Weight Redistribution for Unweighted MAX-SAT

Abdelraouf Ishtaiwi[1,2], John Thornton[1,2], and Abdul Sattar[1,2]

[1] IIIS, Griffith University, QLD, Australia
[2] DisPRR, National ICT Australia Ltd, QLD, Australia
{a.ishtaiwi,j.thornton,a.sattar}@griffith.edu.au

Abstract. Many real-world problems are over-constrained and require search techniques adapted to optimising cost functions rather than searching for consistency. This makes the MAX-SAT problem an important area of research for the satisfiability (SAT) community. In this study we perform an empirical analysis of several of the best performing SAT local search techniques in the domain of unweighted MAX-SAT. In particular, we test two of the most recently developed SAT clause weight redistribution algorithms, DDFW and DDFW$^+$, against three more well-known techniques (RSAPS, AdaptNovelty$^+$ and PAWS). Based on an empirical study across a range of previously studied problems we conclude that DDFW is the most promising algorithm in terms of robust average performance.

1 Introduction

Since the development of GSAT [1] in 1992, there has been considerable interest and progress in developing stochastic local search (SLS) techniques for solving propositional satisfiability (SAT) problems. While the SAT problem has many important applications, it is the case that many, if not most, real-world problems are *over-constrained.* If we express such problems using the SAT formalism, then the search task is no longer to find whether a satisfying assignment exists, but to find an assignment that maximises the number of true clauses (or equivalently minimises the number of false clauses). This problem is known as the unweighted MAX-SAT problem and has many applications in such areas as scheduling and the processing of Bayesian networks [2].

2 Local Search for Unweighted MAX-SAT

Current state-of-the-art SLS techniques can be divided into two main categories: WalkSAT-based heuristics and dynamic local search (DLS) clause weighting heuristics. Of the WalkSAT techniques, AdaptNovelty$^+$ [3] is recognised as one of the most robust and effective SAT solvers for situations where manual parameter tuning is impractical. By default, the Novelty heuristic [4] will take the best cost flip within a randomly selected false clause, unless this flips the clauses's

most recently flipped variable. In that case, Novelty probabilistically chooses between taking the clauses's best and second best cost flip according to a *noise* parameter. The adapt aspect of AdaptNovelty$^+$ automatically controls this noise parameter according to the level of search stagnation, i.e. the more search steps that have elapsed since the last improving move, the greater the noise and hence the probability of selecting the second best flip in a clause. Finally, the $^+$ aspect of AdaptNovelty$^+$ introduces a small probability of taking a randomly selected flip, regardless of cost, to avoid certain deterministic behaviours identified in [5].

Dynamic local search (DLS) SAT algorithms take the approach of adding and manipulating clause weights to provide guidance and avoid stagnation during a search. This means the cost of a flip is calculated by summing the *weights* of the false clauses rather than simply counting their number. Typically, clause weighting involves a two stage process: firstly weight is added to all false clauses when the search becomes trapped in a local minimum (i.e. when it runs out of cost reducing moves). This increases the cost of violating just those clauses that are false at that point in the search. Secondly, clause weights are periodically reduced to avoid the immediate effects of a clause weight increase being swamped by the accumulated weight of past increases. The two older clause weight algorithms used in the current study (PAWS [6] and RSAPS [7]) differ mainly in using an additive weight scheme (PAWS) which increases and decreases weight in single integer units, as opposed to using a multiplicative scheme (RSAPS) that increases and decreases weight according to two real valued multipliers. RSAPS also borrows the adaptive mechanism from AdaptNovelty to adjust its weight reducing multiplier.

2.1 Divide and Distribute Fixed Weight

More recently, a new approach to clause weighting has been proposed, termed divide and distribute fixed weight (DDFW) [8]. DDFW introduces two ideas into the area of clause weighting algorithms for SAT. Firstly, it evenly distributes a fixed quantity of weight across all clauses at the start of the search, and escapes local minima by transferring weight from satisfied to unsatisfied clauses in a single step (rather than using the two phase approach of SAPS and PAWS). Secondly, DDFW exploits neighbourhood relationships between clauses when deciding which pairs of clauses will exchange weight (see [8] for details). For the current MAX-SAT study we additionally changed DDFW so that it keeps track of the current best cost and the current best solution (see lines 3-5 and 9-12 in Algorithm 1).[3]

DDFW comes in two versions: DDFW and DDFW$^+$. DDFW$^+$ uses an adaptive mechanism to alter the total amount of weight distributed according to the degree of stagnation in the search. The original DDFW initialises the weight of each clause to W_{init} (fixed at 8 in the current study), whereas DDFW$^+$ sets this value at 2 and adapts it during the search as follows: if the search executes a consecutive series of i flips without reducing the total number of false clauses,

[3] The same changes were made to all the SAT algorithms appearing in the study.

Algorithm 1 DDFW-MAX-SAT($\mathcal{F}$, $MaxTime$, W_{init}, C_{target}, C_{best}, A_{best})

```
1: A ← a randomly generated truth assignment of F
2: set the weight w_i for each clause c_i ∈ F to W_init
3: set C_best ← number of false clauses in A
4: set A_best ← A
5: while time < MaxTime and C_best > C_target do
6:    select list L of literals causing the greatest reduction in weighted cost Δw when flipped
7:    if (Δw < 0) or (Δw = 0 and probability ≤ 15%) then
8:       update A by flipping a literal randomly selected from L
9:       if (number of false clauses in A < C_best) then
10:         set C_best ← number of false clauses in A
11:         set A_best ← A
12:      end if
13:   else
14:      for each false clause c_f do
15:         select a satisfied same sign neighbouring clause c_k with maximum weight w_k
16:         if w_k < W_init then
17:            randomly select a clause c_k with weight w_k ≥ W_init
18:         end if
19:         if w_k > W_init then
20:            transfer a weight of 2 from c_k to c_f
21:         else
22:            transfer a weight of 1 from c_k to c_f
23:         end if
24:      end for
25:   end if
26: end while
```

where i is equal to the number of literals in the problem, then the amount of weight on each clause is increased by 1 in the first instance. However, if after increasing weights, the search enters another consecutive series of i flips without improvement, then it will reset the weight on each satisfied clause back to 2 and on each false clause back to 3. The search then continues to follow each increase with a reset and each reset with an increase. In this way a long period of stagnation will produce oscillating phases of weight increase and reduction, such that the total weight can never exceed $3\times$ the total number of clauses plus the total number of false clauses (for more detail see [8]).

3 Experimental Study

Initial empirical evaluations of DDFW and DDFW$^+$ on a range of structured and unstructured SAT problems have shown both algorithms can outperform AdaptNovelty$^+$ and RSAPS, and that DDFW$^+$ has a better average performance than DDFW [8]. However, a similar study has yet to be attempted in the MAX-SAT domain. To this end we decided to compare both DDFW variants with AdaptNovelty$^+$, RSAPS and PAWS10 (PAWS10 is a fixed parameter version of PAWS with the Max_{inc} parameter set at 10). Our first criteria was that manual parameter tuning is not allowed. This practical consideration means that the best clause weighting algorithms must be limited to using fixed parameter values (as in PAWS10) or to using an adaptive mechanism (as in RSAPS). We further chose AdaptNovelty$^+$ as our example WalkSAT algorithm because it has performed consistently well in the last three SAT competitions (see http://www.satcompetition.org).

To evaluate the relative performance of these algorithms we followed the strategy used to compare SAPS with iterated local search and GLSSAT2 in [2] and divided our empirical study into three problem categories:

bor problem set: the bor-ku problem set comprises of a range of unweighted unsatisfiable MAX-2-SAT (bor-2) or MAX-3-SAT (bor-3) instances used in [2] and first described by Borcher in [9]. We used all 17 bor-2 and 11 bor-3 unsatisfiable unweighted instances from this set.

jnh problem set: the jnh problems are generated with n variables. Each variable is added to a clause with probability $1/n$ and each literal is negated with probability $1/2$ (unit and empty clauses are then deleted). We used 33 unsatisfiable jnh instances taken from the SATLIB library (www.satlib.org) each with 100 variables and between 800 and 900 clauses.

***uuf problem set*:** the uuf problems are unsatisfiable uniform random 3-SAT instances generated with n variables and k clauses with each clause containing 3 literals randomly chosen from the $2n$ possible literals. Tautological and duplicate literals are eliminated. We used the same 49 unsatisfiable instances reported in [2] each containing 200 variables and 1000 clauses.

All experiments were performed on a Dell machine with 3.1GHz CPU and 1GB memory. Cut-offs were set as follows: first RSAPS and DDFW were given a trial on each problem with a flip cut-off of 10,000,000. The best cost reached by either algorithm was then recorded and used as a target cut-off for further runs. All algorithms were allowed 100 trials for each instance and terminated either upon reaching the target cost (C_{target} in Algorithm 1) or after 3 seconds per trial.

***bor* Problem Results**: Taken in combination, the graphs in Figure 1 (a and b) and the statistics in Table 1 show DDFW to have the best overall performance on both the bor-2 and bor-3 problem classes. In the bor-2 graph (Figure 1a) DDFW and RSAPS clearly outperform the other techniques with DDFW having a slight advantage. The bor-2 table results help to separate DDFW and RSAPS further by showing DDFW not only to be twice as fast as RSAPS on average, but also to be consistently better than RSAPS on a problem by problem basis (as shown by the 88.24% Winner value). Interestingly, DDFW$^+$ performed quite poorly on this problem class, indicating there may be something in the two-literal clause structure of the bor-2 problems that causes the heuristic to be counter-productive. The bor-3 graph results in Figure 1b are less conclusive, as all algorithms were able to achieve near 100% success, although DDFW and DDFW$^+$ both exhibit superior performance during the first 0.04 seconds. Again, the table results help to clarify that DDFW is the better algorithm, showing it to have the best performance on 68.68% of instances, with DDFW$^+$ coming a close second in terms of average time.

***jnh* Problem Results**: Both the graph in Figure 1c and the table results show a mixed picture for the jnh problems. In the graph, AdaptNovelty$^+$ and DDFW$^+$ emerge as equal best performers in the first 0.1 seconds of execution but are

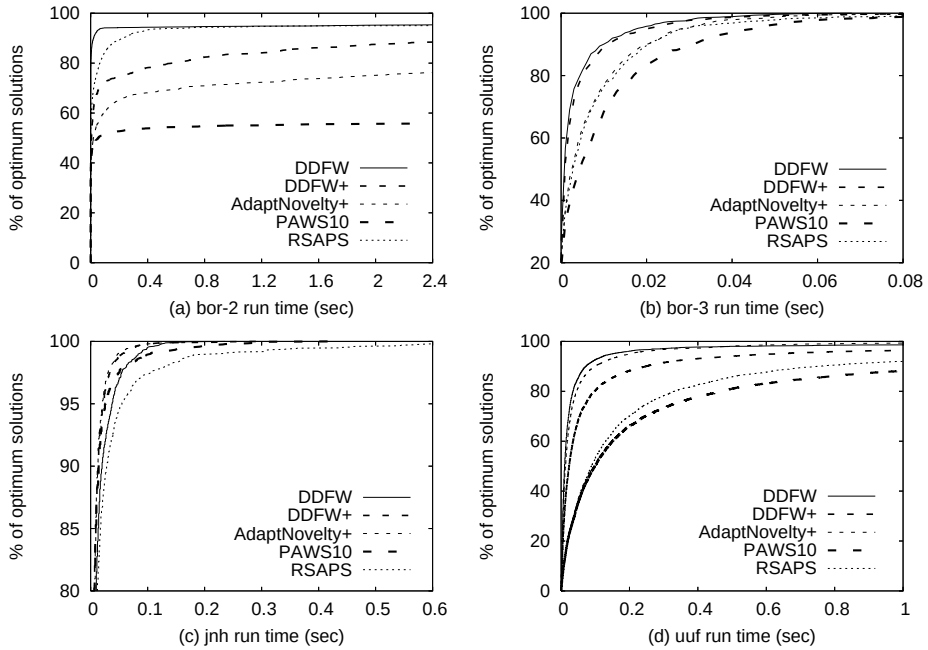

Fig. 1. Graphs of comparative time performance for all problem sets (% optimum refers to the proportion of solutions that reached the target solution cost).

joined by DDFW after 0.2 seconds, with PAWS10 starting well but trailing off and RSAPS coming last. In the table, both AdaptNovelty$^+$ and DDFW$^+$ achieve similar results, with AdaptNovelty$^+$ having a slight advantage in average time, but DDFW$^+$ doing better on flips and the % Winner value. The surprise is that PAWS10 achieves the best result on 48.48% of jnh instances, despite being beaten in terms of average time. This can be explained by PAWS10 taking a longer than average on the more difficult jnh instances. Overall, therefore, there is little to chose between DDFW$^+$ and AdaptNovelty$^+$ on these problems, although they both do better than DDFW, PAWS10 and RSAPS.

***uuf* Problem Results**: The uuf results are similar to the bor-2 results, only this time DDFW has a very similar curve in Figure 1d to AdaptNovelty$^+$ (rather than RSAPS). In this case DDFW starts well but is caught by AdaptNovelty$^+$ after 0.5 seconds, after which AdaptNovelty$^+$ has a slight advantage. This is confirmed by the slightly better uuf success rate for AdaptNovelty$^+$ in Table 1. However, the table also shows DDFW to have the better average time and to have achieved the best result on 67.35% of the instances. Taking this into consideration, we can conclude that DDFW has the better uuf performance, with AdaptNovelty$^+$ coming a close second, followed by DDFW$^+$.

Problem	Method	Mean Flips	Mean Seconds	% Target	% Winner
bor-2	DDFW	**7,719.62**	**0.01777**	**95.41**	**88.24**
	DDFW$^+$	79,308.65	0.15745	88.59	0.00
	AdaptNovelty$^+$	125,590.40	0.19005	77.24	0.00
	RSAPS	28,943.91	0.04536	95.06	11.76
	PAWS10	28,688.04	0.05131	55.82	0.00
bor-3	DDFW	**1,890.02**	**0.00385**	**100.00**	**63.63**
	DDFW$^+$	2,156.29	0.00440	99.91	18.18
	AdaptNovelty$^+$	4,449.40	0.00717	99.73	9.09
	RSAPS	4,798.05	0.00793	99.91	9.09
	PAWS10	6,109.10	0.01093	99.91	0.00
jnh	DDFW	3,721.65	0.00833	100.00	6.06
	DDFW$^+$	**2,376.13**	0.00553	100.00	21.21
	AdaptNovelty$^+$	2,870.73	**0.00546**	100.00	18.18
	RSAPS	7,724.26	0.01489	100.00	6.06
	PAWS10	3,747.89	0.00732	100.00	**48.48**
uuf	DDFW	**15,453.06**	**0.03294**	98.80	**67.35**
	DDFW$^+$	35,161.40	0.07335	96.98	4.08
	AdaptNovelty$^+$	32,441.44	0.04965	**99.92**	28.57
	RSAPS	97,412.31	0.16765	93.80	0.00
	PAWS10	167,275.80	0.30023	95.76	0.00

Table 1. Averaged results for each problem set; the time and flip statistics were measured over runs that reached the target solution cost; % Target measures the proportion of runs that reached the target solution cost; % Winner measures the proportion of problems for which an algorithm obtained the best result.

3.1 Analysis

Overall, the results show that DDFW has the best performance on our chosen problem set. More specifically, DDFW has the better performance on three out of four of the problem classes, emerging as a clear winner on the bor-2 and bor-3 problem sets, slightly better than AdaptNovelty$^+$ on the uuf problems but defeated by AdaptNovelty$^+$ and DDFW$^+$ on the jnh problems. DDFW's closest rivals are AdaptNovelty$^+$ and DDFW$^+$, with AdaptNovelty$^+$ defeating DDFW$^+$ on the uuf problems, equalling it on the jnh problems and being defeated by DDFW$^+$ on the bor-2 and bor-3 problems.

These results partly confirm the initial evaluation of DDFW and DDFW$^+$ in the SAT domain [8]. There, both algorithms were shown to have better performance in comparison to several other state-of-the-art SAT algorithms (including AdaptNovelty$^+$ and RSAPS) on a range of SAT competition benchmark problems. However, it was concluded for SAT that the DDFW$^+$ heuristic was a generally helpful addition to DDFW, causing superior performance in several problem categories while remaining neutral in the others. In contrast, our MAX-SAT study has found the DDFW$^+$ heuristic to have a negative effect on performance on the bor-2, bor-3 and uuf problems, only helping for the jnh

problems. This brings the general applicability of the DDFW$^+$ heuristic under question. There is certainly no sign that it is useful for MAX-SAT problems and it could be that the promising results in the SAT domain were produced simply by a fortuitous selection of problems. We leave answering this question as a matter for further research.

4 Conclusions

The main contribution of this paper is to have evaluated and extended the weight redistribution strategy of the DDFW heuristic to the domain of unweighted MAX-SAT problems. In the process we have shown that DDFW outperforms several class-leading local search algorithms on a range of previously studied MAX-SAT benchmark problems. These results strengthen the case for the use of weight redistribution in clause weighting, as opposed to using the two-stage increase/reduce strategies of PAWS and SAPS. As with earlier studies, DDFW's performance has been shown to be more robust than RSAPS or a fixed parameter PAWS, making it the most promising approach for situations where manual parameter tuning is impractical.

Acknowledgements: We thankfully acknowledge the financial support from NICTA and the Queensland government. NICTA is funded by the Australian Government's *Backing Australia's Ability* initiative, and in part through the Australian Research Council.

References

1. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: Proc. 10th National Conference on AI (AAAI-92). (1992) 440–446
2. Tompkins, D.A., Hoos, H.H.: Scaling and probabilistic smoothing: Dynamic local search for unweighted MAX-SAT. In: Proc. 16th Conference of the Canadian Society for Computational Studies of Intelligence (AI-2003). (2003) 145–159
3. Hoos, H.H.: An adaptive noise mechanism for WalkSAT. In: Proc. 18th National Conference on AI (AAAI-02). (2002) 635–660
4. McAllester, D.A., Selman, B., Kautz, H.A.: Evidence for invariants in local search. In: Proc. 14th National Conference on AI (AAAI-97). (1997) 321–326
5. Hoos, H.H.: On the run-time behaviour of stochastic local search algorithms for SAT. In: Proc. 16th National Conference on AI (AAAI-99). (1999) 661–666
6. Thornton, J.: Clause weighting local search for SAT. Journal of Automated Reasoning **35**(1-3) (2005) 97–142
7. Hutter, F., Tompkins, D.A., Hoos, H.H.: Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Proc. 8th International Conference on Principles and Practice of Constraint Programming (CP-02). (2002) 233–248
8. Ishtaiwi, A., Thornton, J., Anbulagan, Sattar, A., Pham, D.N.: Adaptive clause weight redistribution. In: Proc. 20th International Conference on Principles and Practice of Constraint Programming (CP-06). (2006) 229–243
9. Borchers, B., Furman, J.: A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. Journal of Combinatorial Optimization **2** (1999) 299–306