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

April 30, 2004

1 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 [3]. 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 [2] 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 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) [2] 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 [6], 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$^+$ [2]. We have also conducted experiments that show SAPS is similarly effective on MAX-SAT problem instances [5].

2 Contest Implementation

For the SAT 2004 competition, we entered three algorithm variants: SAPS, RSAPS and SAPS/NR. All three algorithms were implemented in the UBCSAT software package [7], the source code for which is freely available at `http://www.satlib.org/ubcsat`. For all three algorithms, the default parameters were used (α, ρ, wp, P_{smooth}, $SAPS_{thresh}$) = (1.3, 0.8, 0.01, 0.05, -0.1). For the SAPS/NR algorithm, a greedy initialisation was used where each variable was initialised to $\top$, $\bot$ or RandSelect($\{\top, \bot\}$) if there were more occurrences of the positive literal, negative literal, or neither, respectively. The performance results for all three algorithms after the first stage of the competition were nearly identical. RSAPS and SAPS/NR each solved 27 series and 143 instances, while SAPS solved 26 series and 144 instances. We chose SAPS to continue onward into the final rounds of the competition.

3 Ongoing Research

Since the introduction of the SAPS algorithm, our focus has been on understanding the behaviour of SAPS and other DLS algorithms as opposed to improving the performance of the algorithm.

```
procedure SAPS(F, α, ρ, wp, P_smooth, SAPS_thresh)
    input:
        propositional formula F, scaling factor α,
        smoothing factor ρ, random walk probability wp,
        smoothing probability P_smooth,
        SAPS threshold SAPS_thresh
    output:
        variable assignment A

    for i := 1..|A| do a(i) := RandSelect({⊤, ⊥})
    for j := 1..|CLP| do clp(j) := 1
    while (F is unsatisfied under A) do
        curScore := Eval(F, A, CLP)
        bestScore := ∞
        BestVars := ∅
        for each i s.t. variablei 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 ∪ {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. clausej is unsatisfied under A do
                    clp(j) := clp(j) × α
                end for
                with probability P_smooth do
                    for j := 1..|CLP| do
                        clp(j) := clp(j) + (1 − ρ) × clp‾
                    end for
                end with
            end with
        end if
    end while
    return (A)
end procedure SAPS
```

Figure 1: The SAPS algorithm. For each $clausej$ 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 $clausej$ 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.

When studying the performance of SAPS when applied to the unweighted MAX-SAT problem [5] we made some interesting observations. For most of the regular SAT instances we had tested, the optimal value of α was not significantly different from the default value of 1.3. However, we found that for slightly overconstrained MAX-SAT instances, the optimal value of α was *significantly* smaller, and that optimal values were closer to 1.05. For heavily overconstrained instances, the optimal value was even smaller, and closer to 1.01. This suggests that for SAPS and other DLS algorithms, a reactive scheme for updating the scaling parameter α could be very effective.

We set out to investigate the behaviour of DLS algorithms and answer the question if DLS algorithms were warping their search landscapes in an intelligent manner in [6]. We found no evidence that the clause penalties of SAPS were making the warped landscapes any easier to search, which suggests that DLS algorithms are behaving quite differently than expected based on some of the original intuitions underlying these algorithms. In the same study, we also investigated the importance of random decisions in SAPS, and found that for SAPS/NR the random initialisation of variables is sufficient to achieve the full variance in runtime observed by other state-of-the-art stochastic local search algorithms.

Recently, Thornton *et al.* developed the Pure Additive Weighting Scheme (PAWS) algorithm and found that PAWS can outperform SAPS on larger, more difficult instances [4]. With these more recent developments, and our increasingly better understanding of DLS algorithms and their behaviour, we are now in an excellent position for developing the next generation of DLS algorithm for SAT.

References

[1] 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.

[2] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *LNCS 2470: Proc. of the Eighth Int'l Conf. on Principles and Practice of Constraint Programming (CP-02)*, pages 233–248, 2002.

[3] D. Schuurmans, F. Southey, and R. C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proc. of the Seventeenth Int'l Joint Conf. on Artificial Intelligence (IJCAI-01)*, pages 334–341, 2001.

[4] J. Thornton, D. N. Pham, S. Bain, and V. Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proc. of the Ninteenth Nat'l Conf. on Artificial Intelligence (AAAI-04)*, 2004 (to appear).

[5] D. A. D. Tompkins and H. H. Hoos. Scaling and probabilistic smoothing: Dynamic local search for unweighted MAX-SAT. In *LNAI 2671: Proc. of the 16th Conf. of the Canadian Society for Computational Studies of Intelligence (AI-2003)*, pages 145–159, 2003.

[6] D. A. D. Tompkins and H. H. Hoos. Warped landscapes and random acts of SAT solving. In *Proc. of the Eighth Int'l Symposium on Artificial Intelligence and Mathematics (ISAIM-04)*, 2004.

[7] D. A. D. Tompkins and H. H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Seventh Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT2004)*, 2004 (this volume).