Adaptive Novelty$^+$

Holger H. Hoos and Dave A. D. Tompkins
University of British Columbia
Computer Science Department
{hoos,davet}@cs.ubc.ca

1 Preface

This paper is essentially a reprint of the solver description from the 2004 competition, as the software submitted this year is identical to the 2004 software that placed first in the RANDOM category of the competition.

2 Introduction

In this paper we briefly describe Novelty$^+$ and Adaptive Novelty$^+$, two high-performance, stochastic local search (SLS) algorithms for SAT. Based on the WalkSAT architecture, these algorithms combine search intensification and diversification feautures that lead to good peformance on a broad range of SAT instances. The performance of the Novelty$^+$ algorithm critically depends on the setting of a so-called noise parameter, which effectively controls the relative amount of search diversification. Adaptive Novelty$^+$ exploits insights in this performance dependence to dynamically adjust the noise parameter throughout the search, depending on search progress and stagnation.

3 Novelty

Algorithms from the WalkSAT family are amongst the most widely known and best-performing SLS algorithms for SAT [5, 4, 3]. Starting from a randomly chosen variable assignment, they repeatedly select one of the clauses which are violated by the current assignment. Then, according to some heuristic a variable occurring in this clause is flipped using a greedy bias to increase the total number of satisfied clauses. (See Figure 1.)

For the original WalkSAT algorithm [5], also known as *WalkSAT/SKC*, the following heuristic is applied. If any variables in the selected clause are *safe*, meaning they can be flipped without violating any clauses, then a safe variable is randomly chosen. Otherwise, with a fixed probability *p* a variable is randomly chosen from the clause and with probability *1-p* a variable is selected which minimises the number of clauses which are currently satisfied but would become violated by the variable's flip (number of breaks).

Novelty [4] considers the variables in the selected clause sorted according to their score, *i.e.*, the difference in the total number of satisfied clauses a flip would cause. If the best variable according to this ordering (*i.e.*, the one with maximal score) is not the most recently flipped one, it is flipped, otherwise, it is flipped with a probability *1-p*, while in the remaining cases, the second-best variable is flipped.

4 Novelty$^+$

Although Novelty in many cases performs substantially better than WalkSAT/SKC and other WalkSAT algorithms, it has been proven to suffer from *essential incompleteness*, *i.e.*, there are situations in which without restarting the search, even for arbitrarily long runs, the probability of finding an existing solution to a given satisfiable CNF formula approaches a limit strictly less than one. In other words, the search process underlying Novelty can get terminally trapped in non-solution regions of the given search space. In practice, this has been shown to occasionally lead to extreme stagnation behaviour, which affects the performance of the algorithm in a very detrimental way [1].

Novelty$^+$ has been designed to overcome both the theoretical weakness and the practically observed stagnation behaviour [1]. It can be seen as a simple extension of Novelty with an unconditional random walk mechanism, similar to the earlier GSAT with Random Walk algorithm [5]. In Novelty$^+$, in each search step, with a user-specified probability *wp*, the variable to be flipped is randomly selected from the selected clause, while in the remaining cases, the variable is selected according to the heuristic for Novelty. Novelty$^+$ is *probabilistically approximately complete*, *i.e.*, by running the search process sufficiently long (without

```
procedure WalkSAT(F, maxTries, maxSteps, Select)
  for try := 1 to maxTries do
    a := randomly chosen assignment of the variables in F;
    for step := 1 to maxSteps do
      if a satisfies F then return a;
      c := randomly selected clause which is unsatisfied under a;
      v := variable from a selected according to a heuristic Select;
      a := a with v flipped;
    end for;
  end for;
  return "no solution found";
end WalkSAT;
```

Figure 1: The WalkSAT algorithm family.

using restart), arbitrarily high probabilities of finding an existing solution can be guaranteed. In practice, using $wp = 0.01$ is sufficient for avoiding the severe stagnation behaviour occasionally observed for Novelty. More precisely, for sufficiently high values of p, the performance of Novelty$^+$ is basically unaffected by restarts, and using the parameters $maxSteps = \infty$ and $maxTries = 1$ always results in optimal performance.

5 Adaptive Novelty$^+$

As in the case of WalkSAT/SKC and Novelty, the noise parameter, p, which controls the degree of randomness of the search process, has a major impact on the performance and run-time behaviour of Novelty$^+$. Unfortunately, the optimal value of p varies significantly between problem instances, and even small deviations from the optimal value can lead to substantially decreased performance [2].

Adaptive Novelty$^+$ dynamically adjusts the noise setting p based on search progress, as reflected in the time elapsed since the last improvement in the number of satisfied clauses has been achieved.

At the beginning of the search, the search is maximally greedy ($p = 0$). This will typically lead to a series of rapid improvements in the evaluation function value, followed by stagnation (unless a solution to the given problem instance is found). In this situation, the noise value is increased. If the resulting increase in the diversification of the search process is not sufficient to escape from the stagnation situation, that is, if it does not lead to an improvement in the number of satisfied clauses within a certain number of steps, the noise value is further increased. Eventually, p should be high enough for the search process to overcome the stagnation situation, at which point the noise can be gradually decreased, leading to an increase in search intensification, until the next stagnation situation is detected or a solution to the given problem instance is found.

Details of the mechanism used in Adaptive Novelty$^+$ can be found in [2]; as shown there, Adaptive Novelty$^+$ typically achieves the same performance as Novelty$^+$ with approximately optimal static noise, which renders it one of the best-performing and most robust SLS algorithms for SAT currently available.

6 Implementations in UBCSAT

For the SAT competition Adaptive Novelty$^+$ was implemented in the UBCSAT framework [7], precisely following the earlier reference implementations used in [1, 2]. However, different from those, the UBCSAT implementations do not use the caching and incremental updating scheme for variable scores originally developed for GSAT [6], but rather computes effects of variable flips for each variable occurring in a selected clause from scratch, which interestingly improves the efficiency of the implementation for many types of SAT instances.

References

[1] H. H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proc. of the Sixteenth Nat'l Conf. on Artificial Intelligence (AAAI-99)*, pages 661–666, Orlando, Florida, 1999.

[2] H. H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proc. of the 18th Nat'l Conf. in Artificial Intelligence (AAAI-02)*, pages 655–660, 2002.

[3] H. H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[4] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proc. of the Fourteenth Nat'l Conf. on Artificial Intelligence (AAAI-97)*, pages 321–326, 1997.

[5] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. of the 12th Nat'l Conf. on Artificial Intelligence (AAAI-94)*, pages 337–343, 1994.

[6] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the Tenth Nat'l Conf. on Artificial Intelligence (AAAI-92)*, pages 459–465, 1992.

[7] D. A. D. Tompkins and H. H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *LNCS 3542: Proceedings of the Seventh Int'l Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, pages 305–319, 2004.