

# Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT

## CPSC 532D Course Project Report

Frank Hutter  
University of British Columbia  
Department of Computer Science  
mail@fhutter.de

Dave Tompkins  
University of British Columbia  
Department of Electrical Engineering  
davet@ece.ubc.ca

April 13, 2002

### Abstract

In this paper, we study the approach of Dynamic Local Search for the SAT problem. We focus on the recent and promising Exponentiated Sub-Gradient (ESG) algorithm, and examine its' dependence on specific parameter settings. We propose a new variant on this algorithm that we call Scaling and Probabilistic Smoothing (SAPS) which significantly improves upon the time performance of the regular ESG algorithm. We also provide a reactive version of SAPS (RSAPS) that can adaptively tune one of the algorithm parameters. We conclude with some insights into how these ESG-based algorithms are achieving their exceptional performance.

## 1 Introduction and Background

The Satisfiability problem (SAT) is an important subject of study in many areas of computer science. Since SAT is NP complete, there is little hope to develop a complete algorithm that scales well on every problem instance; however, fast algorithms are needed to solve big problems from various domains, including prominent AI problems like planning [26] and Constraint Satisfaction [1]. Throughout this paper, we deal with the model finding variant of SAT, i.e. given a propositional formula, find a model of it.<sup>1</sup> Some of the best known methods for model finding problems are Stochastic Local Search (SLS) algorithms, which are inherently incomplete, but succeed in practice due to their superior performance [3, 4, 5].

Although SLS algorithms for SAT differ in their implementation details, the general search strategy is mostly the same [2]. At every stage of the search, each variable is assigned a

---

<sup>1</sup>More specifically, we deal with SAT problems in conjunctive normal form (CNF). Propositional formulas in CNF are conjunctions of disjunctions of literals. A literal is a propositional variable or its' negation. Generally, the disjunctions of literals are referred to as *clauses*.

truth value, which comprises a candidate solution. Between two stages of the search, the truth assignment of one variable is reversed (the variable is *flipped*), which is known as a search step. Because one variable is flipped at each step, the number of steps is often referred to as the number of flips. A model of the formula has to satisfy all the clauses, so the objective function to be minimised measures the number of unsatisfied clauses.

Since the first usage of local search techniques in SAT [3], much research has been conducted in this area. Major speed improvements were made by the usage of noise [4] and the development of the WalkSAT (or short WSAT) scheme [5]. Further analysis lead to more insights into the actual role of noise [6, 7, 8], and to the development of slightly more complicated WalkSAT variants such as Novelty and R-Novelty [6]. Additional insight into the nature of SLS algorithms lead to the development of Novelty<sup>+</sup> [24]. Finally, the heavy dependence of GSAT- and WSAT-like algorithms on their noise parameter setting was resolved by approaches which automatically adapt the noise. Auto-WalkSAT [7] achieves a roughly optimal fixed noise parameter setting by the exploitation of noise invariants and short initial runs of the algorithm which solely determine a good noise setting. A recent adaptive variant of Novelty<sup>+</sup> [8], which we will refer to as Adaptive Novelty<sup>+</sup>, increases the noise parameter when search stagnation is sensed and decreases it accordingly when an improvement in the objective function value has taken place.

In parallel to the development of more refined versions of iterative improvement strategies like GSAT and WalkSAT, another SLS method has become increasingly popular in SAT solving. This method is known as dynamic local search (DLS) [2]. The key idea of DLS is that although the objective function for a given problem instance will always stay the same (the number of unsatisfied clauses), the evaluation function which guides the local search can be adapted. A common approach in DLS strategies is to use a clause weighting scheme to adjust the evaluation function. This was first suggested in [4], although this approach is not very dynamic, as the weights are only changed between subsequent tries. Many variants of this scheme have been proposed: the breakout method [23] simply adds one to the weight of every unsatisfied clause whenever a local minimum is encountered. Cha and Iwama [15] use a unique approach of weighting clauses by actually adding new clauses, which may not fit the strictest definition of a DLS. Frank [14, 13] uses a DLS weighting scheme that is updated every time a variable is flipped. The Discrete Lagrangian Method [16] only updates the weights when a local minimum is encountered, as does the SDF approach [17]. The latter one introduced the concept of non-linear (exponential) weights. A very recent algorithm, on which we will base our work, is known as the exponentiated sub-gradient algorithm (ESG) [12], which follows up on SDF.

The ESG algorithm outperforms Novelty<sup>+</sup> when comparing numbers of steps needed to find a solution. However, due to its higher complexity, its' time performance is often inferior to Novelty<sup>+</sup>. In this paper, we study the ESG algorithm in more detail and develop an improved algorithm that does not suffer from poor time performance and thus outperforms Novelty<sup>+</sup> in both step and time performance.

The remainder of the paper is organised as follows. After stating our conventions for empirical evaluation, we investigate the characteristics of the ESG algorithm in Section 2. We study the algorithms' reliance on good parameter settings and give first intuitions on

why it is performing as well as observed. Section 3 introduces a new algorithm which resolves the main disadvantage of ESG, namely its' bad time performance. In Section 4, we give a parameter-less version of this algorithm, in which one of the key parameters is automatically adapted. More intuitions as to why the algorithm is working are given in Section 5. Section 6 concludes the paper and shows possibilities for further work.

## 1.1 Conventions for empirical evaluation

Due to the stochastic nature of SLS algorithms, the following sections make heavy usage of empirical analysis of algorithms. The benchmark problems on which our analysis is based are taken from the online Satisfiability library SATLIB <sup>2</sup>. We focused our analysis on uniform random 3-SAT (uf150 and uf200), but also present data for flat graph coloring (flat100), and the planning domains blocksworld and logistics. Empirical run-time and run-length distributions as well as each data point in correlation plots are based on 1000 tries unless otherwise stated. Sensitivity plots on a single parameter are also based on 1000 tries for each data point, whereas 2D sensitivity surface plots are based on 400 runs for each point unless otherwise stated. In many instances throughout this paper, especially when we are dealing with the problem sets uf150 and flat100, we label an instance as *hard* or *easy*. Even though all of the instances are taken from the phase transition region, there is a large variance in the difficulty of the instances. We measure hardness as the median time it takes Novelty<sup>+</sup> to solve the instance. We randomly selected problems from the top 5% and the bottom 5% of the hardness distribution for our experiments, and tested more than one instance to ensure that the behaviour was consistent with similarly difficult problems.

## 2 ESG Implementation

In this section, we will describe our efforts to analyse the ESG algorithm and understand its' properties. In Section 2.1 we describe the original ESG algorithm. In Section 2.2 we look at individual characteristics of the algorithm, and in Section 2.3 we look at how some of those characteristics interact.

### 2.1 Algorithm Description

The ESG algorithm was designed for a wide variety of constraint satisfaction problems. If a SAT-based ESG algorithm were to be implemented directly as described in [12], it would not compete with state of the art SLS SAT algorithms. Indeed, the implementation used in those experiments was tuned for SAT performance. The main description of the original SAT-based ESG algorithm can be found in Figure 1, with some of the functions described in Figures 2 and 3.

---

<sup>2</sup><http://www.satlib.org/> [25]

```

procedure ESG( $\pi, \alpha, \rho, wp$ )
  input:
    problem instance  $\pi$ , scaling factor  $\alpha$ , smoothing factor  $\rho$ , random walk probability  $wp$ 
  output:
    solution  $s$  or  $\emptyset$ 
   $s := \text{Init}(\pi)$ 
   $W := \text{InitWeights}(\pi)$ 
  while not terminate( $\pi, s$ ) do
     $s' := \text{WeightedSearch}(\pi, s, W)$ 
    if  $s' = \emptyset$  then
      With probability  $wp$  do
         $s := \text{RandomStep}(\pi, s)$ 
      otherwise do
         $W := \text{UpdateWeights}(\pi, s, W, \alpha, \rho)$ 
      end
    else
       $s := s'$ 
    end
  end
  if ( $s$  is not a solution) then  $s = \emptyset$ 
end

```

Figure 1: The Original ESG Algorithm. Init randomly initialises  $s$ , InitWeights initialises the clause weights to 1; WeightedSearch and UpdateWeights are to be defined later.

From Figure 1, it can be seen that the clause weights are only updated when a local minimum is reached, and when a random walk step does not occur. It is possible for the *WeightedSearch* function to return a null ( $\emptyset$ ), meaning that a local minimum has been reached and the search has failed. The *WeightedSearch* function is shown in Figure 2. In this function, the evaluation function  $g(s)$  is the sum of the unsatisfied clause weights in candidate solution  $s$ .

The *WeightedSearch* function reveals some of the interesting behaviour in the ESG algorithm. It can be seen that only true improvement steps are allowed, and sideways steps are not explored. Another interesting feature of the *WeightedSearch* function is the method in which variables are selected for consideration. In GSAT, all variables are considered as flip candidates. In WalkSAT, an unsatisfied clause is selected at random, and then the variables appearing in that clause are considered. The method used here, named UGSAT by Frank in [14] lies somewhere between GSAT and WalkSAT. It has strong intensification like GSAT, but has the added WalkSAT feature that at least one new clause becomes satisfied with each flip. It should also be pointed out that the selection mechanism is a best selection method, where ties are broken randomly.

Figure 3 illustrates the *UpdateWeights* function. The use of the scaling factor  $\alpha$  in this function is slightly different than the one described in [12]. While  $\alpha$  is a scaling factor in both implementations, our description is more straightforward. It is our intent to avoid confusion by using a similar, but simpler scaling notation.

```

procedure WeightedSearch( $\pi, s, W$ )
  input:
    problem instance  $\pi$ , candidate solution  $s$ , clause weights  $W$ 
  output:
    candidate solution  $\hat{s}$ 
   $U_v = \{\text{variables of } \pi \text{ that appear in an unsatisfied clause in } s\}$ 
   $S' := \{\hat{s} \mid \hat{s} \text{ is } s \text{ with variable } v \in U_v \text{ flipped}\}$ 
   $best := \min\{g(\pi, \hat{s}, W) \mid \hat{s} \in S'\}$ 
   $S := \{\hat{s} \in S \mid g(\pi, \hat{s}, W) = best\}$ 
  if  $best \geq 0$  then
     $\hat{s} := \emptyset$ 
  else
     $\hat{s} := \text{draw}(S)$ 
  end
end

```

Figure 2: The Original ESG Weighted Search.

```

procedure UpdateWeights( $\pi, s, W, \alpha, \rho$ )
  input:
    problem instance  $\pi$ , candidate solution  $s$ , clause weights  $W$ , scaling factor  $\alpha$ , smoothing factor  $\rho$ 
  output:
    clause weights  $W$ 
   $C = \{\text{clauses of } \pi\}$ 
   $U_c = \{c \in C \mid c \text{ is unsatisfied in } s\}$ 
  for each  $i$  s.t.  $c_i \in U_c$  do
     $w_i := w_i \times \alpha$ 
  end
  for each  $i$  s.t.  $c_i \in C$  do
     $w_i := w_i \times \rho + (1 - \rho) \times \bar{w}$ 
  end

```

Figure 3: The Original ESG Update Weights;  $\bar{w}$  denotes the average over all clause weights  $w_i$ .

## 2.2 ESG Algorithm Characteristics

In this section, we will describe our efforts to analyse certain characteristics of the ESG algorithm. We identified seven interesting features that seem to control the algorithm performance:

- A UGSAT selection mechanism
- Update clause weights every local minimum
- No sideways steps
- Best search, breaking ties randomly

- Parameters:
  - random walk ( $w_p$ )
  - scaling ( $\alpha$ )
  - smoothing ( $\rho$ )

We will explore each of these features individually over the following sections.

### 2.2.1 UGSAT Selection Mechanism

The UGSAT selection mechanism has not been very popular amongst the recent SLS algorithms. WalkSAT has demonstrated that by only examining a small number of variables, significant speedups can be achieved over GSAT, while the intensification is safely reduced. It seemed apparent that the same logic could be applied to this ESG algorithm, but we found that it was not the case. While some WalkSAT ESG variants could outperform Novelty<sup>+</sup>, the UGSAT-based selection mechanism outperformed the WalkSAT variants in all of our experiments.

We believe that there are three reasons why those variants were not as successful. The first reason stems from the reduced intensification. Regular UGSAT does not perform very well because the intensification is too strong, causing the algorithm to get stuck quite often in local minima. However, an ESG-based UGSAT can escape from those minima quite easily, and does not suffer from this problem. As a result, the reduced intensification of a WalkSAT variant just delays the convergence to a solution. The next two explanations as to why the WalkSAT variant was unsuccessful will be addressed in the next section.

### 2.2.2 Updating Clause Weights Every Local Minimum

In the ESG algorithm, the clause weights are updated at every local minimum. In our very first experiments, we misinterpreted the original ESG description, and had the clause weights updated at every step. Except for a very few quirky exceptions, those results were disastrous. One of the principal motivations behind DLS is that the weighting scheme can dynamically change the search landscape to make local minima appear less attractive to the search algorithm. By weighting clauses at every iteration, the search landscape was being weighted in ways that had very little to do with local minima, and were nullifying one of the principal motivations of DLS. This leads to the second explanation we have as to why the WalkSAT ESG algorithm was unsuccessful.

In most WalkSAT variants, when a minimum appears, the algorithm simply takes the least worsening step. In other words, WalkSAT algorithms do not behave any differently at local minima than they do at any other region of the search space. This mechanism of taking the least worsening step adds to the diversification of the WalkSAT algorithm, and is a key to its' success. Conversely, an ESG-based WalkSAT algorithm won't take the worsening step, and will instead update the clause weights. We believe that this

loss in diversification is only a small reason for why the WalkSAT ESG variant does not perform well, but provides a clue into the third and final reason for poor WalkSAT ESG performance.

In a single WalkSAT step, there are actually two distinct stages. The first non-deterministic stage selects a clause at random, reducing the variables that can be considered in the second stage. In other words, the search neighbourhood is restricted in this second stage. While there may be several improving steps available in the full neighbourhood, there may be none present in the restricted neighbourhood. As a result, a false local minimum can be detected. In an ESG implementation, if the false local minima are treated as true local minima, then the clause weights can be updated in undesirable ways. We implemented a WalkSAT variant that did a full UGSAT step whenever a false local minimum was encountered by WalkSAT. In this implementation, the performance was slightly worse than regular UGSAT. There may be an opportunity to make this implementation more efficient, but we do not believe it would significantly outperform the UGSAT variant.

So far, we have addressed what happens when the weights are updated more frequently than every local minimum, but we have not addressed what happens if they are updated less frequently. This will be discussed at length in Section 3.

### 2.2.3 Sideways Steps

Another characteristic of the ESG algorithm is that it does not take sideways steps, and so it does not search across plateau regions. Because of the nature of the clause weighting scheme, we would not expect plateaus to occur as frequently as they do with schemes with simpler evaluation functions. As such, it would not seem that this characteristic would have a large impact on the performance of the algorithm. However, from our experiments we have found that this behaviour is absolutely critical. The explanation is straightforward. Consider a region in the search space where there are two adjacent local minima with no exit. This may be very common early on in the algorithm, when there is little difference between the clause weights. Since the algorithm has strong intensification, it will continue to switch between the two states until it is forced to take a random walk step. All of this behaviour is unsurprising, and is common with many GSAT or UGSAT algorithms.

One interesting approach might be to virtually eliminate the possibility of sideways steps. This could be achieved by initializing all of the clause weights to unique values. For example, the clause weights could be initialised to a normal distribution with a mean of 1.0 and a small standard deviation. This weight distribution would make the algorithm virtually deterministic after the initialization, and might be an interesting approach to explore in the future.

#### 2.2.4 Best Search, Randomly Breaking Ties

Another feature of the ESG algorithm is that it performs best search, and breaks ties randomly. We did not evaluate this characteristic of the algorithm in any detail. Breaking ties randomly is generally a solid approach in a SLS algorithm, but there is an opportunity to implement a variant that could consider when each of the variables was last flipped, similar to the methods used in TABU [27] and the Novelty<sup>+</sup> variants of WalkSAT [5]. As mentioned previously, a non-uniform clause weight initialization could virtually eliminate the requirement for breaking ties. Although we never tried implementing a first improvement approach, we do not suspect that it would work, and any speed performance would not make up for the loss in intensification.

#### 2.2.5 Random Walk Parameter $wp$

It has been found that most SLS algorithms benefit from having a random walk parameter, even if the random walk probability is very small [24]. Scurmans et al. [12] claim that the added walk probability improves the performance of the ESG algorithm. However, in our experiments we did not encounter any problem instances where the algorithm would become stuck if there was no random walk. This further suggests that the ESG algorithm is very robust at escaping from local minima. Regardless, we think that all SLS algorithms should have a small random walk probability to achieve probabilistic approximate completeness almost for free [24]. From our experience, a value of  $wp = 0.01$  is sufficient.

#### 2.2.6 Scaling Factor $\alpha$

The scaling factor  $\alpha$  controls how much to increase the unsatisfied clause weights whenever a local minimum is encountered. In our ESG implementation, we only increase the weights for unsatisfied clauses, and do not decrease the satisfied clause weights. When examining how  $\alpha$  affects the algorithm, there are some obvious intuitions. First of all, setting  $\alpha = 1$  makes the behaviour exactly the same as a regular UGSAT algorithm, which can easily get trapped in local minimums. Next, setting  $\alpha < 1$  will most likely trap the algorithm in the first local minimum encountered, since it will bias the evaluation function to avoid unsatisfying any of the currently satisfied clauses, which is usually necessary to escape from a local minimum, and always required to escape from a strict local minimum. It is clear that  $\alpha$  must be greater than 1, so the challenge is finding the optimal value.

When deciding on a value of  $\alpha$ , it is convenient to review the original motivation for increasing the clause weights. By weighing unsatisfied clauses when a local minimum is encountered, the algorithm will be steered away from that minimum in the future. If the value of  $\alpha$  is too small, then the small increase in clause weights may not be enough to steer the algorithm away. Conversely, if  $\alpha$  is too large, then it will violently thrust the search algorithm into a region of the search space far away from the current minimum. If a solution were to exist somewhere close to that minimum, then it would be increasingly

difficult to reach it. Additionally, a large value of  $\alpha$  may continuously force the search algorithm in opposing directions, like a “ping-pong” around the search space.

To study the effect of  $\alpha$  on the performance of the algorithm, we looked at problem instances where no smoothing ( $\rho = 1$ ) is required to achieve good performance. In Figure 4, we present the median flip performance for various values of  $\alpha$  for two different problem instances. We were surprised to see that both problem instances had very similar optimum values of  $\alpha$ . We analysed several other problem instances, and it seems that a good “rule of thumb” is to set  $\alpha = 1.3$ . We were surprised by this result, and will explore this further in Section 5.

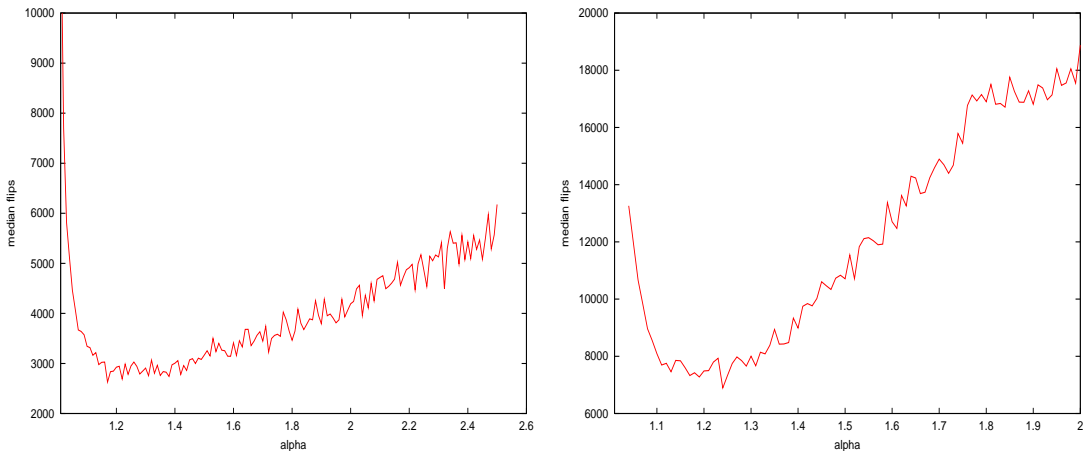


Figure 4: Sensitivity to  $\alpha$ . Problem Instances: `bw_large.a` (left) and `logistics.c` (right),  $w_p = 0.01$ ,  $\rho = 1.0$

## 2.2.7 Smoothing Parameter $\rho$

The final parameter to consider is the smoothing parameter  $\rho$ . The value of  $\rho$  ranges between 1 and 0. A value of 1 corresponds to no smoothing, while a value of 0 corresponds to absolute smoothing. One way of thinking about  $\rho$  is as a mechanism to control the memory of the algorithm. In other words, smoothing is a way of forgetting the weights that were previously scaled. When the value of  $\rho$  is closer to 1, then the algorithm is very slow to forget scaled values. Conversely, when the value of  $\rho$  is close to 0, then the algorithm quickly forgets the scaling.

For some problem instances, the ESG algorithm performs very well without any smoothing. The blocksworld and logistics encoding from the previous section are two such instances. The logistics problems we tested were the rare exceptions that were solved more efficiently with absolutely no smoothing. However, we encountered far more problem instances that benefitted from at least a little smoothing. For example, most uniform random 3-SAT problems require some smoothing to achieve optimal performance.

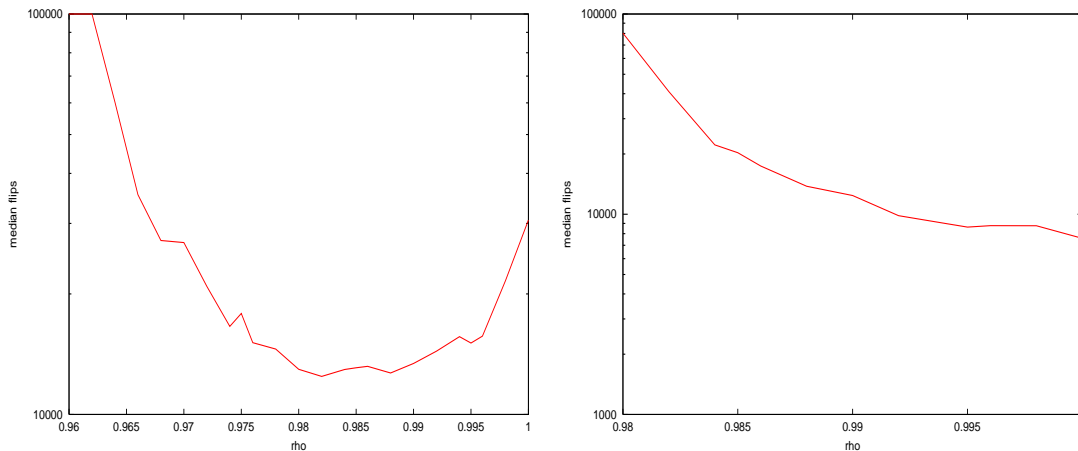


Figure 5: Sensitivity to  $\rho$ . Problem Instances: Hard Uniform Random 3-SAT from uf150 (left) and logistics.c (right),  $w_p = 0.01$ ,  $\alpha = 1.3$

In Figure 5, the sensitivity to parameter  $\rho$  is shown for two different problem instances. For these figures, the value of  $\alpha$  was set to our “rule of thumb” value of 1.3. The behaviour for the hard uniform random 3-SAT problem on the left is characteristic of most of the problem instances we tested. Surprisingly, just as we found safe values for  $\alpha$  sensitivity, it seems that for most problem instances, setting the value of  $\rho$  to 0.99 is a good “rule of thumb”. Certainly there are exceptions, and the logistics encoding in Figure 5 is one of those exceptions. For this instance, 1 is the optimal value of  $\rho$  when  $\alpha$  is 1.3.

While we were very surprised to find a reasonably safe value of  $\rho$ , we were even more surprised by what that value was (0.99). It seems that only a very small amount of smoothing is beneficial. We certainly did not expect that there could be an order of magnitude difference in performance between  $\rho = 0.99$  and  $\rho = 0.96$ . This is an interesting phenomenon that we will further discuss in Section 5.

### 2.3 Problem Hardness, Smoothing and Scaling

In the previous sections, we addressed several characteristics of the ESG algorithm, and how they affect the performance of the algorithm. Two of the more interesting components of the algorithm are the scaling ( $\alpha$ ) and smoothing ( $\rho$ ) parameters. In this section, we will examine the relationship between  $\alpha$  and  $\rho$  and how they relate to the hardness of the problem instance.

As mentioned in Sections 2.2.6 and 2.2.7, setting the default values of  $\alpha$  to 1.3 and  $\rho$  to 0.99 seems to work as a pretty good “rule of thumb”. In Figure 6, we use those default parameters to compare against Novelty<sup>+</sup> and Adaptive Novelty<sup>+</sup>. Clearly, the default values seem to work well, and the step performance of the ESG algorithm is impressive over a wide variety of problem hardness.

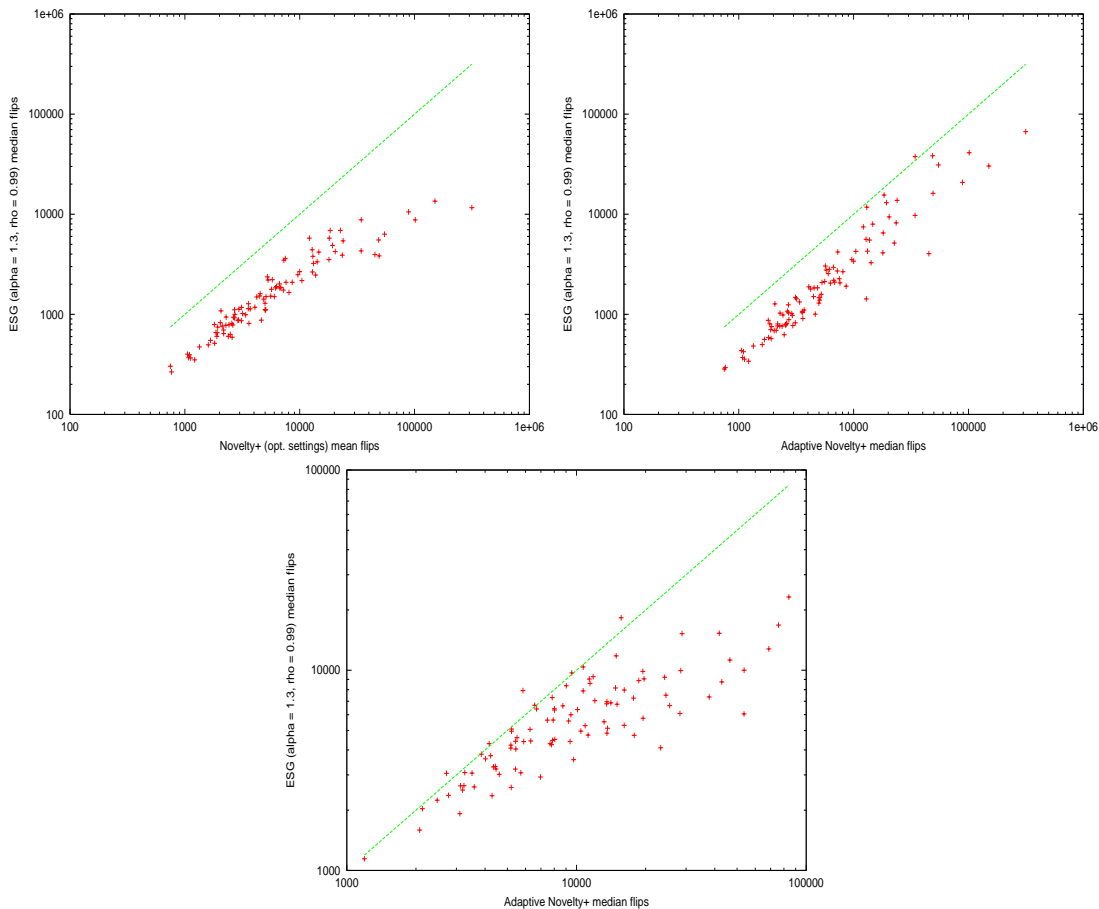


Figure 6: Comparing ESG to Novelty<sup>+</sup> (with optimal noise) (left) and Adaptive Novelty<sup>+</sup> (right,below). Problem Instances: Uniform Random 3-SAT (uf150) (above), Flat Graph Coloring (flat100) (below)

Although the default values for  $\rho$  and  $\alpha$  seem to work well for both hard and easy instances, hard problems are much more sensitive to their parameters than easy problems, which can be seen in Figure 7. Similar results can be seen for  $\rho$  sensitivity as well.

Up to this point, we have only been focusing on varying one parameter at a time, but clearly there will be a relationship between scaling and smoothing. If the scaling is increased, then more smoothing may be required to compensate. In Figures 8 through 10 we present a topology map showing the performance for several values of  $\alpha$  and  $\rho$ .

In Figure 8 we can again see how harder instances are much more sensitive to their parameter settings. In Figure 9 we demonstrate that this sensitivity also applies to flat graph colouring and other problems, not just uniform random 3-SAT. Note that in the figures, the cutoff parameter was set to 100000 flips, so flat regions at 100000 flips can be deceiving.

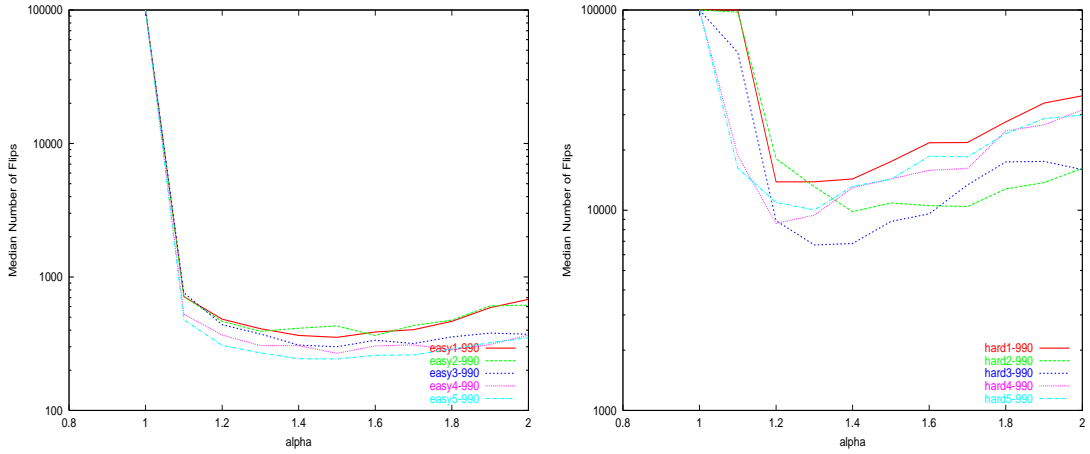


Figure 7: Uniform Random 3-SAT: Sensitivity to  $\alpha$ , Easy Instances (left) and Hard Instances (right). Data set: uf150,  $\rho = 0.99$ ,  $wp = 0.01$ .

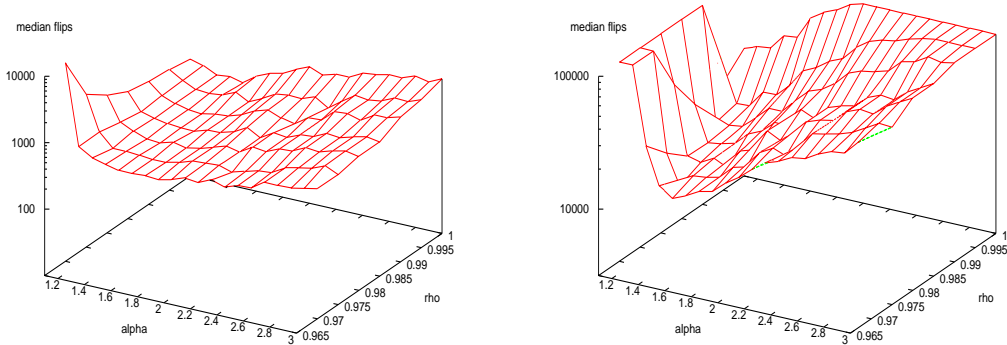


Figure 8:  $\rho - \alpha$  Landscape for Uniform Random 3-SAT (uf150), easy instance (left) and hard instance (right).

In Figure 10 we revisit the instances from Section 2.2.6 that did not require smoothing. In particular, we examine the blockworld problem `bw_large.a`, which is not very sensitive to system parameters. We also plot the landscape for the `logistics.c` encoding, which is a rare example of when smoothing actually decreases the performance of the algorithm.

It would seem feasible that there exists a more formal relationship between the performance of the algorithm, and the values of the  $\alpha$  and  $\rho$  parameters. This relationship will be explored in Section 5.

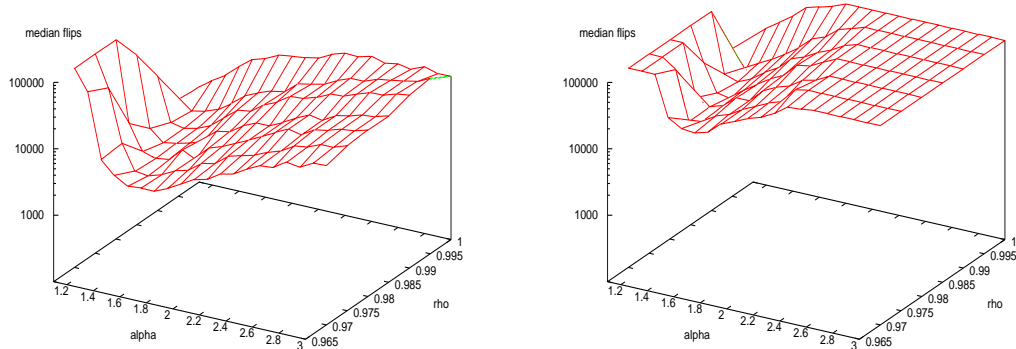


Figure 9:  $\rho - \alpha$  Landscape for Flat Graph Colouring (flat100), easy instance (left) and hard instance (right).

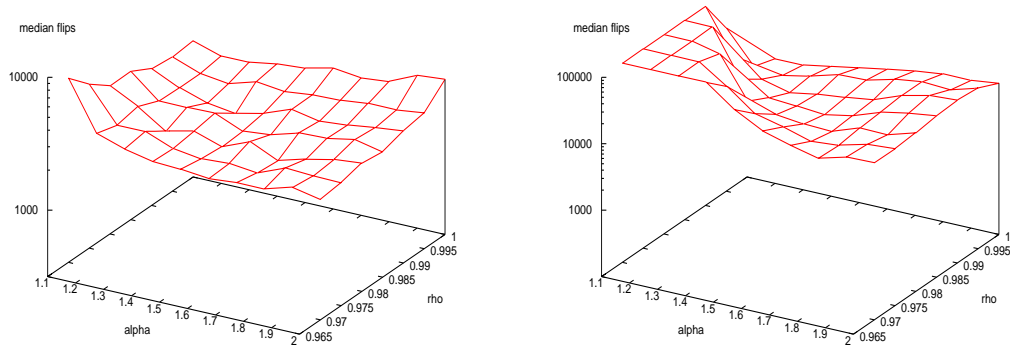


Figure 10:  $\rho - \alpha$  Landscape for Blockworld Encoding bw\_large.a (left) and Logistics Encoding logistics.c (right).

### 3 Scaling and Probabilistic Smoothing (SAPS)

So far, we have only considered the step performance of the ESG algorithm [12], as compared to state of the art algorithms such as the WalkSAT variants Novelty<sup>+</sup> [8] (with approximately optimal noise parameter setting) and Adaptive Novelty<sup>+</sup> [24]. The number of steps is a perfectly robust measure with respect to different evaluation platforms and also gives key insights into how promising a general algorithmic approach is. An ingenious implementation of a well-studied approach can be revealed to be theoretically inferior to a straightforward implementation of a new idea. According to our intuition, the ESG algorithm is such a new approach, capable of being the best choice for a broad range of problems; however, it needs to be understood in more depth and implemented in a way that takes advantage of the gained insights.

A first step toward this direction is given in this section. Realizing that a great deal of time is spent with the smoothing of clause weights, and considering that the smoothing parameter  $\rho$  is a mechanism to forget, we investigate the effects of forgetting more rarely but in larger chunks.

In Section 3.1, we introduce a variant of the ESG algorithm which does not smooth in every local minimum. Rather, it only smooths in local minima with a fixed smoothing probability  $P_\rho$ , whereas the weights of unsatisfied clauses are still scaled by  $\alpha$  in every local minimum; hence, we call it *Scaling and Probabilistic Smoothing* (SAPS). As will be shown in Section 3.2, SAPS shows a major improvement in time performance over ESG, while the step performance stays the same. Since in practice one is concerned about the actual time performance of an algorithm, this resolves the major disadvantage of the original ESG algorithm.

#### 3.1 SAPS Algorithm

In figure 11, we introduce a modified way to update the clause weights. The only difference to the original updating procedure is that the smoothing is applied with a fixed smoothing probability  $P_\rho$  instead of at every local minimum. The rest of the procedure and the main algorithm stays the same as stated in Section 2.1.

In order to see how this can help to improve time performance it is helpful to take a look at the complexity of smoothing as compared to the complexity of scaling: in a particular local minimum, smoothing is applied to all of the clauses, while scaling only occurs for unsatisfied clauses. Thus, denoting the set of clauses with  $C$  and the set of unsatisfied clauses with  $U_c$ , the complexity of the original *Update Weights* procedure is  $\Theta(|C| + |U_c|) = \Theta(|C|)$ , whereas the expected complexity of the variant in Figure 11 is  $\Theta(P_\rho|C| + |U_c|)$ . Since after a short initial search phase the total number of clauses  $|C|$  is far greater than the number of unsatisfied clauses  $|U_c|$ , this complexity is considerably smaller than the original one. One might think the performance of *Update Weights* does not play a key role since this procedure is only called in local minima. However, experiments on uniform random 3-SAT showed that ESG as well as SAPS encounter a local minimum approximately every

```

procedure UpdateWeights( $\pi, s, W, \alpha, \rho, P_\rho$ )
  input:
    problem instance  $\pi$ , candidate solution  $s$ , clause weights  $W$ , scaling factor  $\alpha$ ,
    smoothing factor  $\rho$ , smoothing probability  $P_\rho$ 
  output:
    clause weights  $W$ 
   $C = \{\text{clauses of } \pi\}$ 
   $U_c = \{c \in C \mid c \text{ is unsatisfied in } s\}$ 
  for each  $i$  s.t.  $c_i \in U_c$  do
     $w_i := w_i \times \alpha$ 
  end
  With probability  $P_\rho$  do
    for each  $i$  s.t.  $c_i \in C$  do
       $w_i := w_i \times \rho + (1 - \rho) \times \bar{w}$ 
    end
  end

```

Figure 11: The modified UpdateWeights procedure of SAPS

four to five steps. Due to the UGSAT selection scheme, the complexity of an ESG/SAPS step when not hitting a local minimum is determined by the number of variables  $U_v$  that appear in unsatisfied clauses; since this number is usually only a small constant factor larger than  $|U_c|$ , the time spent for smoothing makes up the major part of time needed by ESG.

Note that when smoothing is not carried out at every local minimum, we don't expect to get the same optimal value for the smoothing parameter  $\rho$ . For example, with a smoothing probability of  $P_\rho = 0.05$ , we smooth every 20 steps on average; since parameter  $\rho = 0.99$  works good as a rule of thumb when smoothing at every local minimum and  $\rho$  is a multiplicative factor, a first guess for a good setting of  $\rho$  with  $P_\rho = 0.05$  might be  $\rho = 0.99^{20} \approx 0.818$  to compensate for the missed smoothing steps.

### 3.2 SAPS Results

Figure 12 shows the average step performance of the algorithm for different values of  $\rho$  when varying the smoothing probability. To illustrate this more clearly, we split the plot, showing results for larger values of  $P_\rho$  in the left and for smaller values of  $P_\rho$  in the right plot, respectively.

As can be seen, the optimal smoothing parameter  $\rho$  gets smaller as we decrease  $P_\rho$ . In Figure 13(left), we see that the optimal step performance for different smoothing probabilities  $P_\rho$  and approximately optimal values for  $\rho$  is stable over quite a big range of values for  $P_\rho$ . The gain in time performance becomes obvious in Figure 13(right).

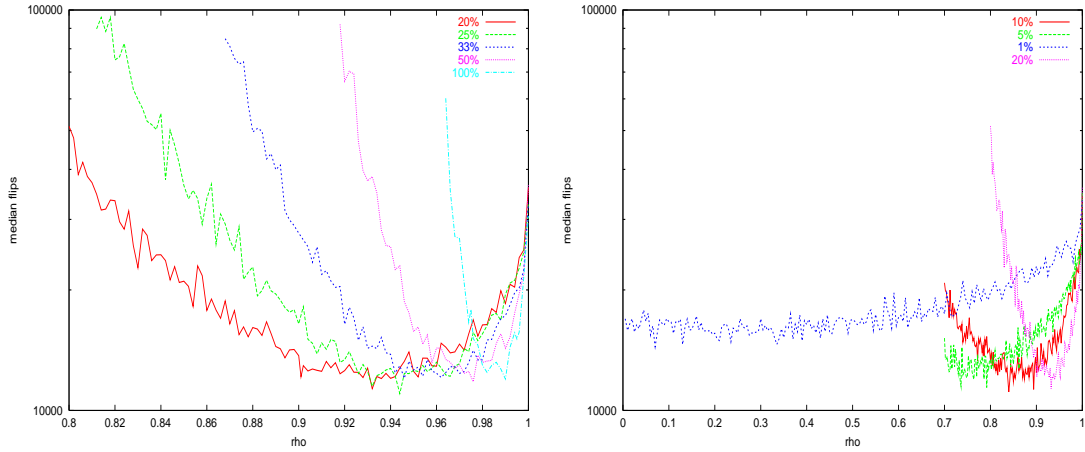


Figure 12: Varying  $\rho$  for different values of  $P_\rho$ . As  $P_\rho$  decreases, the optimal value for  $\rho$  gets smaller, indicating that a more intense smoothing is needed when smoothing is performed less frequently. Larger values of  $P_\rho$ (left), smaller values of  $P_\rho$ (right). Problem Instance: Hard uniform random 3-SAT (uf150).

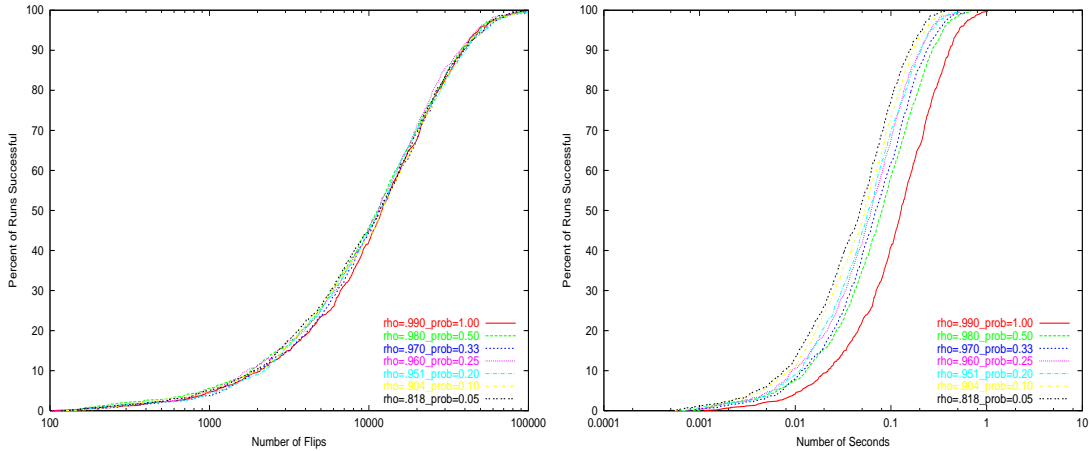


Figure 13: Comparing flip performance (left) and time performance (right) for  $\rho = (0.99)^n$  and  $P_\rho = \frac{1}{n}$ . Problem Instance: Hard Uniform Random 3-SAT (uf150)

## 4 Reactive SAPS (RSAPS)

Although the SAPS algorithm introduced in the last section already performs very well over a broad spectrum of instances, its' parameter settings seem somewhat arbitrary. Indeed, Figure 14 shows that the optimal value of  $\rho$  varies across instances to some degree. It is also not guaranteed that the default choices of  $\alpha = 1.3$  and  $P_\rho = 0.05$  are optimal; nevertheless, we fixed these values and adapted the smoothing parameter  $\rho$ , which appeared

to be the most promising alternative. Adapting  $P_\rho$  and  $\alpha$  or, more interesting, adapting a combination of all three parameters is left for further studies. In this section, we introduce a reactive algorithm to adapt the smoothing parameter  $\rho$  which is highly similar to the approach Hoos describes for adaptive noise setting in WalkSAT Novelty<sup>+</sup> [8]; However, other adaptive strategies could have been implemented.<sup>3</sup>

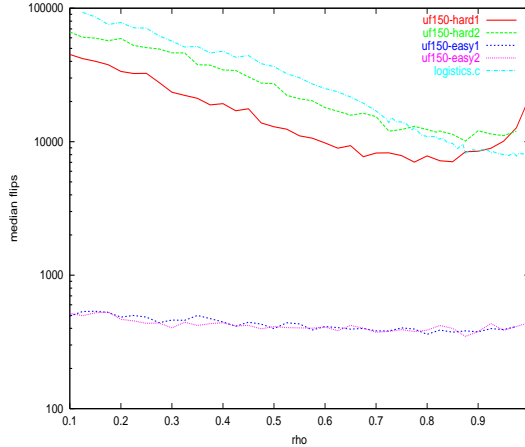


Figure 14: Sensitivity to  $\rho$  for several problem instances.  $\alpha = 1.3$ ,  $P_{rho} = 0.05$ .

The reactive scheme is implemented in a very straightforward way, most likely leaving room for improvement. However, lengthy parameter optimization is rendered unnecessary and, much more important in practice, the reactive scheme also works for problems with an unknown optimal parameter setting. Moreover, in some instances it seems possible to actually adapt parameters in a way that performs even *better* than the optimal fixed parameter setting. Further analysis needs to be done in this direction; Specifically, the optimal parameter settings have not been determined for a larger amount of problem instances.

<sup>3</sup>Preliminary results suggest that another approach to stagnation detection also works well in our framework. This approach is based on the mobility measure introduced by Southey and Schuurmans in [17] and measures the hamming distance traveled over a horizon of  $k$  steps, where we set  $k$  to ten. A simple approach that worked well for uniform random 3-SAT was to assume stagnation (and thus decrease the smoothing) whenever the mobility was smaller than nine. Accordingly, the smoothing was increased whenever the mobility was equal to its' maximal value ten. This stagnation criterion was motivated by statistics that consistently showed a low mobility for runs with too much smoothing. We also carried out preliminary experiments in which we adapted the smoothing probability  $P_\rho$  instead of  $\rho$ . The promising results of these experiments were higher average values of  $P_\rho$  for problem instances that actually profit from smoothing. We also found a bias for more smoothing on harder instances; however, this might be an artifact of the algorithm and needs further investigation. Moreover, superficial experiments on the logistics domain suggest that the optimal mobility is quite domain dependent.

## 4.1 Algorithm

The method we use to adapt the smoothing parameter  $\rho$  is motivated by the following intuition: when an SLS algorithm does not achieve an improvement of the objective function value for quite a long time, it is likely to be stuck in a small region of the search space. The key insight into the role of smoothing is that it acts as an intensification of the search: by smoothing (partly *forgetting*) the differences between clause weights, the algorithm gets more and more similar to a basic UGSAT algorithm. Thus, if we sense stagnation, the smoothing is reduced (which corresponds to an increase of  $\rho$ ). Accordingly, whenever an improvement takes place, the value of  $\rho$  is decreased. This will intensify the search by smoothing the clause weights more in the next local minimum and help to favour the current region of the search space.

The implementation of the stagnation criterion is almost the same as used in [8], the only difference being that in our algorithm the number of steps after which  $\rho$  is increased when no improvement has been made is bigger; specifically, if no improvement has been made in the evaluation function value for  $(\text{number of clauses}) / \theta$  steps then the smoothing is reduced, where  $\theta$  is set to 10. For our algorithm, this setting appeared to work more robust over various domains than the setting of  $\theta = 6$  used in [8]. However, the sensitivity to  $\theta$  is small, and values between one and 50 only resulted in minor performance changes.

Since higher values of  $\rho$  tend to work more robustly, we place a bias on the reactive scheme by increasing  $\rho$  by a larger amount than when decreasing. Both increases and decreases are done exponentially, and are controlled by  $\tau$ , which determines the steepness of the adjustment. This factor is (in accordance with [8]) set to five, but can be changed by small amounts without big performance losses. Pseudocode for *AdaptRho* is given in Figure 15; this procedure is called in every step of the reactive SAPS algorithm (RSAPS), depicted in Figure 16.

```
procedure AdaptRho( $\pi, s, H, \rho$ )  
  input:  
    problem instance  $\pi$ , candidate solution  $s$ , partial search history  $H$ , smoothing factor  $\rho$   
  output:  
    smoothing factor  $\rho$   
   $C = \{\text{clauses of } \pi\}$   
   $\theta := 10$   
   $\tau := 5$   
  if (no improvement has been made for  $|C|/\theta$  steps) then  
     $\rho := \rho + (1 - \rho) * (1/\tau)$   
    mark the current step as the last improvement  
  else if (an improvement has been made in this step) then  
     $\rho := \rho - \rho * (1/4\tau)$   
  end  
end
```

Figure 15: Adaptive setting of the smoothing parameter  $\rho$  in RSAPS.

```

procedure RSAPS( $\pi$ )
  input:
    problem instance  $\pi$ 
  output:
    solution  $s$  or  $\emptyset$ 
   $wp := 0.01$ ,  $\alpha := 1.3$ ,  $P_\rho := 0.05$ ,  $\rho := 0.818$ 
   $s := \text{Init}(\pi)$ 
   $W := \text{InitWeights}(\pi)$ 
  while not terminate( $\pi, s$ ) do
     $s' := \text{WeightedSearch}(\pi, s, W)$ 
    if  $s' = \emptyset$  then
      With probability  $wp$  do
         $s := \text{RandomStep}(\pi, s)$ 
      otherwise do
         $W := \text{UpdateWeights}(\pi, s, W, \alpha, \rho)$ 
      end
    else
       $s := s'$ 
    end
     $H := \text{history of recent search}$ 
    AdaptRho( $\pi, s, H, \rho$ )
  end
  if ( $s$  is not a solution) then  $s = \emptyset$ 
end

```

Figure 16: The RSAPS Algorithm.

## 4.2 Results

In various experiments, we compared the performance of SAPS( $\rho = 0.818$ ) and RSAPS, as well as their performance compared to the current state of the art in SLS SAT solving, namely Adaptive Novelty+ [8] and the original ESG algorithm [12].

Figure 17 compares the performance of SAPS( $\rho = 0.818$ ) and RSAPS on uniform random 3-SAT. We observe that the reactive variant performs slightly better. However, we need to emphasize that a setting of  $\rho = 0.818$  is by no means guaranteed to be optimal. We would have liked to compare RSAPS to SAPS with optimal setting of  $\rho$  and  $P_\rho$  for each problem instance, but these settings are simply not known yet.

Figure 18 compares RSAPS and the original ESG algorithm with smoothing parameter  $\rho = 0.99$  on uniform random 3-SAT, both for step (left) and time performance (right). We observe that the step performance is roughly equal and time performance is improved for RSAPS. We were concerned that the original ESG algorithm works as well as RSAPS when the smoothing is omitted. Fortunately, Figure 18 shows that that the bad step performance also carries over to time performance. ESG outperforms Novelty+ when comparing steps [12] and is only slightly worse when comparing time; thus, we expect RSAPS to perform better than Novelty+ on steps as well as on time, which is confirmed in Figure 19 (here, as before, the more recent variant Adaptive Novelty+ is used).

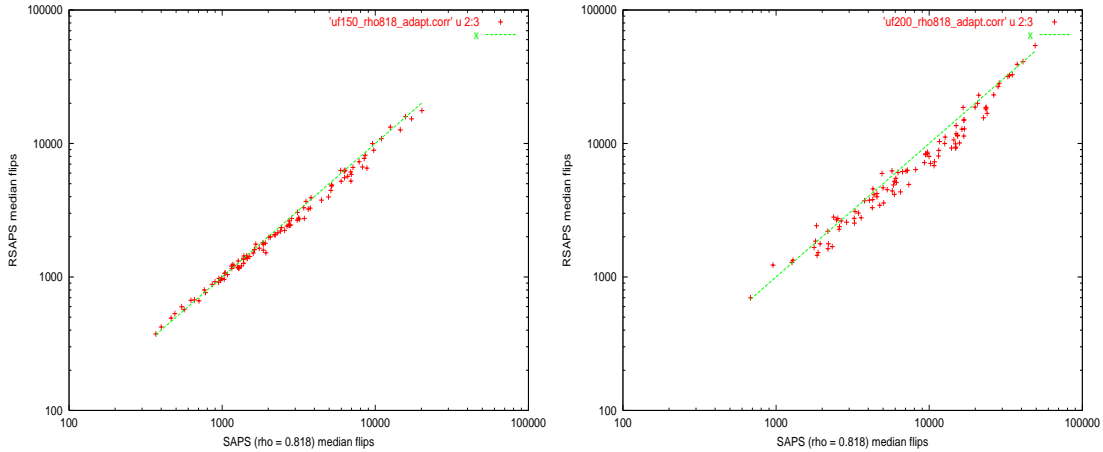


Figure 17: Correlation plots for SAPS( $\rho = 0.818$ ) and RSAPS, on uniform random 3-SAT (uf150, left and uf200, right)

Figure 20 shows that for flat graph coloring the same relation between RSAPS, SAPS( $\rho = 0.8181$ ) and Adaptive Novelty<sup>+</sup> holds as for uniform random 3-SAT. Comparison with ESG is omitted; however, we found the results for uniform random 3-SAT to scale. ESG performed just like RSAPS when comparing steps and considerably worse when comparing run-time.

Figure 21 shows empirical run-time distributions (RTDs) and run-length distribution (RLDs) of the various algorithms on one instance of the logistics planning domain. Since this problem does not require smoothing, the original ESG algorithm (or SAPS with smoothing probability zero) outperforms all the other algorithms. Since the optimal strategy for this problem is not to smooth at all, neither SAPS nor RSAPS can advance on the original ESG (without smoothing). This suggests that SAPS and RSAPS would have to adapt their smoothing probability to reach peak performance, which is left as future work.

Figure 22 shows RTDs and RLDs for one instance of the blocksworld planning domain. The algorithms' performance is quite similar to the logistics domain. Although smoothing is advantageous in this case, it does not pay off when comparing run-times.

In the last two sections, we have developed a DLS algorithm, which we call SAPS, and introduced a sophisticated reactive variant, called RSAPS. Both SAPS( $\rho = 0.818$ ) and RSAPS show robust performance over a wide variety of problems. They outperform Adaptive Novelty<sup>+</sup> in every domain we encountered, both in step and time performance, and are now amongst the state of the art in SAT solvers. However, SAPS and RSAPS are still based upon the original ESG algorithm, and we have not yet demonstrated a solid understanding of how and why that algorithm performs so well. We will try and address this issue in the following section.

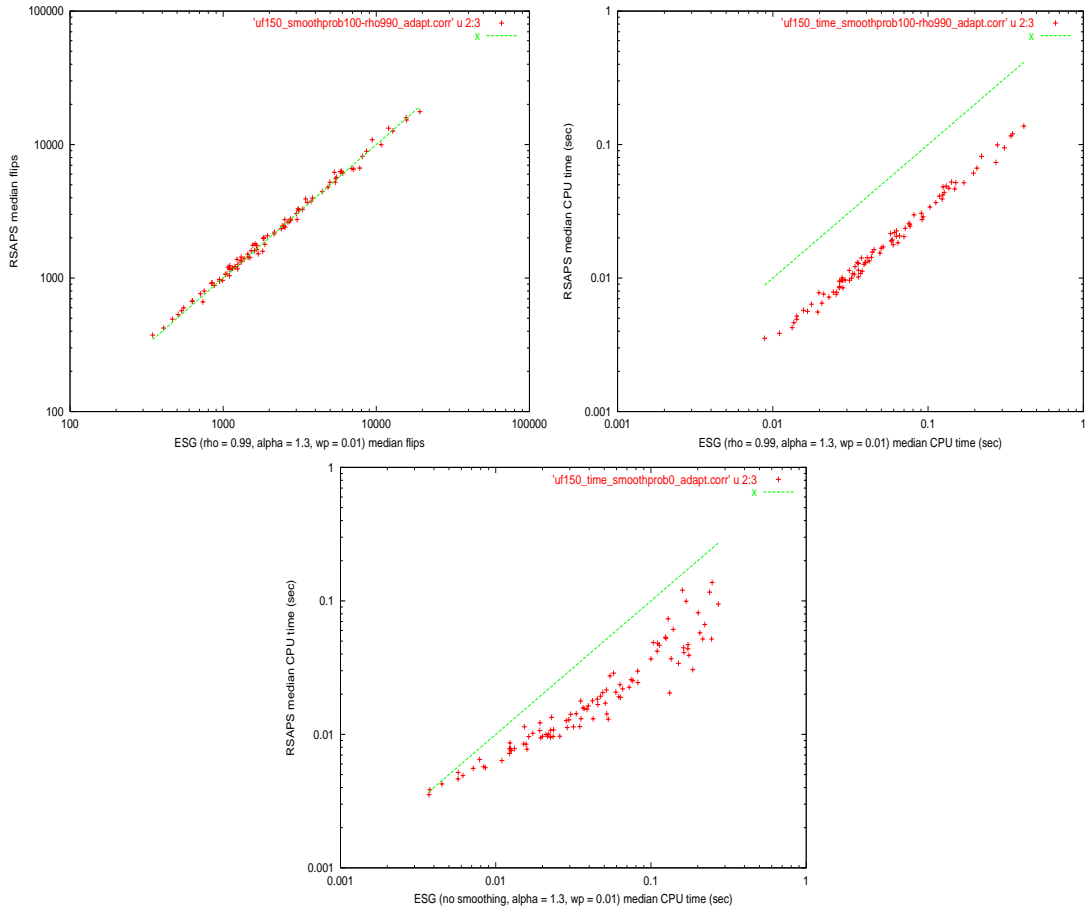


Figure 18: Correlation plots for the original ESG algorithm( $\rho = 0.99$ ,  $\alpha = 1.3$ ,  $wp = 0.01$ ) and RSAPS of step performance (left) and time performance(right) on uniform random 3-SAT (uf150). Bottom: the original ESG algorithm( $\alpha = 1.3$ ,  $wp = 0.01$ ) without smoothing as compared to RSAPS.

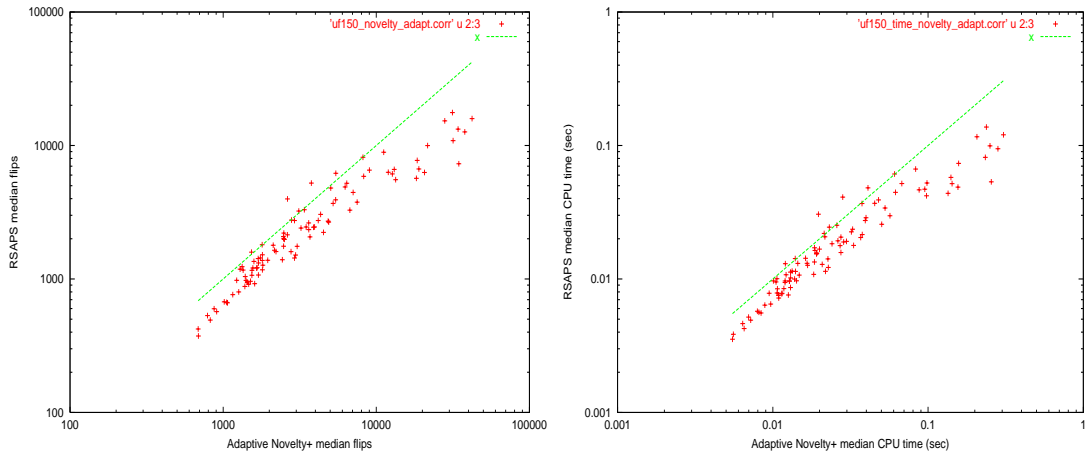


Figure 19: Correlation plots for Adaptive Novelty<sup>+</sup> and RSAPS of step performance (left) and time performance(right) on uniform random 3-SAT (uf150)

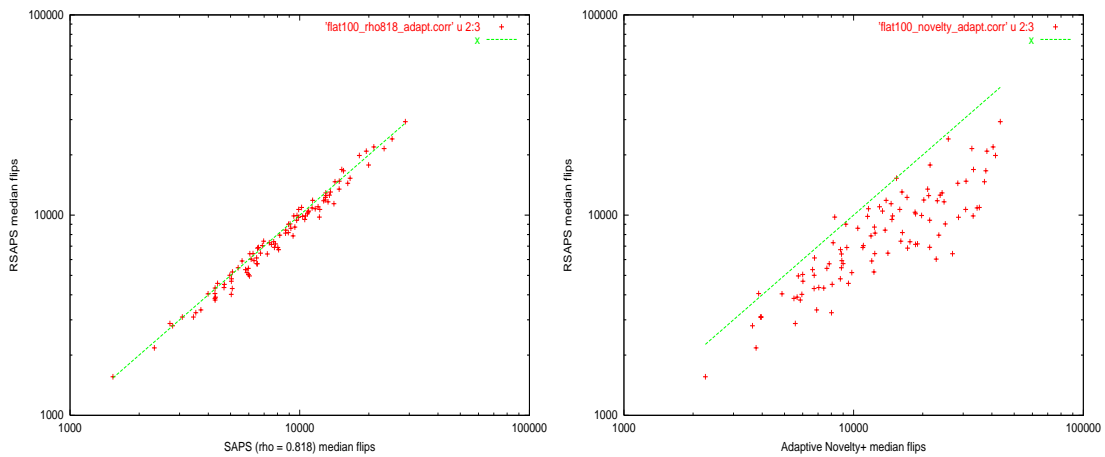


Figure 20: Correlation plots on step performance for flat graph coloring (flat100), RSAPS vs. SAPS( $\rho = 0.818$ ) (left) and Adaptive Novelty<sup>+</sup> (right) respectively.

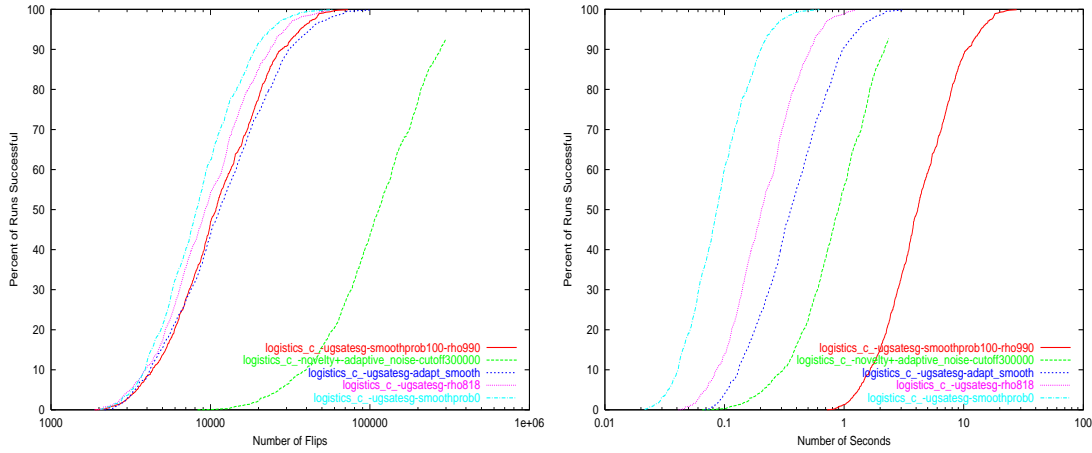


Figure 21: Empirical length (left) and run-time (right) distributions for SAPS( $\rho = 0.818$ ), RSAPS, ESG without smoothing( $\alpha = 1.3$ ,  $wp = 0.05$ ), ESG with smoothing( $\rho = 0.99$ ,  $\alpha = 1.3$ ,  $wp = 0.05$ ) and Adaptive Novelty<sup>+</sup> for the logistics planning problem logistics.c.

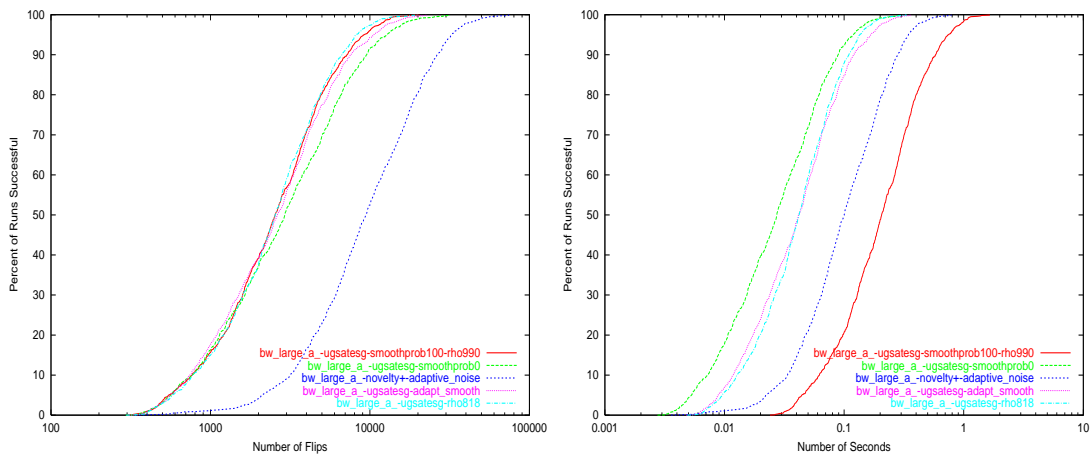


Figure 22: Empirical length (left) and run-time (right) distributions for SAPS( $\rho = 0.818$ ), RSAPS, ESG without smoothing( $\alpha = 1.3$ ,  $wp = 0.05$ ), ESG with smoothing( $\rho = 0.99$ ,  $\alpha = 1.3$ ,  $wp = 0.05$ ) and Adaptive Novelty<sup>+</sup> for the blocksworld planning problem bwlarge.a.

## 5 Understanding How ESG Works

We have presented SAPS and RSAPS, two new ESG-based variants that can solve many SAT problems quite effectively. However, we still do not have a complete understanding of how the properties, mechanisms and dynamics of the core ESG algorithm achieve such impressive performance. To help better understand the larger picture, we shall re-examine our original interpretations and intuition into the search strategy of the ESG algorithm.

The core of the search strategy tries to minimise the evaluation function value, which is the sum of the unsatisfied clause weights. To achieve this, variables are tested to see what the potential difference in evaluation function value would be if that variable was flipped. The variable that can achieve the best decrease is flipped. If none of the variables can decrease the current evaluation function value, then a local minimum has been reached, and the clause weights are updated.

In the ESG, SAPS and RSAPS algorithms, clause weights are scaled at every local minimum. However, the rate at which the smoothing occurs depends on the algorithm. We will use the notation of *scaling steps* and *smoothing steps* to differentiate. We will restrict our first analysis to the ESG algorithm and cover the effect of smoothing probability  $P_\rho < 1$  later in this section. The ESG scaling step is given in Equation 1, and the smoothing step is given in Equation 2.

$$w_i := \begin{cases} w_i \times \alpha & \text{if clause } i \text{ is unsatisfied} \\ w_i & \text{otherwise} \end{cases} \quad (1)$$

$$w_i := w_i \times \rho + (1 - \rho) \times \bar{w} \quad (2)$$

Each variable, when flipped, will satisfy some clauses and will unsatisfy others. The higher the value of a clause’s weight, the more likely the algorithm is to try and make (or keep) that clause satisfied. For convenience of notation, we will say that a clause with a higher weight will be *favoured* by the algorithm.

To help better understand how the clause weights change over time, we will look at clauses with different histories. First, we will consider the scenario in Figure 23, where no smoothing is applied. In this figure, we see that the weight for a clause is simply  $\alpha^n$ , where  $n$  is the total number of times the clause was unsatisfied. There is no distinction with respect to the order in which clauses are unsatisfied, and so clauses (b) and (c) have the same weight.

Let’s now consider how our default “rule of thumb” value ( $\alpha = 1.3$ ) would affect these weights. Clause (a) would be favoured with a weight of  $1.3^2 = 1.69$ , while clauses (b) and (c) would have weights of 1.3. Now we shall consider how the algorithm will behave when it must choose between two *sets* of clauses. Using the default value of 1.3, after the first scaling step the algorithm will favour 4 unsatisfied clauses (5.2) over 5 satisfied clauses (5.0), and will treat 13 satisfied clauses and 10 unsatisfied clauses equally. Furthermore,

scaling step	$w_a$	$w_b$	$w_c$	unsatisfied clauses
0	1	1	1	
1	$\alpha$	1	1	(a)
2	$\alpha$	$\alpha$	1	(b)
3	$\alpha$	$\alpha$	1	
4	$\alpha^2$	$\alpha$	$\alpha$	(a,c)

Figure 23: Clause Weights with No Smoothing ( $\rho = 1$ )

if a clause was unsatisfied three times (2.197), it will be favoured over 2 clauses that have always been satisfied. This is an interesting dynamic, and it is hard to appreciate how it will affect behaviour after 100 steps or 100000 steps.

Having gained some intuition into how scaling works, we will now examine smoothing. As we mentioned previously, smoothing can be thought of as a mechanism to “forget” scaling that has already occurred. We will look into the difference between the weights of two clauses in Figure 24, where one of them was initially unsatisfied, and subsequently they both stay satisfied. The difference between the two clause weights is  $(\alpha - 1)\rho^n$  after  $n$  smoothing steps have occurred.

scaling step	smoothing step	$w_a$	$w_b$	$w_a - w_b$	unsatisfied clauses
0	0	1	1	0	
1	1	$\alpha$	1	$(\alpha - 1)$	(a)
	2	$\alpha\rho + (1 - \rho)\bar{w}$	$\rho + (1 - \rho)\bar{w}$	$(\alpha - 1)\rho$	
	3			$(\alpha - 1)\rho^2$	
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
	$n$			$(\alpha - 1)\rho^n$	

Figure 24: The Difference Between Two Clause Weights Over Time. In step 1, (a) is unsatisfied, after which clauses (a) and (b) are always satisfied.

Our original intuition was that the first component of Equation 2 ( $w_i \times \rho$ ) was in place as a mechanism to eventually forget the scaling. However, as we can see from Figure 24, the difference between the two clauses will approach zero, but never reach it. In other words, in the example of Figure 24, the algorithm will *always* favour clause (a). For perspective, using the default values of  $\alpha = 1.3$  and  $\rho = 0.99$ , the difference between the clause weights will be 0.297, 0.271, 0.109 and 0.000013 after 1, 10, 100 and 1000 steps respectively.

Since our intuition regarding the first component of Equation 2 failed us, we must re-

consider what role it does play in the algorithm. Surprisingly, we found it to be completely unnecessary. We removed that component and found no performance loss whatsoever; There was even a marginal increase in speed due to the removal of a multiplication. In Figure 25, we compared the results with and without the multiplication by  $\rho$  and found no difference. We were concerned that after a large number of flips, this component may have more impact, so we compared the 0.99 percentile of our results and found no difference there either. For all problem instances we tested, there was no difference between the two implementations. Since we have determined that scaling all of the clauses has no impact on performance, it may be useful to perform scaling for other reasons. For example, In our implementation, we found it helpful to re-scale the weights to avoid precision errors.

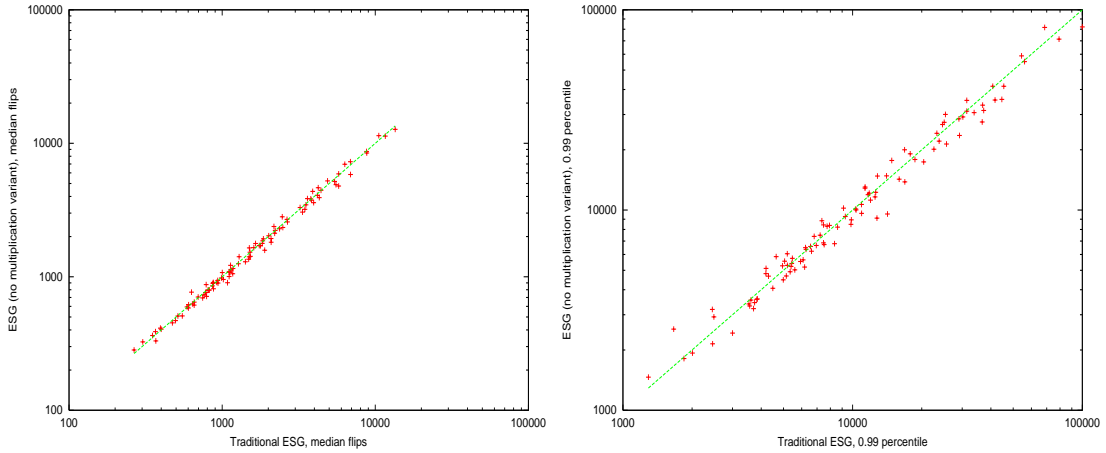


Figure 25: Correlation between traditional ESG, and a new variant without the multiplication by  $\rho$ : Median (left) and 0.99 percentile (right). Problem Instances: Uniform Random 3-SAT (uf150).

Since we are no longer multiplying by  $\rho$ , it does not seem appropriate to keep the same notation. In Equation 3, we have re-written the expression, and have replaced  $(1 - \rho)$  with a new constant  $k$ .

$$w_i := w_i + k\bar{w} \tag{3}$$

We will now consider how the mechanics of Equation 3 effect the algorithm. In Figure 26, we look at the weights of three different clauses. The two steps in this example are identical to the first two steps in Figure 23, but there are critical differences between the resulting weights. The most important difference is that now after two steps, the more recently satisfied clause (b) will be favoured over clause (a), even though they have been unsatisfied the same number of times. Using our default values of  $\alpha = 1.3$  and  $k = 0.01$ , and assuming  $\bar{w} \approx 1$  for simplicity, the weights of (a), (b) and (c) would be 1.32, 1.323 and 1.02 respectively. We believe that this phenomenon is an important clue into how the ESG algorithm behaves. If  $k$  is nonzero and positive, the weights will continue to increase. As the weights increase, the distance between subsequently satisfied clauses will

also increase. In other words, the algorithm will always favour clauses that were unsatisfied more recently.

scaling step	smoothing step	$w_a$	$w_b$	$w_c$	unsatisfied clauses
0	0	1	1	1	
1		$\alpha$	1	1	(a)
	1	$\alpha + k\bar{w}$	$1 + k\bar{w}$	$1 + k\bar{w}$	
2		$\alpha + k\bar{w}$	$\alpha + \alpha k\bar{w}$	$1 + k\bar{w}$	(b)
	2	$\alpha + 2k$	$\alpha\bar{w} + (1 + \alpha)k\bar{w}$	$1 + 2k\bar{w}$	

Figure 26: The Difference Between Clause Weights Over Time. For simplicity, we assume  $\bar{w} \approx 1$

At this point, we would like to depart from some of our previous interpretations, and consider modeling the ESG algorithm as a clause-based TABU mechanism. Although this is clearly not an accurate model of the system, it does have some interesting similarities. The first critical difference is that since we are dealing with clauses instead of variables, our notion of TABU must be different. If a clause is taboo, then it must be satisfied. The next big difference is that the clauses are never truly taboo. Instead of a “hard” TABU list, we have a “soft” list of clauses that become increasing less taboo over time. This coincides nicely with our previous notion of smoothing.

Over time, with the continuous addition of  $k\bar{w}$ , the relative difference between two satisfied clauses will approach zero. When  $k$  is increased, this convergence to zero will occur more quickly. This change would correspond to a steeper soft-TABU list, and a shorter TABU tenure. This relation between  $k$  and the soft-TABU “tenure” is illustrated in Figure 27 as a Clause Weight Distribution (CWD). In hard-TABU, the weights would appear as a step function, but with soft-TABU, they appear as a curve. From this figure, we can see how the soft-TABU “tenure” is reduced when  $k$  is increased. Decreasing a hard-TABU tenure decreases diversification (mobility), and we expect that the same applies to soft-TABU, and so an increase in  $k$  will decrease the diversification. This is consistent with preliminary experiments showing that an increase of  $\rho$  improves mobility. To summarize, an increase in  $\rho$  corresponds to a decrease in  $k$ , hence a longer TABU tenure, which also improves mobility, so our model seems to be holding up.

After seeing how different values of  $k$  affect the CWD, it might now be useful to revisit  $\alpha$ . The CWD for varying values of  $\alpha$  is shown in Figure 28. Not too surprisingly, increasing the value of  $\alpha$  will generally raise the clause weight values. Comparing Figures 27 and 28, we finally get some insight into how the values of  $\alpha$  and  $k$  ( $\rho$ ) interact and determine the behaviour of the search algorithm. From this one instance, it appears that  $k$  changes the shape of the curve, whereas  $\alpha$  changes the height. Another interesting feature is that the value of  $k = 0.01$  appears to be smoother than the other values, which may provide

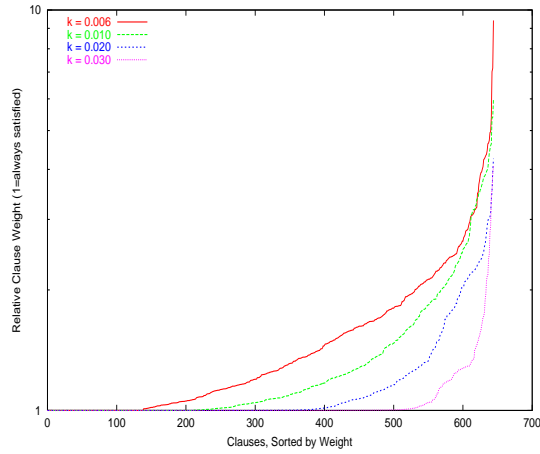


Figure 27: The Effect of parameter  $k$  on Clause Weight Distribution after 400 steps. Problem Instance: Hard Uniform Random 3-SAT,  $\alpha = 1.3$ ,  $wp = 0$ , all runs initialised with the same random seed.

some insight into our “rule of thumb” values. However, we do not want to jump to any premature conclusions. To properly understand this phenomenon, it will take a significant amount of additional research.

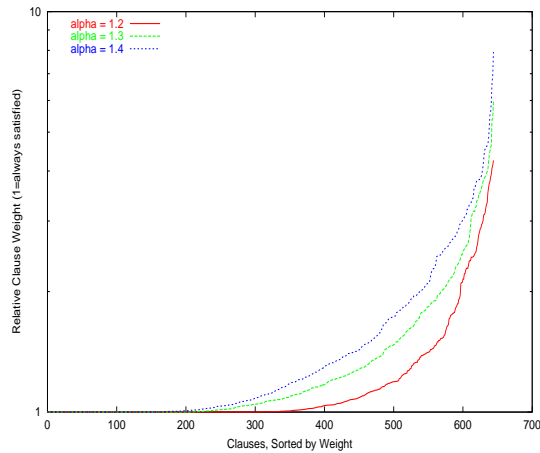


Figure 28: The Effect of parameter  $k$  on Clause Weight Distribution after 5000 steps. Problem Instance: Hard Uniform Random 3-SAT,  $k = 0.01$ ,  $wp = 0$ , all runs initialised with the same random seed.

So far, our analysis has been restricted to the original ESG algorithm and we have not considered our SAPS or RSAPS algorithms. We have demonstrated with the SAPS and RSAPS algorithms that it is not necessary to smooth every step. There can be numerous scaling steps occurring between smoothing steps. We will refer to that time between

smoothing steps as a *smoothing window*. A high smoothing probability corresponds to a short window, and vice-versa. It now becomes apparent that all of the TABU-like behaviour we were describing does not occur within the smoothing window, but rather between windows. If two clauses with the same weight are both made unsatisfied within a smoothing window, they will continue to have the same weights, even if they are unsatisfied at different scaling steps. When we increase the window size, we would expect that  $k$  would have to increase to compensate. Referring back to Section 3, we have a startling result. Previously, we found that an approximately optimal value of  $\rho$  can be found by setting  $\rho = (0.99)^n$  when  $P_\rho = \frac{1}{n}$ . This was motivated by Equation 2. However, we have demonstrated that the use of  $\rho$  in that equation is ineffectual, and indeed, the SAPS and RSAPS algorithm perform well without the multiplication by  $\rho$ . So it would seem that we must re-evaluate our methods to find a suitable value of  $k$ . In particular, we believe that the values of  $P_\rho$  and  $k$  will interact and change the CWD in interesting ways.

In this section, we have looked in depth at how the ESG-based algorithm behaves, and have attempted to explain why the algorithm works so well. Although we have gained interesting insights, there are still many puzzle pieces missing before we can grasp the big picture.

## 6 Conclusions & Future Work

In this report, we focused on the recent and promising Exponentiated Sub-Gradient (ESG) algorithm, and examined its' dependence on specific parameter settings. We proposed a new variant of this algorithm that we call Scaling and Probabilistic Smoothing (SAPS), which significantly improves upon the time performance of the original ESG algorithm. We also provide a reactive version of SAPS (RSAPS) that can adaptively tune one of the algorithm parameters. We demonstrated how our two algorithms are now amongst the state of the art SAT solvers, and then went on to further study how these ESG-based algorithms are achieving their exceptional performance.

Throughout this document, we identified numerous opportunities for further research and development. Some of the more promising topics that we would like to explore further are:

- Clause Weight Distributions (CWDs) and how  $(\alpha, k, P_\rho)$  affect them
- How are CWDs affected by different problem instances
- Achieving the same CWD behaviour with simpler algorithms
- Adapting  $P_\rho$ , especially for the planning benchmarks
- Determine optimal  $(\alpha, k, P_\rho)$  settings for more problem instances
- Better understand the interdependence between the parameters  $(\alpha, k, P_\rho)$

## References

- [1] Holger H. Hoos. *Stochastic Local Search - Methods, Models, Applications*, PhD thesis, Department of Computer Science, Darmstadt University of Technology, 1998.
- [2] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search - Foundations and Applications*, Morgan Kaufmann Publishers, to appear in 2003.
- [3] 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, AAAI Press, 1992.
- [4] Bart Selman and Henry A. Kautz. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 290–295, Morgan Kaufmann Publishers, 1993.
- [5] Bart Selman and Henry A. Kautz and Bram Cohen. Noise Strategies for Improving Local Search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343, AAAI Press, 1994.
- [6] David A. McAllester and 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, AAAI Press, 1997.
- [7] Donald J. Patterson and Henry Kautz. Auto-Walksat: a Self-Tuning Implementation of Walksat. Presented at the *LICS 2001 Workshop on Theory and Applications of Satisfiability Testing*, Boston University, MA, June 14-15, 2001.
- [8] Holger H. Hoos. An Adaptive Noise Mechanism for WalkSAT, submitted to the American Association for Artificial Intelligence (AAAI), 2002.
- [9] Andrew J. Parkes and Joachim P. Walser. Tuning Local Search for Satisfiability Testing. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 356–362, AAAI Press, 1996.
- [10] Eric Horvitz, Yongshao Ruan, Carla Gomes, Henry A. Kautz, Bart Selman and Max Chickering. A bayesian approach to tackling hard computational problems, In *Proceedings of the Seventeenth Conference On Uncertainty in Artificial Intelligence (UAI-01)*, Morgan Kaufmann Publishers , 2001.
- [11] Carla P. Gomes and Bart Selman. Algorithm Portfolio Design: Theory vs. Practice. In *Proceedings of the Thirteenth Conference On Uncertainty in Artificial Intelligence (UAI-97)*, Morgan Kaufmann Publishers, 1997.
- [12] Dale Schuurmans, Finnegan Southey, and Robert. C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 334-341, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001.
- [13] Jeremy Frank. Learning short-term clause weights for GSAT. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 384-389, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.

- [14] Jeremy Frank. Weighting for Godot: Learning Heuristics for GSAT. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-96)*, pages 338–343, AAAI Press, 1996.
- [15] Byungki Cha and Kazuo Iwama. Adding new clauses for faster local search. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-96)*, pages 332–337, AAAI Press, 1996.
- [16] Zhe Wu and Benjamin W. Wah. Trap Escaping Strategies in Discrete Lagrangian Methods for Solving Hard Satisfiability and Maximum Satisfiability Problems, In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 673–678, AAAI Press, 1999.
- [17] Dale Schuurmans, and Finnegan Southey. Local search characteristics of incomplete SAT procedures. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, AAAI Press, 2000.
- [18] Roberto Battiti and Giampietro Tecchioli. The reactive tabu search. In *ORSA Journal on Computing*, Volume 6, No.2, pages 126–140, 1994.
- [19] Roberto Battiti. Reactive Search: Toward Self-Tuning Heuristics. Printed in *Modern Heuristic Search Methods*, chapter 4, pages 61–83, John Wiley and Sons Ltd, 1996.
- [20] Roberto Battiti and Marco Protasi. Reactive Search, a History-Sensitive Heuristic for Max-Sat. In *The ACM Journal of Experimental Algorithmics*, Volume 2, Article 2, 1997.
- [21] Roberto Battiti. The Reactive Search (RS) Home Page. <http://rtm.science.unitn.it/~battiti/reactive.html>
- [22] Holger H. Hoos and Thomas Stützle. Local Search Algorithms for SAT: An Empirical Evaluation. In *Journal of Automated Reasoning*, Volume 24, Number 4, pages 421–481, 2000.
- [23] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 40–45. AAAI Press, 1993.
- [24] 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. AAAI Press, 1999.
- [25] Holger H. Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT. In I.P. Gent, H.v. Maaren and Toby Walsh, editors, *SAT 2000*, pages 283 – 292, IOS Press, 2000.
- [26] Henry Kautz and Bart Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201. AAAI Press, 1996.
- [27] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, London, 1997.