Scaling and Probabilistic Smoothing (SAPS)

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

1 Preface

The first part of this paper is essentially a reprint of the solver description from the 2005 competition, as the software submitted this year is identical to the 2005 software. We entered the SAPS variant implemented in the UBCSAT software package [7], the source code for which is freely available at `http://www.satlib.org/ubcsat`.

The only difference to 2005 is that we submit two versions of SAPS, one with the original default parameters [4], and one with a new set of tuned parameters. These parameters have been found using an automatic approach based on local search in parameter space [3]. Section 3 gives a brief overview of this tuning approach and shows very promising performance of the automatically tuned parameters.

2 SAPS and Variants

The SAPS algorithm is a Dynamic Local Search (DLS) algorithm conceptually closely related to the Exponentiated Sub-Gradient (ESG) algorithm developed by Schuurmans, Southey and Holte [5]. When introducing SAPS, our major contributions were a reduction in the algorithmic complexity as compared to the ESG algorithm and a new perspective on how the two algorithms were behaving. The SAPS algorithm is described in detail in our paper [4] and Figure 1 contains a pseudo-code representation that accurately reflects how the SAPS algorithm has been implemented in practice.

Similar to most DLS algorithms, SAPS assigns a clause penalty clp to each clause, and the search evaluation function of SAPS is the sum of the clause penalties of unsatisfied clauses. The core search procedure is a greedy descent without sideways steps. Whenever a local minimum occurs (no step improvement in the evaluation function greater than $SAPS_{thresh}$ is possible) a random walk step occurs with probability wp. Otherwise, a *scaling step* occurs, where the penalties for unsatisfied clauses are multiplied by the scaling factor α (*i.e.* $clp' := \alpha \cdot clp$). After a scaling step, a *smoothing step* occurs with probability P_{smooth}. In a smoothing

```
procedure SAPS($F$, $\alpha$, $\rho$, $wp$, $P_{smooth}$, $SAPS_{thresh}$)
  input:
    propositional formula $F$, scaling factor $\alpha$,
    smoothing factor $\rho$, random walk probability $wp$,
    smoothing probability $P_{smooth}$,
    SAPS threshold $SAPS_{thresh}$
  output:
    variable assignment $A$

  for $i$ := 1..$|A|$ do $a(i)$ := RandSelect($\{\top, \bot\}$)
  for $j$ := 1..$|CLP|$ do $clp(j)$ := 1
  while ($F$ is unsatisfied under $A$) do
    $curScore$ := Eval($F, A, CLP$)
    $bestScore$ := $\infty$
    $BestVars$ := $\varnothing$
    for each $i$ s.t. $variable i$ appears in an unsatisfied clause do
      $score$ := Eval($F$,Flip$(A, i), CLP$)
      if $score < bestScore$ then
        $bestScore$ := $score$
        $BestVars$ := $\{i\}$
      else if $score = bestScore$ then
        $BestVars$ := $BestVars \cup \{i\}$
      end if
    end for
    if $(bestScore - curScore) < SAPS_{thresh}$ then
      $k$ := RandSelect($BestVars$)
      $A$ := Flip($A, k$)
    else
      with probability $wp$ do
        $k$ := RandSelect($\{1..|A|\}$)
        $A$ := Flip($A, k$)
      otherwise
        for each $j$ s.t. $clause j$ is unsatisfied under $A$ do
          $clp(j) := clp(j) \times \alpha$
        end for
        with probability $P_{smooth}$ do
          for $j$ := 1..$|CLP|$ do
            $clp(j) := clp(j) + (1 - \rho) \times \overline{clp}$
          end for
        end with
      end with
    end if
  end while
  return ($A$)
end procedure $SAPS$
```

Figure 1: The SAPS algorithm. For each clause j in F there is a clause penalty $clp(j)$ in CLP, and $\overline{clp}$ is the mean of all clause penalties. Eval(F, A, CLP) is the sum of all $clp(j)$ where $clause j$ is unsatisfied in F by A. In practice, Eval(...) values are cached and updated after each flip. Flip(A, i) returns the variable assignment A with variable i flipped.

Parameter	Default value	Values considered for tuning	Tuned value for random instances
α	1.3	1.01, 1.066, 1.126, 1.189, 1.256, 1.326, 1.4	1.126
ρ	0.8	0, 0.17, 0.333, 0.5, 0.666, 0.83, 1	0.666
P_{smooth}	0.05	0, 0.033, 0.066, 0.1, 0.133, 0.166, 0.2	0.033
wp	0.01	0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06	0.06
$SAPS_{thresh}$	-0.1	-0.1	-0.1

Table 1: SAPS parameters, and their default and tuned values. We considered seven values for each of the four tuning parameters, equally spaced on a grid with a manual chosen upper and lower bounds (only for the multiplicative parameter α, we used a logarithmic grid).

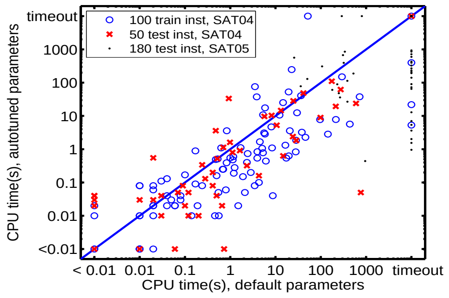

Figure 2: Performance of tuned SAPS parameters vs. its defaults on training and test data.

step, all penalties are adjusted according to the mean penalty value $\overline{clp}$ and the smoothing factor ρ (*i.e.* $clp' := clp + (1 - \rho) \cdot \overline{clp}$).

Along with the SAPS algorithm, we also developed a reactive variant (RSAPS) [4] that reactively changes the smoothing parameter ρ during the search process whenever search stagnation is detected, using the same adaptive mechanism as Adaptive Novelty$^+$ [1]. More recently we have developed a *de-randomised* variant of SAPS called SAPS/NR [8], which eliminates all sources of random decisions throughout the search (breaking ties deterministically, performing periodic smoothing, and no random walk steps) and which relies upon the initial random variable assignment as the only source of randomness.

In our experiments, we have found that SAPS, RSAPS and SAPS/NR are amongst the state-of-the-art SLS SAT solvers, and each typically performs better than ESG, and the best WalkSAT variants *e.g.*, Novelty$^+$ [4]. We have also conducted experiments that show SAPS is similarly effective on MAX-SAT problem instances [6].

3 Automated Parameter Tuning

We are currently performing research in automatic methods for parameter adjustment (for both local search and tree search algorithms) and applied one of our methods for tuning SAPS. This method, called ParamILS [3], views parameter tuning as an optimisation problem. In a nutshell, it performs an iterated local search in parameter configuration space, computing the objective function as the median runtime of SAPS for solving a fixed number of N instances (we used $N = 100$).

We considered the four SAPS parameters (α, ρ, wp, P_{smooth}) for tuning, fixing the fifth one, $SAPS_{thresh}$ (but tuning $SAPS_{thresh}$ would also be interesting). In previous research [4, 6, 2], we noticed that optimal parameter values can vary a lot and so we allowed a wide range of parameter values; we summarise our choices in Table 1.

We tuned SAPS on 100 random instances from the SAT04 competition and tested on the remaining 50 random SAT04 instances, as well as on the 180 available random instances from the SAT05 competition. Figure 2 shows the results. Thus, we expect the tuned version of SAPS to outperform the default version quite clearly on random instances. We cannot say anything about its performance on other types of instances, but, using ParamILS, it would be very easy to tune SAPS for good performance on a more general distribution of instances as well (possibly loosing peak performance for specialised instances).

References

[1] H. H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proc. AAAI-02*, pages 655–660, 2002.

[2] F. Hutter, Y. Hamadi, K. Leyton-Brown, and H. H. Hoos. Performance prediction and automated tuning of randomized and parametric algorithms. In *Proc. CP-06*, pages 213–228, 2006.

[3] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. Under review.

[4] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proc. CP-02*, pages 233–248, 2002.

[5] D. Schuurmans, F. Southey, and R. C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proc. IJCAI-01*, pages 334–341, 2001.

[6] D. A. D. Tompkins and H. H. Hoos. Scaling and probabilistic smoothing: Dynamic local search for unweighted MAX-SAT. In *Proc. AI-03*, pages 145–159, 2003.

[7] D. A. D. Tompkins and H. H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Proc. SAT04*, pages 305–319, 2004.

[8] D. A. D. Tompkins and H. H. Hoos. Warped landscapes and random acts of SAT solving. In *Proc. ISAIM-04*, 2004.