

Combining Adaptive and Dynamic Local Search for Satisfiability

Duc Nghia Pham

duc-nghia.pham@nicta.com.au

John Thornton

john.thornton@nicta.com.au

Charles Gretton

charles.gretton@nicta.com.au

Abdul Sattar

abdul.sattar@nicta.com.au

SAFE Program, Queensland Research Lab, NICTA Ltd., Australia

and

IIS, Griffith University, QLD, Australia

Abstract

In this paper we describe a stochastic local search (SLS) procedure for finding satisfying models of satisfiable propositional formulae. This new algorithm, gNovelty⁺, draws on the features of two other WalkSAT family algorithms: AdaptNovelty⁺ and G²WSAT, 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: AdaptG²WSAT0, G²WSAT 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 ; published

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

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 competition¹. 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⁺ [?] and G²WSAT [?]. 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 [?]) and multiplicative weighting (e.g. (R)SAPS [?]).

In the remainder of the paper we describe in more detail the techniques used in G²WSAT, 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 [?] 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) 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

1. <http://www.satcompetition.org>

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

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 [?], due to its deterministic variable selection³ it may loop indefinitely and fail to return a solution even where one exists [?, ?]. We refer the reader to [?] for an example instance that is satisfiable but for which Novelty unable to find a solution regardless of the noise parameter setting. Hoos [?] 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 clause c with a *walk* probability wp and behaves exactly as Novelty in other cases.

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 [?, ?]. This means that without extensive empirical tuning, the average case performance of a WalkSAT algorithm is quite poor. Hoos [?] 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 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 [?] demonstrated experimentally that this adaptive noise mechanism is effective both with Novelty⁺ and the other WalkSAT variants.

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

2.2 G²WSAT

More recently Li and Huang [?] proposed a new heuristic to solve the problem of determinism in Novelty (discussed in the previous section). As opposed to using a Novelty⁺-type random walk [?], 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 random clause c for the next move with a *diversification* probability dp , otherwise it performs as Novelty. Li and Huang [?] further improved Novelty⁺⁺ by combining the greedy heuristic in GSAT [?] with a variant of tabu search [?] 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 G²WSAT 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, G²WSAT selects the least recently flipped one, and if the search hits a local minimum, G²WSAT disregards the tabu list and performs as Novelty⁺⁺ until it escapes.

Algorithm 2 G²WSAT(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 dynam-

ically 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 [?], 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 such promising variable exists, it randomly selects a variable for the next move with a 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. In addition, clause weights are probabilistically smoothed and reduced to the average clause weight by a factor ρ . 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 [?] attempted to dynamically adjust the value of the smooth probability sp using the same approach as AdaptNovelty⁺ [?], while holding the other three parameters (wp , α and ρ) fixed. Their experimental study showed that the new RSAPS solver can achieve similar and sometimes better results in comparison to SAPS [?]. However, the other parameters in RSAPS, especially ρ , still need to be manually tuned in order to achieve optimal performance [?, ?].

2.4 PAWS

More recently, Thornton *et al.* [?] 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 [?, ?] demonstrated the overall superiority of PAWS over SAPS for solving large and difficult problems.

Algorithm 4 PAWS⁺(F, $fp=0.15$, w_{inc})

```

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  $x$ , 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 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’;

```

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:

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