantage of gNovelty$^+$ and VW2, performed better than these two algorithms. Notice that gNovelty$^+$GC$_{wa}$, being a compromise between gNovelty$^+$GC$_a$ and gNovelty$^+$GC$_w$, was slightly better than gNovelty$^+$GC$_a$, but worse than gNovelty$^+$GC$_w$.

### 4.2 Experiments on Verification Problems

The experiments were conducted on three real world verification problems: cbmc, swv and sss-sat-1.0. The first two instance sets are software verification problems: (i) 39 cbmc instances generated by a bounded model checking tool and (ii) 75 swv instances generated by the CALYSTO checker.[3] The third one is Velev's sss-sat-1.0 which contains 100 instances encoding verification of super-scalar microprocessors.[4] Even though these instance sets can be easily solved by the complete solver PICOSAT [Biere, 2008], they are currently a remarkable challenge for SLS solvers [Tompkins *et al.*, 2011]. The experiments on these datasets were conducted by running the solvers 50 times on each instance with a timeout of 600s.

In Figure 2, we compared gNovelty$^+$GC$_a$ with the original gNovelty$^+$ , sattime2011 (being the best SLS solver and ranked 4$^{th}$ overall on crafted instances in the SAT 2011 competition), and Sparrow (being the best solver for random instances in the SAT 2011 competition). As we can see from this figure, gNovelty$^+$GC$_a$ performed significantly better than all these three solvers.

The full empirical results are detailed in Table 1 in terms of three criteria: success rates, run times (in seconds), and flip counts (in millions). Data for VW2, Sparrow and CCASat were partially omitted as their flip counters were overflowed. The run times and flip counts for each problem set are the averages of the median data on each instance in that set. In general, our results show that the three gNovelty$^+$GC variants outperformed other solvers in all three criteria. Among these variants, gNovelty$^+$GC$_w$ is the best for cbmc and swv. Especially, gNovelty$^+$GC$_w$ is five times faster than two other variants on cbmc instances in terms of CPU time. In contrast, gNovelty$^+$GC$_w$ was no match to the other two variants on the sss-sat-1.0 dataset. Indeed, it is the worst variant in all three criteria. Although gNovelty$^+$GC$_{wa}$ was not the best solver among three variants in any dataset, it performed very steadily. Overall, it had the best success rate on all three datasets (being 100%, 49% and 99% for cbmc, swv and sss-sat-1.0 respectively). Moreover, it achieved reasonable average run time compared to the other solvers.

[3]The test instances of cbmc and swc are available at http://people.cs.ubc.ca/davet/papers/sat10-dave-instances.zip

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

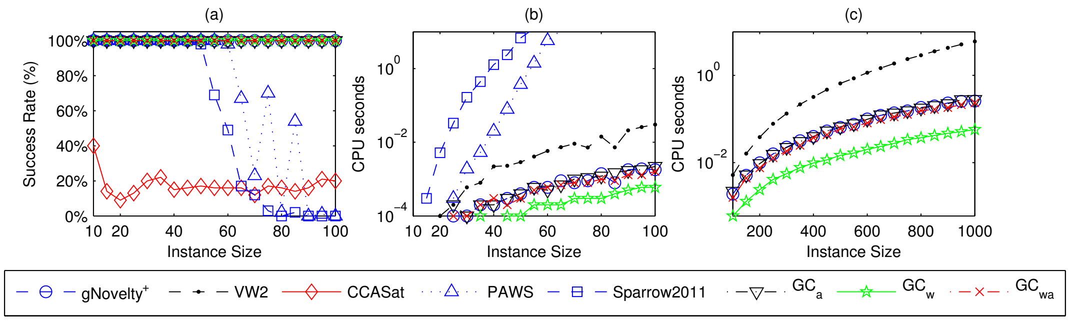

Figure 1: Experiments on ternary chains.

Table 1: Experimental results on cbmc, swv, and sss-sat-1.0.

| Instances | TNM | vw2 | Sparrow2011 | sattime2011 | EagleUP | CCASat | gnovelty$^+$ | GC$_a$ | GC$_w$ | GC$_{wa}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| cbmc (39) | 92% | 31% | 51% | 54% | 0% | 54% | 85% | 97% | **100%** | **100%** |
| | 76.599 | 439.128 | 384.359 | 322.528 | 600.000 | 276.196 | 247.997 | 25.779 | **3.523** | 19.381 |
| | 79.955 | - | - | 354.874 | 544.647 | - | 230.030 | 25.479 | **3.543** | 19.318 |
| swv (75) | 23% | 23% | 21% | 36% | 0% | 29% | 25% | **49%** | **49%** | **49%** |
| | 464.013 | 466.002 | 486.949 | 397.786 | 600.000 | 430.744 | 459.097 | 327.537 | **307.620** | 307.737 |
| | 440.235 | - | - | 137.054 | 153.571 | - | 214.360 | 81.675 | **59.223** | 59.531 |
| sss-sat-1.0 (100) | 18% | 24% | 7% | 20% | 1% | 68% | 50% | **99%** | 86% | **99%** |
| | 517.709 | 481.357 | 572.993 | 497.736 | 594.222 | 193.682 | 348.512 | **23.041** | 105.317 | 30.380 |
| | 274.499 | - | - | 337.899 | 177.572 | - | 164.495 | **15.458** | 91.975 | 19.144 |

Table 2: Experimental results on SAT 2011 competition *structured* benchmarks.

| Instances | TNM | vw2 | Sparrow2011 | sattime2011 | EagleUP | CCASat | gnovelty$^+$ | GC$_a$ | GC$_w$ | GC$_{wa}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Application (8) | 25% | 12% | 25% | 25% | 0% | 38% | 12% | 25% | **50%** | 25% |
| | 455.876 | 526.838 | 515.799 | 452.464 | 600.000 | 443.871 | 527.431 | 456.873 | **392.166** | 453.922 |
| | **173.463** | - | - | 224.578 | 252.589 | - | 320.968 | 186.102 | 205.426 | 190.997 |
| Crafted (82) | 68% | 37% | 56% | **72%** | 28% | 62% | 62% | 68% | 67% | **72%** |
| | 213.498 | 390.427 | 285.159 | **186.153** | 433.516 | 238.058 | 251.332 | 215.764 | 214.575 | 192.656 |
| | 142.967 | - | - | **93.128** | 273.928 | - | 163.151 | 109.012 | 114.896 | 102.072 |

Table 3: Experimental results on SAT 2011 competition *random* benchmarks.

| Instances | TNM | vw2 | Sparrow2011 | sattime2011 | EagleUP | CCASat | gnovelty$^+$ | GC$_a$ | GC$_w$ | GC$_{wa}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Random | **100%** | 19% | **100%** | **100%** | **100%** | 98% | 75% | **100%** | **100%** | **100%** |
| Medium (201) | **3.020** | 499.761 | 40.671 | 3.785 | 16.815 | 19.613 | 184.698 | 4.085 | 4.881 | 3.834 |
| | 1.793 | - | - | 1.887 | 9.849 | - | 227.457 | **1.659** | 1.848 | 1.862 |
| Random | 73% | 0% | 20% | **75%** | 36% | 70% | 0% | 39% | 25% | 23% |
| Large (64) | **248.099** | 600.010 | 509.487 | 275.933 | 398.783 | 193.343 | 600.000 | 395.203 | 481.060 | 496.909 |
| | **85.380** | - | - | 152.147 | 127.281 | - | 256.643 | 96.730 | 174.258 | 200.040 |

## 4.3 Experiments on SAT2011 Benchmarks

We compared the performance of gNovelty$^+$GC variants against other solvers on the SAT 2011 benchmarks. Each solver was ran 10 times on each instance with a timeout of 600s. The full results are reported in Tables 2 and 3 in terms of success rates, run times (in seconds), and flip counts (in millions).[5] Data for VW2, Sparrow and CCASat were partially omitted as their flip counters were overflowed. The run times and flip counts for each problem set are the averages of the median data on each instance in that set.

[5]Because many instances were too hard to solve, we reported only instances that were solved at least once by a tested solver.

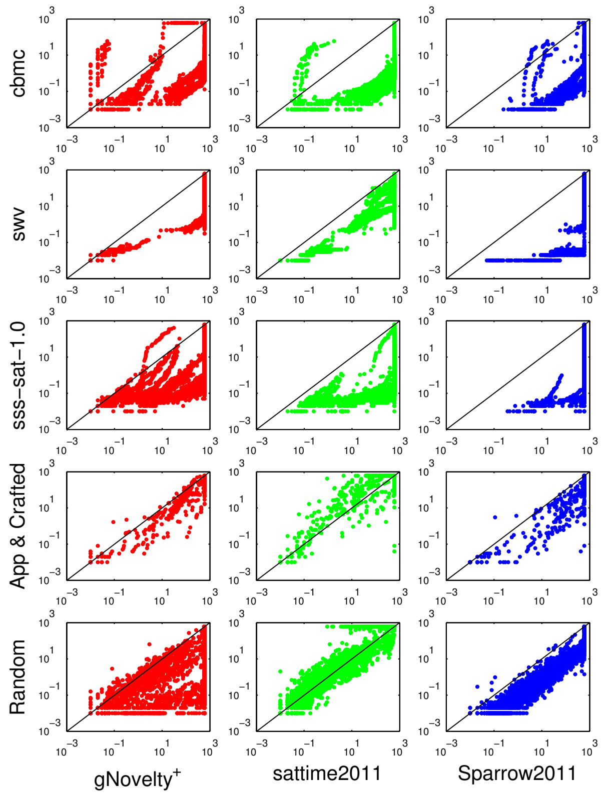

Figure 2: Comparison of gNovelty$^+$GC$_a$ (y-axis) against other solvers (x-axis) in terms of CPU times.

We also plotted the run times of gNovelty$^+$GC$_a$ against gNovelty$^+$, sattime2011 and Sparrow in Figure 2. As illustrated in that figure, gNovelty$^+$GC$_a$ was significantly better over gNovelty$^+$ on both structured and random instances. Moreover, it outperformed Sparrow in both datasets and achieved comparably equal performance with sattime2011.

Overall, Tables 2 and 3 show that all gNovelty$^+$GC variants outperformed gNovelty$^+$, Sparrow and VW2 in all problem sets. However, some variants performed worse than other solvers in terms of success rate on certain datasets. For example, on application instances, gNovelty$^+$GC$_a$ and gNovelty$^+$GC$_{wa}$ were on par with TNM and sattime2011 in terms of success rate (25%) and run times. However, they were not as good as CCASat (38%) in terms of success rate. gNovelty$^+$GC$_w$ with the success rate of 50% was the best solver on application instances.

On crafted instances, sattime2011 and gNovelty$^+$GC$_{wa}$ had the best success rate (72%), but in terms of CPU time, sattime2011 was better. Furthermore, the other two variants with the success rate of 68% and 67% were considerably better than many other solvers. However, the three best solvers were TNM, sattime2011 and CCASat, whose success rates were respectively 73%, 75%, and 73% on large random instances. The three gNovelty$^+$GC variants only achieved the success rates of 39%, 25% and 23% respectively. On medium random instances, they achieved steadily good results with 100% success rate. In terms of CPU time, all gNovelty$^+$GC variants were significantly better than the other solvers except TNM and sattime2011.

## 4.4 Discussions

Table 4 presents the parameter settings for gNovelty$^+$GC variants tuned by ParamILS, a local search optimisation tool for parameterised algorithms [Hutter *et al.*, 2009].

Table 4: gNovelty$^+$GC parameter settings.

| Solvers | GC$_a$ | | GC$_w$ | | GC$_{wa}$ | |
|---|---|---|---|---|---|---|
| Instances | $\beta$ | sp | $\beta$ | sp | $\beta$ | sp |
| cbmc | 0.15 | 0.0 | 0.05 | 0.0 | 0.05 | 0.0 |
| swv | 0.25 | 0.0 | 0.15 | 0.1 | 0 | 0.15 |
| sss-sat-1.0 | 0.3 | 0.0 | 0.05 | 0 | 0.3 | 0.0 |
| SAT2011 | 0.15 | 0.40 | 0.25 | 0.20 | 0.0 | 0.40 |

As seen from Table 4, the smooth probability $sp$ of structured instances are very low. For example, on cbmc and sss-sat-1.0, it is 0 for all variants, meaning that no clause weight was reduced. In contrast, it has a higher value on SAT 2011 dataset. This means the clause weight smoothing was needed occasionally. The majority of $\beta$ values is low ($\leq$ 30%), meaning the solvers were more greedy in picking false clauses. This demonstrates the essence of greedy clause selection.

In addition, our weight-based clause selection biases the search towards satisfying clauses that have been unsatisfied more often. This is confirmed when comparing the distribution of clause lengths of the original instance and that of the set of unsatisfied clauses being selected. Using our heuristic, the two distributions are different, but are very similar when selecting clauses randomly. We also found that the distribution of clause lengths over the set of selected clauses is significantly skewed towards unit clauses on SAT2011 application instances in comparison to such distribution on BMC instances. This strong bias gives a hint on why our heuristic is less efficient on application instances.

## 5 Conclusion And Future work

This paper presented a method to enhance diversification of SLS solvers by leveraging the advantage of clause weights and variable weights. The first contribution of this paper is a technique to balance between greediness and randomness in clause selection at local minima. The second contribution is the employment of variable weights as a diversification criterion in breaking ties. The results on real world structured instances and the SAT 2011 competition datasets showed that significant enhancement was achieved by greedily selecting clauses. Furthermore, our new heuristics which utilise both types of weights significantly boost the performance of gNovelty$^+$ . Another finding of this work is that the gNovelty$^+$GC$_{wa}$ version is a good compromise of gNovelty$^+$GC$_w$ and gNovelty$^+$GC$_a$ . Based on the analysis of parameter settings, it is recommended that the smoothing probability on gNovelty$^+$GC should have a small value for structured instances but a high value for random instances.

In the future, we plan to investigate adaptation of the diversification factor $\beta$ as the search progresses. We also intend to apply NoveltyGC on a wide range of SLS solvers.

## References

[Balint and Fröhlich, 2010] Adrian Balint and Andreas Fröhlich. Improving stochastic local search for SAT with a new probability distribution. In *SAT*, pages 10–15, 2010.

[Biere, 2008] Armin Biere. PicoSAT essentials. *JSAT*, 4(2-4):75–97, 2008.

[Cai and Su, 2012] Shaowei Cai and Kaile Su. Configuration checking with aspiration in local search for SAT. In *AAAI*, 2012.

[Duong *et al.*, 2012] Thach-Thao Duong, Duc Nghia Pham, and Abdul Sattar. A study of local minimum avoidance heuristics for SAT. In *ECAI*, pages 300–305, 2012.

[Duong *et al.*, 2012] Thach-Thao Duong, Duc Nghia Pham, and Abdul Sattar. A Method to Avoid Duplicative Flipping in Local Search for SAT. In *AI*, pages 218-229, 2012.

[Gableske and Heule, 2011] Oliver Gableske and Marijn Heule. EagleUP: Solving random 3-SAT using SLS with unit propagation. In *SAT*, pages 367–368, 2011.

[Hutter *et al.*, 2009] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)*, 36:267–306, 2009.

[Li and Huang, 2005] Chu Min Li and Wen Qi Huang. Diversification and determinism in local search for satisfiability. In *In Proceedings of SAT-2005*, pages 158–172, 2005.

[Li and Li, 2012] Chu Min Li and Yu Li. Satisfying versus falsifying in local search for satisfiability - (poster presentation). In *SAT*, pages 477–478, 2012.

[McAllester *et al.*, 1997] David A. McAllester, Bart Selman, and Henry A. Kautz. Evidence for invariants in local search. In *AAAI/IAAI*, pages 321–326, 1997.

[Pham *et al.*, 2008] Duc Nghia Pham, John Thornton, Charles Gretton, and Abdul Sattar. Combining adaptive and dynamic local search for satisfiability. *JSAT*, 4(2-4):149–172, 2008.

[Pham *et al.*, 2012] Duc Nghia Pham, Thach-Thao Duong, and Abdul Sattar. Trap avoidance in local search using pseudo-conflict learning. In *AAAI*, pages 542–548, 2012.

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

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

[Selman *et al.*, 1992] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *AAAI*, pages 440–446, 1992.

[Thornton *et al.*, 2004] John R. Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proceedings of AAAI-04*, pages 191–196, 2004.

[Tompkins and Hoos, 2010] Dave A. D. Tompkins and Holger H. Hoos. Dynamic scoring functions with variable expressions: New SLS methods for solving SAT. In *SAT*, pages 278–292, 2010.

[Tompkins *et al.*, 2011] Dave A. D. Tompkins, Adrian Balint, and Holger H. Hoos. Captain Jack: New variable selection heuristics in local search for SAT. In *Proceedings of SAT-2011*, pages 302–316, 2011.

[Wei and Li, 2009] Wanxia Wei and Chu Min Li. Switching between two adaptive noise mechanisms in localsearch. In *Booklet of the 2009 SAT Competition*, 2009.

[Wei and Selman, 2002] Wei Wei and Bart Selman. Accelerating random walks. In *Proceedings of CP-02*, pages 216–232, 2002.

[Wei *et al.*, 2008] Wanxia Wei, Chu Min Li, and Harry Zhang. A switching criterion for intensification and diversification in local search for SAT. *JSAT*, 4(2-4):219–237, 2008.