Scaling and Probabilistic Smoothing: Dynamic Local Search for Unweighted MAX-SAT

Dave A. D. Tompkins[1] and Holger H. Hoos[2⋆]

[1] Department of Electrical Engineering
[2] Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada
davet@ece.ubc.ca,hoos@cs.ubc.ca
http://www.cs.ubc.ca/labs/beta

Abstract. In this paper, we study the behaviour of the Scaling and Probabilistic Smoothing (SAPS) dynamic local search algorithm on the unweighted MAX-SAT problem. MAX-SAT is a conceptually simple combinatorial problem of substantial theoretical and practical interest; many application-relevant problems, including scheduling problems or most probable explanation finding in Bayes nets, can be encoded and solved as MAX-SAT. This paper is a natural extension of our previous work, where we introduced SAPS, and demonstrated that it is amongst the state-of-the-art local search algorithms for solvable SAT problem instances. We present results showing that SAPS is also very effective at finding optimal solutions for unsatisfiable MAX-SAT instances, and in many cases performs better than state-of-the-art MAX-SAT algorithms, such as the Guided Local Search algorithm by Mills and Tsang [8]. With the exception of some configuration parameters, we found that SAPS did not require any changes to efficiently solve unweighted MAX-SAT instances. For solving weighted MAX-SAT instances, a modified SAPS algorithm will be necessary, and we provide some thoughts on this topic of future research.

1 Introduction and Background

The propositional satisfiability problem (SAT) is an important subject of study in many areas of computer science. Since SAT is $\mathcal{NP}$-complete, there is little hope to develop a complete algorithm that scales well on all types of problem instances; however, fast algorithms are needed to solve large problems from various domains. As with most other work on SAT algorithms, we consider only propositional formulae in conjunctive normal form (CNF), *i.e.*, formulae of the form $F = \bigwedge_i \bigvee_j l_{ij}$, where each l_{ij} is a propositional variable or its negation. The l_{ij} are called *literals*, while the disjunctions $\bigvee_j l_{ij}$ are called *clauses* of F.

Unweighted MAX-SAT is the optimisation variant of SAT in which the goal is, given a CNF formula F, to find an assignment of truth values to the propositional variables in F that maximises the number of satisfied clauses. MAX-SAT is a conceptually simple $\mathcal{NP}$-hard combinatorial problem of substantial theoretical and practical interest; many

⋆ To whom correspondence should be addressed.

application-relevant problems, including scheduling problems or Most Probable Explanation (MPE) finding in Bayes nets can be encoded and solved as MAX-SAT [10]. Although MAX-SAT is defined as a maximisation problem, it is usually more convenient to consider the corresponding minimisation problem, where the goal is to *minimise* the number of *unsatisfied* clauses. In *weighted MAX-SAT*, each clause c_i is associated with a weight w_i, and the goal is to minimise the total weight of the unsatisfied clauses. Obviously, unweighted MAX-SAT is equivalent to weighted MAX-SAT with each clause weight equal to one ($\forall i : w_i = 1$).

Some of the best known methods for solving SAT are Stochastic Local Search (SLS) algorithms; these are typically incomplete, *i.e.*, they cannot determine with certainty that a formula is unsatisfiable but they often find models of satisfiable formulae surprisingly effectively [6]. Although SLS algorithms for SAT differ in their details, the general search strategy is mostly the same. Starting from an initial, complete assignment of truth values to all variables in the given formula F, in each search step, the truth assignment of one variable is changed from true to false or vice versa; this type of search step is also called a *variable flip*. Variable flips are typically performed with the purpose of minimising an objective function that sums the number of unsatisfied clauses (in the case of weighted MAX-SAT, this would easily generalise into minimising the total weight of the unsatisfied clauses). Since the introduction of GSAT [15], a simple best-improvement search algorithm for SAT, much research has been conducted in this area. Major performance improvements were achieved by the usage of noise strategies [13], the development of the WalkSAT architecture [14], and further advancements such as the Novelty$^+$ variant of WalkSAT [5].

In parallel to the development of more refined versions of randomised iterative improvement strategies such as WalkSAT, another SLS method has become increasingly popular in SAT solving. This method is based on the idea of modifying the evaluation function in order to prevent the search from getting stuck in local minima or other attractive non-solution areas of the underlying search space. We call this approach *Dynamic Local Search (DLS)*. DLS strategies for SAT typically associate a clause penalty with each clause of the given formula, which is modified during the search process. These algorithms then try to minimise the total penalty rather than the number of the unsatisfied clauses. GSAT with clause penalties [13] was one of the first algorithms based on this idea, although it changes penalties only in connection with restarting the search process. Many variants of this scheme have been proposed: Frank [3] uses a DLS penalty scheme that is updated every time a variable is flipped. Morris' Breakout Method [9] simply adds one to the penalty of every unsatisfied clause whenever a local minimum is encountered. The Discrete Lagrangian Method (DLM) [18] is based on a tabu search procedure and uses a similar, but slightly more complicated penalty update scheme. Additionally, DLM periodically and deterministically invokes a smoothing mechanism that decreases all clause penalties by a constant amount. The Smoothed Descent and Flood (SDF) approach [11] introduced a more complex smoothing method, and the concept of multiplicative penalty updates, which evolved into the Exponentiated Sub-Gradient (ESG) method [12]. Our Scaling and Probabilistic Smoothing (SAPS) [7] method improved upon the ESG approach; SAPS will be described in detail in Section 2.

With the SAT problem, both complete solvers and SLS solvers have had a large amount of success. However, complete solvers often have difficulty with MAX-SAT problems, whereas SLS methods have been extremely effective. For MAX-SAT, much emphasis has been placed on developing polynomial-time algorithms that can achieve solutions within a bounded factor of the optimal solution, but in practice these algorithms are not as effective as SLS approaches. In principle, any SLS algorithm for SAT can be applied to the unweighted MAX-SAT problem with some simple modifications, but it is not clear that algorithms effective in the SAT domain are also effective in the MAX-SAT domain. Furthermore, in many cases the straightforward extensions of SLS algorithms for SAT to weighted MAX-SAT problems appear to perform rather poorly.

One of the first SLS techniques applied to MAX-SAT was the Steepest Ascent Mildest Descend (SAMD) algorithm [4], and numerous approaches have been used since. The Iterated Local Search (ILS) algorithm by Yagiura and Ibaraki [19] (ILS-YI) is an effective MAX-SAT solver, and is different from many approaches in that it implements a multi-flip neighbourhood. The Guided Local Search (GLS) approach of Mills and Tsang (GLSSAT2) [8] is currently considered one of the best-performing algorithms for MAX-SAT. In this paper, we apply SAPS, our recently developed, state-of-the-art DLS algorithm for SAT, to the unweighted MAX-SAT problem. Our empirical performance results show that for a wide range of problem instances, SAPS finds quasi-optimal (*i.e.*, provably optimal or best known) solutions significantly faster than the state-of-the-art GLSSAT2 algorithm. This suggests that extensions of SAPS to weighted MAX-SAT might also reach or exceed state-of-the-art performance.

The remainder of this paper is structured as follows. In Section 2 we review the SAPS algorithm and discuss some of its important characteristics. In Section 3, we report and discuss the results from our empirical study of SAPS on MAX-SAT. In Section 4 we discuss how SAPS can be extended from unweighted to weighted MAX-SAT. Finally, Section 5 contains conclusions and points out directions for future work.

2 Scaling and Probabilistic Smoothing

In this section, we describe the Scaling and Probabilistic Smoothing (SAPS) algorithm [7]. SAPS is a Dynamic Local Search (DLS) algorithm, developed as a variant of the ESG algorithm of Schuurmans *et al.* [12].

Like most DLS algorithms, SAPS associates a clause penalty clp_i with each clause i, which is dynamically changed throughout the search process.[3] The clause penalties help to direct the search, ideally away from local minima and toward a global optimum. There are two distinct stages involved in updating the clause penalties: a *scaling stage* and a *smoothing stage*. In the scaling stage, all currently unsatisfied clause penalties are multiplied by a scaling factor α. In the smoothing stage, all penalties are adjusted toward the clause penalty mean $\overline{clp}$ according to a smoothing factor ρ. In SAPS, outside of a local minimum the best flip candidate is always chosen, with ties broken randomly. Whenever a local minimum is encountered, a random walk step is taken with

[3] To avoid potential confusion between the clause weights in Weighted MAX-SAT and the clause weights used by DLS algorithms, we strictly refer to the latter as *clause penalties*. This differs from the terminology we used in [7].

procedure UpdatePenalties(F, x, CLP, α, ρ,P_{smooth})
  input:
    propositional formula F, variable assignment x, clause penalties $CLP = (clp_i)$,
    scaling factor α, smoothing factor ρ, smoothing probability P_{smooth}
  output:
    clause penalties CLP
  C = {clauses of F}
  U_c = $\{c \in C \mid c$ is unsatisfied under $x\}$
  for each i s.t. $c_i \in U_c$ **do**
    $clp_i := clp_i \times \alpha$
  end
  with probability P_{smooth} **do**
    for each i s.t. $c_i \in C$ **do**
      $clp_i := clp_i \times \rho + (1 - \rho) \times \overline{clp}$
    end
  end
  return (CLP)
end

Fig. 1. The SAPS penalty update procedure; $\overline{clp}$ is the average over all clause penalties.

probability wp (by flipping a variable that has been selected uniformly at random from the set of all variables of F), otherwise scaling is performed, after which a smoothing stage is executed with probability P_{smooth}. While the SAPS algorithm described in [7] only performs probabilistic smoothing at local minima, recent experiments (not reported here) demonstrate that the smoothing stage can be performed outside of local minima (completely decoupled from the scaling stage) which provides for more robust values of P_{smooth}. Recent experiments (not reported here) also suggest that the random walk step is more effective when performed with probability wp outside of local minima. However, the performance enhancement is marginal, and for consistency we used the original SAPS algorithm for the experiments conducted in the context of this study.

In Figures 1 and 2 we provide the details of our SAPS algorithm. Figure 1 shows the algorithm outline of our penalty update procedure. Compared to ESG and other DLS algorithms with frequent smoothing, the time complexity incurred by the smoothing stage of SAPS is considerably reduced, since in each local optimum, smoothing is only performed with a probability P_{smooth}. Figure 2 shows the main SAPS algorithm and its underlying search procedure with penalties; overall, the main algorithm is conceptually very similar to ESG. The only modification to the SAPS algorithm for SAT required in the context of this work was the addition of a simple mechanism by which the best candidate solution found within a given run is stored and returned when the algorithm terminates.

The behaviour and performance of the SAPS algorithm depend on four parameters: α, ρ, P_{smooth}, and wp. For satisfiable instances, the performance of SAPS *w.r.t.* these parameters is very robust, and setting the values of (α,P_{smooth},wp) to (1.3,0.05,0.01) will provide nearly optimal results for most instances. To adjust for differences in the smoothing parameter ρ, we developed a reactive variant of SAPS (RSAPS) that reactively changes the value of ρ when search stagnation is detected [7].

```
procedure SAPS(F, α, ρ, wp, P_smooth)
    input:
        propositional formula F, scaling factor α,
        smoothing factor ρ, random walk probability wp,
        smoothing probability P_smooth
    output:
        variable assignment x
    x := Init(F)
    x̂ := x
    CLP := InitPenalties(F)
    while not terminate(F, x) do
        x' := PenalisedSearchStep(F, x, CLP)
        if x' = ∅ then
            with probability wp do
                x := RandomStep(F, x)
            otherwise
                CLP := UpdatePenalties(F, x, CLP, α, ρ, P_smooth)
            end
        else
            x := x'
        end
        if f(F, x) < f(F, x̂) then
            x̂ := x
        end
    end
    return (x̂)
end

procedure PenalisedSearchStep(F, x, CLP)
    input:
        propositional formula F, variable assignment x, clause penalties CLP
    output:
        variable assignment x̂ or ∅
    U_v = {variables of F that appear in clauses unsatisfied under x}
    X' := {x̂ | x̂ is x with variable v ∈ U_v flipped}
    best :=min{g(F, x̂, CLP) | x̂ ∈ X'}
    X := {x̂ ∈ X' | g(F, x̂, CLP) = best}
    if best ≥ 0 then
        x̂ := ∅
    else
        x̂ := draw(X)
    end
    return (x̂)
end
```

Fig. 2. The SAPS Algorithm for (unweighted) MAX-SAT. 'Init' generates a random variable assignment x, 'InitPenalties' initialises all clause penalties to 1. 'RandomStep(F,x)' returns an assignment obtained from x by flipping a variable that has been selected uniformly at random from the set of all variables of F; $f(F, \hat{x})$ and $g(F, \hat{x}, CLP)$ denote the number and the total penalty of the clauses in F that are unsatisfied under assignment $\hat{x}$, respectively. The function 'draw(X)' returns an element that is selected uniformly at random from set X.

3 Experimental Design and Results

To evaluate the performance of SAPS on the unweighted MAX-SAT problem, we conducted extensive experiments on several sets of well-known benchmark instances, in addition to some newly developed test sets of MAX-SAT instances that were designed to test the performance of MAX-SAT algorithms in more detail:

jnh The set of instances known as the DIMACS `jnh` set have been popular benchmark instances, and are available at SATLIB (`www.satlib.org`). These instances are generated with n variables such that each variable is added to each clause with probability $1/n$. The literals are negated with probability $1/2$ and all unit and empty clauses are removed. Instances generated in this manner are often referred to as Random P-SAT [16] or constant density model instances. Several of the `jnh` instances are satisfiable.

rnd$n$-m**u** We used several test sets of overconstrained (unsatisfiable) Uniform Random 3-SAT instances in our experiments. These instances are obtained by randomly and independently generating m clauses of length three as follows: The respective three literals are selected uniformly at random from the set of all possible literals over the given n variables; clauses that contain more than one literal with the same variable are discarded [16].

bor-k**u** These test sets consist of satisfiable and unsatisfiable MAX-2-SAT and MAX-3-SAT instances (both weighted and unweighted) and have been described and used in [1]. For each problem size and clause length, there are only 2–9 instances, and all instances are overconstrained. For our experiments, we only ran experiments on the unsatisfiable unweighted instances, which we grouped together in one MAX-2-SAT and one MAX-3-SAT test set.

rndu1000a These instances are unweighted variants of instances from a set originally used by Yagiura and Ibaraki [20].[4] They were generated in a fashion similar to the `jnh` instances and have 1000 variables and 7700 clauses each.

In total, we used 789 unique instances in our empirical evaluation of SAPS for unweighted MAX-SAT. All experiments were conducted on IBM servers with dual 2GHz Intel Xeon processors (hyperthreading disabled for accurate time results) with 512KB CPU cache and 4GB RAM, running Red Hat Linux 7.3. For each problem instance and algorithm, we measured empirical run-length and run-time distributions (RLDs and RTDs) [6] based on at least 100 successful runs, in each of which the algorithm was run until the respective quasi-optimal solution quality was reached. Run-times are reported in terms of the medians of the corresponding RLDs (measured in search steps) or RTDs (measured in CPU seconds). For the SAPS experiments, no instance-specific parameter tuning was conducted; the parameters (α,ρ,P_{smooth},wp) were set to (1.05,0.8,0.05,0.01) as generally good values, except for the heavily overconstrained instances, where we used $\alpha = 1.01$. For the GLSSAT2 software, the default settings were used (except for cutoff), and for the ILS-YI software, the neighbourhood size was set to 2.

[4] The respective weighted test set is available online at: `http://www-or.amp.i.kyoto-u.ac.jp/members/yagiura/benchmarks.html`.

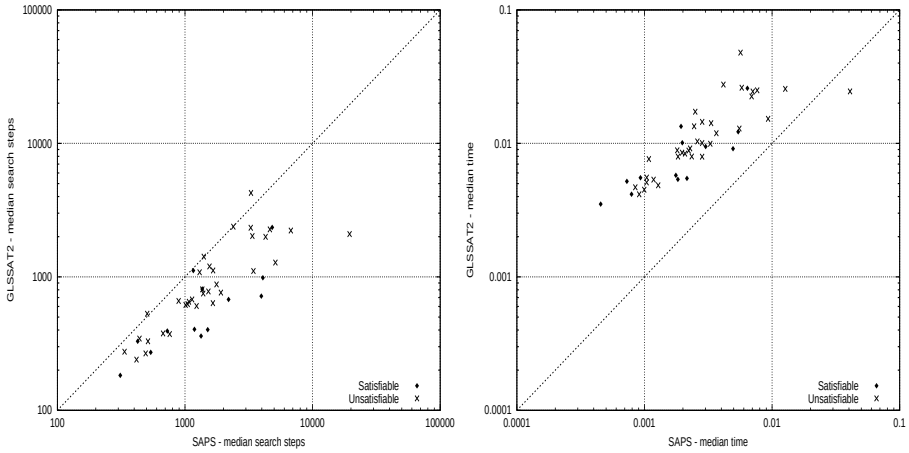

Fig. 3. Comparison between search step (left) and run-time (right) performance of SAPS and GLSSAT2 on the `jnh` test set. Every point correspond to the median performance over 100 runs of each algorithm on a single instance. Run-time is measured in CPU seconds.

In our empirical study, we generally measured the run-time and number of search steps for reaching provably optimal or best-known solution qualities. Where possible, Borcher and Furman's complete solver `maxsat` [1] was used to determine the optimal solution quality for each instance. (This is the best-performing complete MAX-SAT solver we are aware of.) For larger instances, which become quickly intractable for this solver, we used *quasi-optimal* solution qualities obtained from Iterated Robust Tabu Search, a state-of-the-art SLS algorithm for MAX-SAT, in conjunction with an "iterative deepening" scheme (for details, see [17], where exactly the same solution qualities are used). In all cases, the solution qualities thus obtained could not be improved upon with any method we are aware of (including SAPS), and in several cases their optimality was confirmed by the complete `maxsat` solver (which took several CPU days).

As we will demonstrate in Table 1, GLSSAT2 consistently outperforms ILS-YI, and so we compare the performance of SAPS primarily with that of GLSSAT2. Our first comparison between SAPS and GLSSAT2 is shown in Figure 3, where we compare the search step performance and time performance of the two algorithms on the `jnh` problem instances. Each data point in the figure represents a single problem instance, where points above and to the left of the diagonal represent instances where SAPS outperformed GLSSAT2. From Figure 3 (left) it is clear that GLSSAT2 generally requires fewer search steps than SAPS for finding quasi-optimal solutions. However, comparing the search step results to the time performance results in Figure 3 (right) clearly illustrates one of the key features of SAPS: fast search steps. When the comparison is made for time performance, SAPS outperforms GLSSAT2 by a median speedup factor *(s.f.)* of 3.9.

In Figure 4, we present similar results for the `rnd100-500u`, `rnd150-750u`, and `rnd200-1000u` test sets. All of these sets have the same clauses/variables ratio

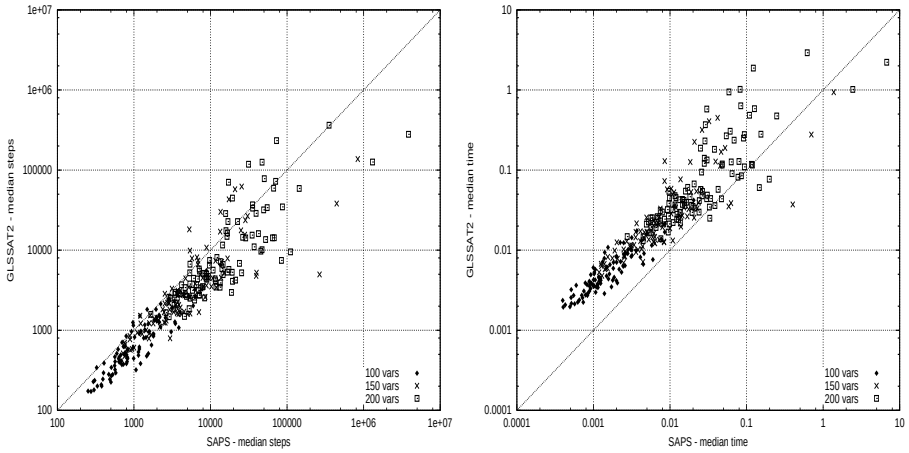

Fig. 4. Comparison between search step (left) and run-time (right) performance of SAPS and GLSSAT2 on the instances from the test sets `rnd100-500u`, `rnd150-750u`, and `rnd200-1000u`. All points correspond to the median performance for at least 100 runs of a single instance. Time is in CPU seconds.

of 5, so we would expect to see similar differences in performance between the two algorithms. Considering that the solubility phase transition for Uniform Random-3-SAT occurs at a clauses/variables ratio near 4.3 [2], these instances are slightly overconstrained. The results in Figure 4 (left) and (right) closely resemble the results presented in Figure 3; for most instances, GLSSAT2 requires fewer search steps for finding quasi-optimal solutions, but when taking into account the CPU time per search step, SAPS typically performs better than GLSSAT2.

When analyzing the run-time performance of an SLS algorithm, it is important to study the distribution of the run-times on individual problem instances [6]. In Figure 5 (left) we present the RLDs for the ILS-YI, GLSSAT2 and SAPS algorithms on a medium hardness instance from the set `rnd100-500u`. From this figure, it can be seen that GLSSAT2 consistently outperforms SAPS when measuring run-time in search steps, and SAPS outperforms ILS-YI; the RLDs for all three algorithms are closely approximated by exponential distributions, which is known to be typical for high-performance SLS algorithms for SAT when using optimal or close-to-optimal parameter settings [6]. This picture changes for the respective RTDs shown in Figure 5 (right); clearly, SAPS performs best when measuring CPU time, which again highlights the significant difference between search step performance and run-time performance.

It is interesting to note that the optimal solutions of the instances in the slightly overconstrained test set `rnd100-500u` have between 1 and 5 unsatisfied clauses, with an average of 2.8 unsatisfied clauses per instance. This is in marked contrast to the sets of heavily overconstrained instances with a clauses/variables ratio of 10; *e.g.*, for the instances from test set `rnd100-1000u`, quasi-optimal solutions have between 24 and 36 unsatisfied clauses, with an average of 32.2 unsatisfied clauses per instance.

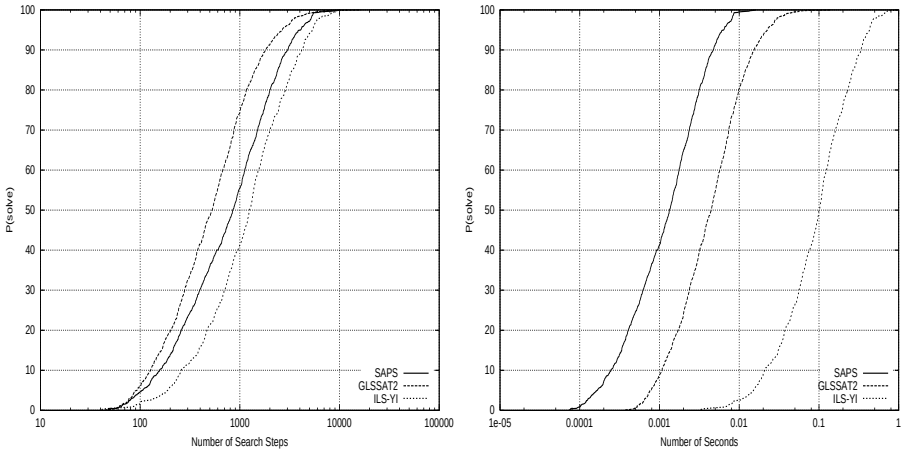

Fig. 5. RLD (left) and RTD (right) of SAPS, GLSSAT2 and ILS-YI on instance `uuf-100.500-017` (medium hardness) from test-set `rnd100-500u`. Run-time is measured in CPU seconds. Parameters for SAPS(α,ρ,P_{smooth},wp) are (1.05,0.8,0.05,0.01), GLSSAT2 is with default settings, and ILS-YI is with default settings, using the 2-flip neighbourhood.

While for all other instances used in this study, SAPS performed well using $\alpha = 1.05$, for these heavily overconstrained instances, a much smaller value of α is required for achieving good performance of SAPS (as previously stated, we use $\alpha = 1.01$). The significance of this difference in α will be addressed later in this Section. In Figure 6 we present a comparison of GLSSAT2 and SAPS performance on heavily overconstrained instances. From this figure, it is clear that SAPS outperforms GLSSAT2 for these test sets in terms of both search steps (left) and CPU time (right) required for finding quasi-optimal solutions.

In Table 1 we summarise the results from our experiments on all test sets used in this study. For each problem instance, at least 100 runs of each algorithm were performed, in each of which a quasi-optimal solution was found. From the RLDs and RLDs thus obtained, we determined the median number of search steps and median CPU time for finding a quasi-optimal solution of the respective instance; the values shown in the table are the medians of these search cost measures over the respective test-sets. To indicate the variability in instance hardness within each set, we also report the ratio of the 90% and 10% quantiles (q.90/q.10) of the instance search cost over the respective test set (measured in search steps). To help further illustrate the excellent time performance of SAPS, the speedup factor for the median instance is given, in addition to the fraction of instances for which the SAPS algorithm was best in terms of CPU time required for finding a quasi-optimal solution of a given instance.

For the heavily overconstrained instances, ILS-YI shows impressive search step performance, but as mentioned previously, when considering the time complexity of the search steps, this performance advantage is not amortised. GLSSAT2 does not perform well on the heavily overconstrained instances, but overall it shows a very impressive

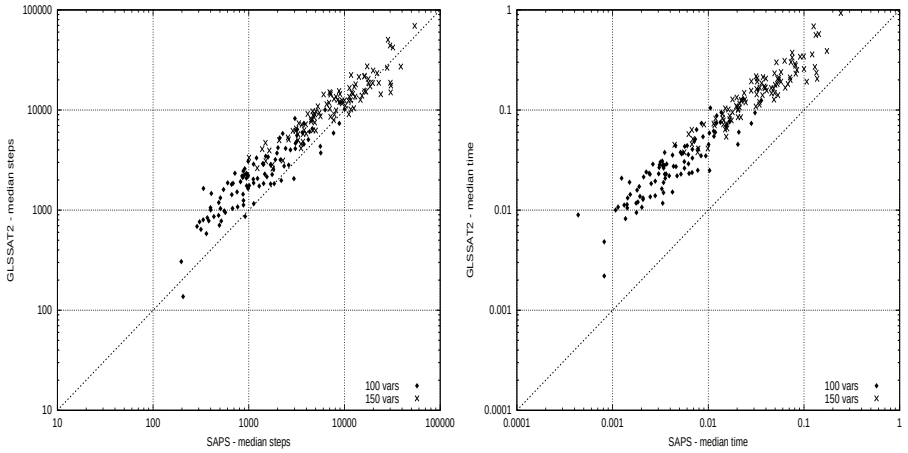

Fig. 6. Comparison between search step (left) and run-time (right) performance of SAPS and GLSSAT2 on the instances from test sets `rnd100-1000u` and `rnd150-1500u`. Every point corresponds to the median performance over 100 runs on a single instance. CPU time is measured in seconds.

search step performance. However, the search steps in GLSSAT2 are expensive, and compared to the search steps of SAPS, they are slow.

Finally, we found that the `bor-`k`u` instances were very easy to solve for all of the algorithms tested here and that once again SAPS showed the best time performance. Furthermore, the most impressive median time speedup of SAPS over GLSSAT2 was found for the `rndu1000a` test set; for the same test set, even after 10 million search steps, less than 50% of the runs of the ILS-YI algorithm reached a quasi-optimal solution quality.

Overall, it is interesting to note that in order to achieve the reported excellent performance of SAPS, the only parameter that needed to be adjusted for the various test-sets was the scaling factor, α. When SAPS was tested on satisfiable instances, a parameter setting of $\alpha = 1.3$ was found to generally give good performance. On most satisfiable instances we tested, near-optimal performance of SAPS was obtained with that setting. However, initial experiments with SAPS on unweighted MAX-SAT with $\alpha = 1.3$ produced discouraging performance results; subsequently, we discovered that a lower value of α was required for unsatisfied, and particularly for heavily overconstrained instances.

Intuitively, this phenomenon can be explained as follows: The obvious difference between satisfiable and unsatisfiable instances is that in the latter case, any optimal solution leaves some clauses unsatisfied. It is reasonable to assume that for unsatisfiable instances, there are sub-optimal local minima in the objective function (number of unsatisfied clauses) that share unsatisfied clauses with the global optima. Clearly, when SAPS encounters such a local minimum, the penalties of these clauses will be increased along with those of all other unsatisfied clauses, which allows the search process to avoid stagnation in this area of the search space. This, however, may easily have

Problem Set	ILS-YI			GLSSAT2			SAPS 1.0					
	steps	time	$\frac{q.90}{q.10}$	steps	time	$\frac{q.90}{q.10}$	α	steps	time	$\frac{q.90}{q.10}$	*f.b.*	*s.f.*
jnh	3,037	419.8	24.1	**751**	9.5	8.5	1.05	1,391	**2.4**	11.6	0.94	3.9
rnd100-500u	1,398	108.7	8.9	**563**	4.5	7.5	1.05	929	**1.2**	8.0	1.00	3.6
rnd125-625u	3,879	302.8	24.2	**1,329**	10.6	12.5	1.05	2,264	**3.3**	17.1	0.94	3.2
rnd150-750u	7,674	607.6	51.5	**2,552**	19.4	21.5	1.05	4,127	**6.4**	18.9	0.95	3.0
rnd175-875u	20,029	1,514.6	120.8	**4,119**	33.1	28.1	1.05	8,920	**15.2**	21.0	0.92	2.2
rnd200-1000u	31,968	2,440.8	29.7	**5,301**	44.2	23.5	1.05	13,343	**21.1**	18.3	0.91	2.1
rnd100-1000u	**884**	133.6	6.1	2,119	27.2	7.4	1.01	1,115	**3.9**	9.9	1.00	7.0
rnd150-1500u	**3,237**	499.7	15.5	11,035	148.1	4.8	1.01	7,723	**34.2**	10.0	1.00	4.3
bor-2u	76	5.6	18.1	88	1.1	14.1	1.05	73	**0.1**	71.2	0.80	7.7
bor-3u	740	65.3	32.0	425	4.7	30.9	1.05	487	**1.1**	39.3	1.00	4.5
rndu1000a	—	—	—	**20,812**	832.4	6.5	1.05	27,434	**67.4**	7.8	0.90	12.3

Table 1. Performance comparison of ILS-YI, GLSSAT2, and SAPS over a range of unweighted MAX-SAT test sets. The precise meaning of the step and time performance is explained in the text, CPU times are measured in CPU milliseconds. The speedup factor *s.f.* shows the improvement in time performance of SAPS over GLSSAT2. The "fraction of best" measure *f.b.* indicates the fraction of instances within the respective test-sets for which SAPS performs better than GLSSAT2 in terms of CPU time. For SAPS, default parameters settings of (ρ,P_{smooth},wp)= $(0.8, 0.05, 0.01)$ were used; GLSSAT2 annd ILS-YI were used with their respective default settings.

the side effect of modifying the search space around the optimal solution(s) in a way that makes it harder for SAPS to efficiently reach an optimal solution. (Note that although it might be desirable to change the evaluation function in such a way that only the current local minimum is eliminated, changes of the clause penalties used by a DLS algorithm for SAT or MAX-SAT will usually affect large areas of the search space.)

For this reason, we would expect that aggressive scaling (*i.e.*, high α settings) becomes more detrimental to the efficiency of SAPS as the number of unsatisfied clauses, and hence the potential undesired side effects of scaling, increases. By using smaller α settings for more heavily overconstrained MAX-SAT instances, the impact of each single scaling stage, and hence presumably the magnitude of the unwanted side effects, is reduced, resulting in improved performance of SAPS, as observed in our empirical study.

On the other hand, when we examined the results (not shown) for different instances *within* the same test set, we found no correlation between the number of unsatisfied clauses in the respective optimal solution qualities and the optimal value of α. However, we did find strong evidence suggesting that within a test set, *harder* instances are solved faster when using a *smaller* value of α. As can be seen in Figure 6, the performance ratio between SAPS and GLSSAT2 appears to slightly decrease with instance hardness within the test set. We found that this effect can be avoided by decreasing the scaling factor α as the instance hardness increases. This could indicate that the hardness differences of MAX-SAT instance sampled from the same random distribution are

partially due to the way in which global and local optima are coupled in terms of shared unsatisfied clauses.

Clearly, it would be highly desirable to shed further light on these issues by analysing the composition and distance of local minima in relation to the optimal solution, and by characterising the changes in the search landscape induced by the dynamic adjustment of the clause penalties in SAPS; both of these directions are currently being followed in our ongoing research. Furthermore, based on our observations and reasoning presented here, it would seem that in contrast to the reactive mechanism used in RSAPS [7] for automatically adjusting the smoothing parameter ρ, an adaption mechanism for the scaling factor α might be more beneficial to SAPS for MAX-SAT; devising and studying such mechanisms constitutes another direction for further research.

4 Initial Thoughts on SAPS for Weighted MAX-SAT

Applying the SAPS algorithm to unweighted MAX-SAT is a necessary stepping stone to developing an effective DLS algorithm for weighted MAX-SAT. When moving from unweighted to weighted MAX-SAT, DLS algorithms have to contend with an interesting issue: how should the fixed clause weights in a weighted MAX-SAT instance interact with the dynamically changing clause penalties? To date, three different DLS algorithms for MAX-SAT have used three very different approaches. In Shang and Wah's first DLM algorithm for weighted MAX-SAT [18], an evaluation function is used which sums the clause weights w_i and the clause penalties clp_i over all unsatisfied clauses. A variant of this algorithm known as DLM-99-SAT uses clause penalties only, but these are initialised to the clause weights, and during the search, the clause penalties are modified proportionately to the respective clause weights. Finally, the GLSSAT algorithm for weighted MAX-SAT defines a utility function as $clp_i/1 + w_i$ to determine which penalty weights are updated, but it does not use the clause weights directly to guide the search.

In addition to the three methods described above, a variety of other approaches can be used for incorporating clause weights and penalties into an extension of SAPS for weighted MAX-SAT. Limited preliminary results suggest that at least some of the resulting SAPS variants appear to perform at least as good as GLSSAT2 on standard benchmarks for weighted MAX-SAT, but a better understanding of the search dynamics and a more thorough empirical analysis is required to confirm these results.

5 Conclusions & Future Work

In this study, we applied the Scaling and Probabilistic Smoothing (SAPS) algorithm to the unweighted MAX-SAT problem; it extends our previous work, in which we developed the SAPS algorithm for the SAT problem, and established it as a state-of-the-art SLS algorithm for SAT. Here, we presented empirical evidence that SAPS performs similarly excellent on unweighted MAX-SAT, where it consistently outperforms GLSSAT2, one of the best performing MAX-SAT algorithms known to-date. For all of the problem test sets we examined, SAPS was at least 2 times faster than GLSSAT2 in finding optimal or best known solutions, and for some instances SAPS was over 10

times faster. We found that the performance of SAPS was relatively robust *w.r.t.* parameter settings; only for heavily overconstrained problem instances, a different setting of the scaling parameter was required to reach state-of-the-art performance. We provided some insight into this behaviour and some of the other phenomena we encountered.

This work provides a solid foundation for assessing and understanding the behaviour of SAPS on MAX-SAT, and provides a natural stepping stone for developing a SAPS variant for weighted MAX-SAT. Since SAPS is a Dynamic Local Search (DLS) algorithm, and uses dynamic clause penalties, there are many different ways of combining the clause penalties with the clause weights specified in a weighted MAX-SAT instance. Some of these variants are currently being implemented and empirically evaluated. When we proposed SAPS for SAT, we found it helpful to reactively tune the smoothing component of the algorithm, but from our work presented here, we found that unsatisfiable instances are more sensitive to the scaling component of the algorithm. Hence, we will turn our attention to developing a SAPS variant that dynamically adapts the scaling parameter during the search. Overall, based on the promising results reported here, we believe that SAPS has the potential to exceed the performance of the best-performing algorithms for MAX-SAT (weighted and unweighted) currently known.

Acknowledgements. This work has been supported by an NSERC Postgraduate Scholarship (PGS-B) to DT and by HH's NSERC Individual Research Grant #238788. We thank Patrick Mills and Edward Tsang for providing us with their GLS implementation.

References

1. B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. In *Journal of Combinatorial Optimization, Vol. 2*, pp. 299–306, 1999.
2. P. Cheeseman, B. Kanefsky and W.M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91*, pp.331–337, 1997.
3. J. Frank. Learning Short-term Clause Weights for GSAT. In *Proc. IJCAI-97*, pp. 384–389, Morgan Kaufmann Publishers, 1997.
4. P. Hansen and B. Jaumard. Algorithms for the maximum Satisfiability problem. In *Computing*, 44:279-303, 1990.
5. H.H. Hoos. On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT. In *Proc. AAAI-99*, pp. 661–666. AAAI Press, 1999.
6. H.H. Hoos and T. Stützle. Local Search Algorithms for SAT: An Empirical Evaluation. In *J. of Automated Reasoning*, Vol. 24, No. 4, pp. 421–481, 2000.
7. F. Hutter, D.A.D. Tompkins, and H.H. Hoos. Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT. In *LNCS 2470:Proc. CP-02*, pp. 233–248, Springer Verlag, 2002.
8. P. Mills and E.P.K. Tsang. Guided Local Search for solving SAT and weighted MAX-SAT problems. In *Journal of Automated Reasoning, Special Issue on Satisfiability Problems*, pp. 24:205–223, Kluwer, 2000
9. P. Morris. The breakout method for escaping from local minima. In *Proc. AAAI-93*, pp. 40–45. AAAI Press, 1993.

10. J.D. Park. Using Weighted MAX-SAT Engines to Solve MPE. In *Proc. AAAI-02*, pp. 682–687. AAAI Press, 2002.
11. D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. In *Proc. AAAI-2000*, pp. 297–302, AAAI Press, 2000.
12. D. Schuurmans, F. Southey, and R.C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proc. IJCAI-01*, pp. 334-341, Morgan Kaufmann Publishers, 2001.
13. B. Selman and H.A. Kautz. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proc. IJCAI-93*, pp. 290–295, Morgan Kaufmann Publishers, 1993.
14. B. Selman, H.A. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In *Proc. AAAI-94*, pp. 337–343, AAAI Press, 1994.
15. B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proc. AAAI-92*, pp. 440–446, AAAI Press, 1992.
16. B. Selman, D.G. Mitchell, and H.J. Levesque. Generating Hard Satisfiability Problems. In *Artificial Intelligence, Vol. 81*. pp. 17–29, 1996.
17. K. Smyth, H.H. Hoos, and T. Stützle. Iterated Robust Tabu Search for MAX-SAT. In *Proc. of the 16th Canadian Conference on Artificial Intelligence (AI 2003)*, to appear, 2003.
18. Z. Wu and B.W. Wah. An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. In *Proc. AAAI-00*, pp . 310–315, AAAI Press, 2000.
19. M. Yagiura and T. Ibaraki. Analyses on the 2 and 3-Flip Neighborhoods for the MAX SAT. In *Journal of Combinatorial Optimization, Vol. 3, No. 1*, pp. 95-114, July 1999.
20. M. Yagiura and T. Ibaraki. Efficient 2 and 3-Flip Neighborhood Search Algorithms for the MAX SAT: Experimental Evaluation. In *Journal of Heuristics, Vol. 7, No. 5*, pp. 423-442, 2001.