Longer-Term Memory in Clause Weighting Local Search for SAT

Valnir Ferreira Jr. and John Thornton

Institute for Integrated and Intelligent Systems
Griffith University
PMB50 Gold Coast Mail Centre
QLD 9726
Email: [v.ferreira, j.thornton]@griffith.edu.au

Abstract. This paper presents a comparative study between a state-of-the-art clause weighting local search method for satisfiability testing and a variant modified to obtain longer-term memory from a global measure of clause perturbation. We present empirical evidence indicating that by learning which clauses are hardest to satisfy, the modified method can offer significant performance improvements for a well-known range of satisfiable problems. We conclude that our method's ability to learn, and consequently to offer performance improvement, can be attributed to its ability to obtain information from a global measure of hardness, rather than from the contextual perspective exploited in previous works.

1 Introduction

Local search methods have attracted substantial attention in the research community due to their ability to efficiently find solutions to satisfiability testing (SAT) problems that are too large for complete search methods. SAT problems are of significant practical and theoretical interest as many real-world applications like artificial intelligence reasoning, constraint satisfaction, and planning can be formulated in this way. The problem consists of finding an assignment for the boolean variables in a propositional formula that makes the formula true [2]. Local search methods for SAT work by iteratively modifying the value of one variable in the problem from true to false or vice-versa. These variable flips are typically performed so as to minimise an evaluation function that maps any given variable assignment x to the number of unsatisfied clauses under x. The methods follow this heuristic until a satisfying assignment is found (all clauses are satisfied) or until either a maximum run-time or number of flips is reached.

Clause weighting local search methods (CWLS) modify a basic local search by having individual weights assigned to all clauses in the SAT problem, thus dynamically changing the evaluation function and the search landscape as the search progresses. Since the introduction of weighted local search more than a decade ago [4, 6], several improvements have been proposed, the most relevant ones being the discrete Lagrangian method (DLM) [9], and SAPS [3], which at the time of its publication achieved state-of-the-art performance on a range of

benchmark SAT problems. Recently, the pure additive weighting scheme (PAWS) was introduced [7] and shown to give significant performance improvements over SAPS on a range of challenging well-known SAT problems from the SATLIB[1] and DIMACS[2] libraries, as well as on a set of SAT-encoded random binary CSPs from the phase transition region.

In this work, we hypothesise that the performance of clause weighting local search methods such as PAWS can be significantly enhanced by learning which clauses are globally hardest to satisfy, and by using this information to treat these clauses differentially. Our work is principally motivated by (a) the fact that there seems to be some speculation and little empirical results on this topic, and (b) the belief that we can improve the performance of CWLS methods for SAT by identifying and exploiting implicit longer-term clause dependencies.

2 Learning while Weighting

Several works have investigated whether clause weights in CWLS methods should be seen as useful information, and therefore be interpreted as learning how to better search SAT instances. These works propose that by dynamically modifying their weight profile while searching a given instance, CWLS methods are in fact rendering the underlying search space easier. This idea was used for many years as an explanation for the efficiency of these methods in general. The work on WGSAT [1] offered some better insights into this topic by concluding that clause weights can only offer information that is limited and contextual (i.e. related to the last few assignments), and should therefore be interpreted only as a source of short-term memory. As the search moves away from a particular context, such information is no longer relevant in the new context. This is a reason why all successful CWLS methods need efficient ways to adjust clause weights as the search progresses, as it is by doing so that they can maintain the weight profile relevant to the context in which they are searching. To this end, most methods can be divided into those that adjust their weight profiles multiplicatively, and those that do it additively.

Multiplicative methods use floating point clause weights and increase/decrease multipliers that combined give the weight profile a much finer granularity. Additive methods, on the other hand, assign integer values to clause weights and increase/decrease amounts, which results in a coarser clause weight distribution. Given that CWLS methods rely on their weight profiles for search guidance, and given the tightly coupled relationship between these weight profiles and the selection of candidate variables to flip (which ultimately impinges on a method's runtime), we currently believe that the efficiency of additive methods like PAWS can be partially explained by the fact that additive methods make less distinction between candidate costs and thus consider a larger domain of candidate moves [7]. One can see why it would be desirable to derive guidance from information that is of a long-term nature, rather than short-term, or contextual. Intuitively,

[1] http://www.satlib.org

[2] http://dimacs.rutgers.edu/Challenges/Seventh/PC

longer-term memory is desirable because it could lead to the development of better flipping heuristics that would take into account global information such as which clauses are hardest to satisfy or which clauses have been unsatisfied the most during the search. To our knowledge, such information is currently not explored by even the most efficient methods.

In an attempt to exploit longer-term memory for remembering and avoiding local minima, DLM was extended [10] with a *special increase* sub-procedure that picks a set C of clauses (the size of C is problem dependent and varies between the number of all false clauses and the number of clauses in the problem) and then computes the ratio between the maximum clause weight in C and the mean weight of all clauses in C. If this ratio is larger than a problem dependent threshold, then the weight of the clause with the maximum clause weight in C is increased by 1. Note that the special increase sub-procedure is called at the end of DLM's clause re-weighting stage, so it can be seen as the adding of an extra penalty to that single most heavily weighted clause. The use of special increases was shown to have flattened the distribution of the number of times clauses had to be re-weighted during the search, with the resulting algorithm showing improved performance over the original DLM for a range of hard satisfiable instances from the DIMACS library. The effect that the special increase has on the weight profile is interesting as, given the resulting better performance, it could point towards a correlation between a less rugged search space and the method's ability to find a solution more efficiently.

The usefulness of clause weights for learning how to better search SAT instances has been brought back into the spotlight recently in [8]. In this work, SAPS was run on a given problem until a solution was found, and the corresponding clause weights at that point were recorded. These weights were then used to generate the so-called weighted instances, i.e., instances where the weight of each clause is initialised to the value the clause had *at the end* of the preceding successful run, rather than to one. The authors then propose that if a method can find a solution to the weighted instance by performing less flips than it would for the corresponding unweighted instance, then the weights carried over from the previous run would be truly making a difference (i.e. making the instance easier). Note that this is the same as restarting the method while maintaining all clause weights unmodified. The authors then go on to say that "if all clause weights represent *knowledge* that the algorithm has accumulated about the search space, then the modified SAPS algorithm is starting with all knowledge *a priori*". Then, a set of relatively easy instances[3] was used to test this hypothesis, and the results demonstrated that there was no significant difference between the two methods, which led to the conclusion that there was no evidence to support the claim that the creation of modified landscapes by CWLS methods render instances any easier, and so the belief that this can be seen as a form of learning is incorrect and should be dismissed.

[3] Unweighted SAPS was able to solve 9 out of 10 instances in less than 200,000 flips; 8 of which it was able to solve in less than 35,000 flips.

The main criticism we level at this approach is that it tries to harness knowledge from clause weights in an unsound way. It uses the weight profile recorded at the time when the method found a solution to initialise the weights of clauses of a new instance where variables are randomly instantiated to values potentially different from the ones assigned to them when the weight profile was recorded. In our view, this type of weight usage is doomed to fail, as there is substantial evidence (for instance [1]) to support the fact that clause weights used in this way are context dependent, i.e., at any point in time during the search they will provide a short-term memory that only goes back a few instantiations (more or less, depending on the weight adjustment mechanism being used by the CWLS method). Alternatively, the heuristic we introduce and explain throughout the remainder of this paper is able to explore longer-term information derived from a global, rather than contextual, measure of clause perturbation that is available as a by-product of the weight update mechanism that is common to CWLS methods.

3 PAWS+US

PAWS is an additive CWLS method for SAT that achieves state-of-the-art performance on a range of hard satisfiable SAT problems. Figure 1 shows the augmented version of PAWS, modified to accommodate what we call the *usual suspects* heuristic (US). The usual suspects are the clauses that emerge from a run as the most heavily weighted, according to a cumulative list that logs the number of times each clause was weighted during the search. We call this modified method *PAWS+US* to distinguish it from the standard PAWS. The methods only differ in respect to the clause weight initialisation (lines 4-8) and weight update (lines 19-26) procedures. In all other respects, PAWS+US is identical to the standard PAWS presented in [7].

The method begins by randomly instantiating all variables in the problem. PAWS initialises all clause weights to 1, whereas PAWS+US initialises the weight of the US clauses to their special weight increment (inc), and the weight of other clauses to one. After this initialisation stage, the search begins and while a solution is not found and the search is not terminated (either by reaching a maximum predetermined time or number of flips) the method builds, at every iteration, a list L of candidate flips, where an element of L is a variable that, if flipped, would reduce the objective function the most (lines 10-16). Then, with probability 1-P_{flat}[4] it randomly selects and flips a variable from L, and with probability P_{flat} it takes a flat move, i.e., a variable flip that would leave the value of the objective function unchanged (lines 17-18). If no potential improvement is found (i.e. any variable flip would result in an increased value for the objective function, and hence the method has reached a local minimum), then a weight update is performed. At this stage PAWS adds one to the weight of every unsatisfied

[4] P_{flat} is one of two parameters of PAWS that determines the probability for a flat move. We found that P_{flat} can be treated as a constant, and its value was set at 15% for all experiments reported here.

clause, whereas PAWS+US adds a special weight increment value (inc) to the US clauses that are unsatisfied at this point and one to every other false clause (lines 20-22). After the weight update stage is finished, if $MaxInc$[5] consecutive weight increases have taken place, then a weight decrease is performed whereby every weighted clause ($c_j \mid w_j > 1$) has its weight subtracted by one.

```
1. procedure PAWS+US
2. begin
3.     generate random starting point
4.     for each clause c_i do:
5.         if PAWS then set clause weight w_i ← 1
6.         else if PAWS+US then set clause weight w_i ← inc_i
7.         end if
8.     end for
9.     while solution not found and not terminated do
10.        best ← ∞
11.        for each literal x_ij in each false clause f_i do
12.            Δw ←change Σw in false clause caused by flipping x_ij
13.            if Δw < best then L ← x_ij and best← Δw
14.            else if Δw = best then L = L ∪ x_ij
15.            end if
16.        end for
17.        if best < 0 or (best = 0 and probability ≤ P_flat) then
18.            randomly flip x_ij ∈ L
19.        else
20.            if PAWS then for each false clause f_i do: w_i ← w_i + 1
21.            else if PAWS+US then for each false clause f_i do: w_i ← w_i + inc_i
22.            end if
23.            if # times clause weights increased % MaxInc = 0 then
24.                for each clause c_j | w_j > 1 do: w_j ← w_j − 1
25.            end if
26.        end if
27.    end while
28. end
```

Fig. 1. The PAWS method with the US extension.

4 Empirical Study

The first part of our empirical study involved creating the lists of usual suspects for each of the 25 problems in our test set. In order to determine the US list for a problem, we ran PAWS 100 times with optimal values for $MaxInc$ while keeping a counter of the number of times each clause was weighted during the

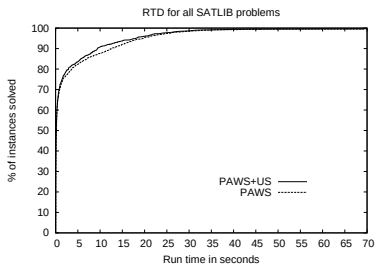
[5] $MaxInc$ is another parameter of PAWS used to determine the point in the search where a weight decrease will be performed.

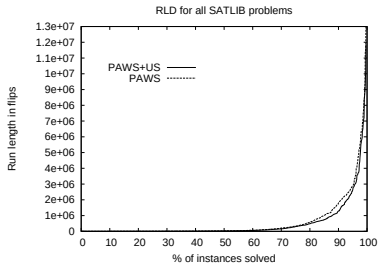
search (i.e. the clause was false at the time a weight increase was performed). We then obtained the mean number of weight increases for every clause over the 100 runs, and ordered the resulting list in descending order of weight increases. This resulting list is used to determine the US clauses for a run of PAWS+US. The method requires two extra parameters in addition to $MaxInc$, namely the usual suspects list length (LL) and the usual suspects weight increment (Inc). For this study we considered list lengths between 1 and 10^6 and weight increments between 2 and 5. For each of the 40 possible (Inc, LL) pairs for a problem, PAWS+US was run 100 times and the statistics for these runs were collected. For example, if PAWS+US(Inc:2,LL:5) is used on a problem p, the top 5 clauses from p's US list will have their weight incremented by 2 (instead of the standard weight increment of 1) every time a weight increase is performed and the clause is unsatisfied. Note here the important distinction between the US heuristic and the heuristic discussed above used with SAPS to investigate the usefulness of clause weights for learning how to better search SAT instances. The US lists provide global information of a longer-term nature that is used to treat the US clauses differently throughout the search every time a weight increase takes place whereas the approach previously discussed uses contextual information obtained at the end of one search to instantiate the weights of clauses in a subsequent search. There is no differential treatment of these clauses in any way. Therefore, in comparison, we can say that our approach first *learns* which clauses are typically hardest to satisfy and then uses this information to treat these clauses differentially.

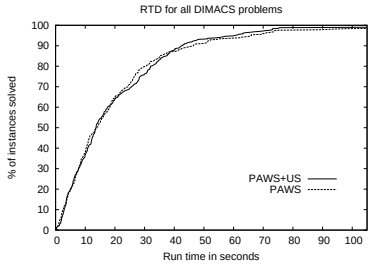
Our problem set originates from one of our previous studies [7] and is significantly diverse as it draws problems from four different domains. SATLIB's -med and -hard instances correspond to the median and hardest instances as found by a run of SAPS with optimal parameters on the respective original sets flat100, flat200, uf100 and uf250. From DIMACS we use the two most difficult graph colouring problems (g125.17 and g250.29) and the median and hardest 16-bit parity learning problems (par16-2-c and par16-3-c). For the random 3-SAT problems, we first generated 3 sets of problems (400, 800 and 1600 variable sets) from the 4.3 clause-to-variable ratio hard region. To these sets we added the f400, f800, and f1600 problems from DIMACS and repeated the procedure described above to determine the median and hardest instances, which resulted in the 6 random 3-SAT problems (f400, f800 and f1600 -med and -hard). Finally, a range of random binary CSPs (also from the accepted 4.3 ratio hard region) were generated and transformed into SAT instances using the multi-valued encoding procedure described in [5]. These problems were divided into 3 sets of 5 problems each according to the number of variables (v), the domain size (d) and the constraint density from the originating CSP (c), which resulted in the 30v10d40c, 30v10d80v and 50v15d80c problem sets from which the hardest of 5 problems was added to our problem set.

[6] Initially, we also tried list lengths consisting of the top 5 and top 10% most heavily weighted clauses but due to discouraging results we decided not to consider these list lengths further.

				Time (s)	Flips	Wilcoxon
Problem	Solver	Settings	Success (%)	mean median	mean median	t: time f: flips
SATLIB						
bw_large.a	PAWS	Max:34	100	0.05	3,107	
				0.05	2,533	
	PAWS+US	Inc:3,LL:5	100	0.04	2,330	*0.0000t
				0.04	1,805	*0.0170f
bw_large.b	PAWS	Max:50	100	0.40	47,001	
				0.29	31,620	
	PAWS+US	Inc:4,LL:2	100	0.29	31,448	*0.0212t
				0.22	22,474	*0.0314f
bw_large.c	PAWS	Max:5	100	10.23	1,245,757	
				7.53	888,581	
	PAWS+US	Inc:5,LL:2	100	6.29	720,611	*0.0003t
				5.11	569,357	*0.0001f
bw_large.d	PAWS	Max:4	100	14.85	1,369,449	0.2946t
				11.64	1,054,353	0.4211f
	PAWS+US	Inc:5,LL:3	100	14.75	1,314,033	
				11.05	1,050,479	
ais10	PAWS	Max:52	100	0.14	16,253	0.1609t
				0.10	10,220	0.1579f
	PAWS+US	Inc:5,LL:2	100	0.16	18,820	
				0.11	12,020	
flat100−med	PAWS	Max:16	100	0.05	10,498	
				0.04	6,162	
	PAWS+US	Inc:3,LL:1	100	0.05	8,991	0.1429t
				0.04	6,180	0.1633f
flat100−hard	PAWS	Max:46	100	0.13	38,252	
				0.09	26,326	
	PAWS+US	Inc:3,LL:4	100	0.09	27,032	*0.0117t
				0.07	18,682	*0.0180f
flat200−med	PAWS	Max:9	100	0.56	177,173	
				0.41	125,979	
	PAWS+US	Inc:3,LL:2	100	0.50	160,767	0.1813t
				0.38	116,965	0.1952f
flat200−hard	PAWS	Max:74	100	10.44	3,343,427	
				7.37	2,365,859	
	PAWS+US	Inc:2,LL:2	100	9.24	2,976,442	0.1979t
				6.54	2,097,565	0.2062f
uf100−hard	PAWS	Max:15	100	0.03	2,993	
				0.03	2,288	
	PAWS+US	Inc:2,LL:8	100	0.02	2,843	*0.0024t
				0.03	2,288	0.3680f
uf250−med	PAWS	Max:15	100	0.05	6,766	
				0.04	4,675	
	PAWS+US	Inc:2,LL:1	100	0.04	4,721	0.0632t
				0.04	3,686	*0.0220f
uf250−hard	PAWS	Max:18	100	1.26	316,767	
				0.80	200,491	
	PAWS+US	Inc:5,LL:9	100	1.11	261,294	0.4567t
				0.72	165,989	0.4040f
DIMACS						
g125.17	PAWS	Max:4	100	11.81	719,942	
				7.93	486,419	
	PAWS+US	Inc:3,LL:1	100	11.80	724,633	0.3484t
				7.01	424,301	0.3675f
g.250.29	PAWS	Max:4	100	32.71	382,086	0.3475t
				26.35	285,976	0.2520f
	PAWS+US	Inc:4,LL:2	100	31.93	383,378	
				27.59	325,129	
par16−3−c	PAWS	Max:40	97	14.46	3,928,952	
				10.97	2,980,095	
	PAWS+US	Inc:3,LL:5	100	13.88	3,816,071	0.2157t
				10.48	2,890,594	0.2694f
par16−2−c	PAWS	Max:36	99	16.15	4,421,398	0.0909t
				10.63	2,903,163	0.0919f
	PAWS+US	Inc:3,LL:1	97	17.66	4,820,909	
				12.04	3,280,032	







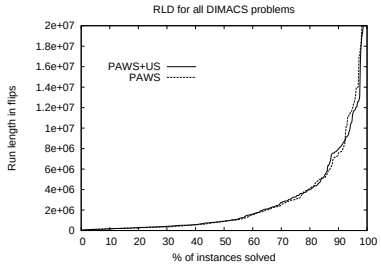


Table 1. Results for the SATLIB and DIMACS problems.

Local search run-times for the same problem can vary greatly due to different starting points and subsequent randomised decisions. For this reason, empirical studies have traditionally reported on statistics such as mean, median and standard deviation obtained from many runs on the same problem as a means to ascertain one algorithm's superiority over another. As the standard deviation is only informative for normally distributed data, and local search run-time and

				Time (s)	Flips	Wilcoxon
Problem	Solver	Settings	Success (%)	mean median	mean median	t: time f: flips
Random 3–SAT						
f400–med	PAWS	Max:9	100	0.19	42,862	
				0.14	29,140	
	PAWS+US	Inc:3,LL:4	100	0.12	23,526	*0.0001t
				0.08	12,869	*0.0001f
f400–hard	PAWS	Max:11	100	5.84	1,367,508	
				4.73	1,108,036	
	PAWS+US	Inc:3,LL:1	100	3.72	861,328	*0.0101t
				2.84	654,044	*0.0085f
f800–med	PAWS	Max:9	100	0.55	107,092	*0.0011t
				0.39	73,492	*0.0012f
	PAWS+US	Inc:3,LL:1	100	0.92	186,309	
				0.61	119,224	
f800–hard	PAWS	Max:10	100	4.07	912,512	
				2.50	529,681	
	PAWS+US	Inc:2,LL:1	100	4.11	870,033	0.4679t
				2.14	443,084	0.3833f
f1600–med	PAWS	Max:10	100	1.97	406,891	0.2800t
				1.51	273,914	0.3323f
	PAWS+US	Inc:3,LL:3	100	2.13	418,190	
				1.53	291,807	
f1600–hard	PAWS	Max:11	99	31.10	5,144,062	
				22.97	3,795,800	
	PAWS+US	Inc:2,LL:10	100	23.73	3,501,171	*0.0273t
				16.69	2,501,328	*0.0033f
SAT–encoded random binary CSPs						
30v10d40c	PAWS	Max:7	100	0.23	26,142	*0.0011t
				0.16	16,439	*0.0006f
	PAWS+US	Inc:2,LL:2	100	0.47	61,088	
				0.29	35,478	
30v10d80c	PAWS	Max:7	100	0.12	10,677	0.2217t
				0.10	8,164	0.2204f
	PAWS+US	Inc:3,LL:1	100	0.14	13,775	
				0.10	10,297	
50v15d80c	PAWS	Max:5	100	2.01	136,914	*0.0170t
				1.53	105,270	*0.0170f
	PAWS+US	Inc:4,LL:1	100	2.87	197,701	
				2.14	147,661	

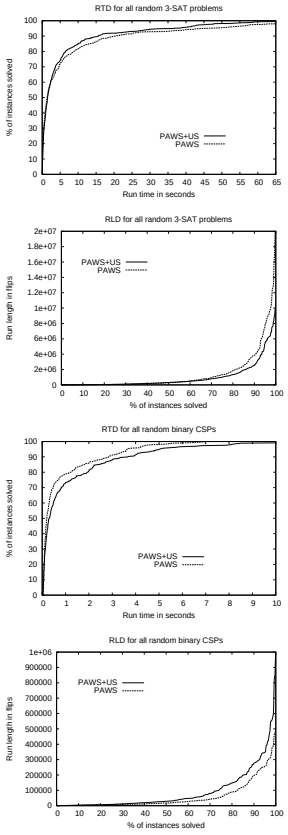


Table 2. Results for the random binary CSP and random 3-SAT problems.

run-length distributions are usually not normally distributed, we recently proposed that the non-parametric Wilcoxon rank-sum test be used to measure the confidence level of these assertions [7]. The test requires that the number of flips or run-times from two sets of observations A and B be sorted in ascending order, and that observations be ranked from 1 to n. Then, the sum of the ranks for distribution A is calculated and its value can be used to obtain, using the normal approximation to the Wilcoxon distribution, the z value that will give the probability P that the null hypothesis $H_0 : A \geq B$ is true. The Wilcoxon value in tables 1 and 2 give the probability P that the null hypothesis $A \geq B$ is true, where A is the distribution of the number of flips (or run-times) that has the smaller rank-sum value. We record the P value against distribution A, and take $P < 0.05$ to indicate with an asterisk that A is *significantly* less than B.

For all experiments, we set the maximum flip count to 20 million flips and the maximum time limit to infinity. All experiments were performed on a Sun supercomputer with 8 × Sun Fire V880 servers, each with 8 × UltraSPARC-III

900 MHz CPU and 8 GB memory per node. All statistics were derived from 100 runs of the algorithms.

We analyse our results from three different perspectives: the individual problem level, the problem domain level, and the overall level, where we consider a method's performance over the whole problem set. The results for PAWS were obtained using the the best known setting for $MaxInc$, whereas for PAWS+US we used the same $MaxInc$ and picked the optimal of 40 possible (Inc, LL) combinations based on a criteria of greatest completion rate, followed by the smallest mean number of flips. At the problem level, PAWS+US offers significantly better performance both in terms of run-time and number of flips for nine problems (uf100-hard time only, and uf250-med flips only). This means that for these nine problems the use of at least one (the optimal), and sometimes more (Inc, LL) pairs significantly improves PAWS's performance. PAWS, on the other hand, is significantly better for three of the problems.

When the analysis is taken to the problem domain level, PAWS+US is better (but not significantly) than PAWS for the random 3-SAT and SATLIB problems, whereas the reverse is true for the random binary CSPs, as demonstrated by the run-time and run-length distributions in table 2. By inspecting the distributions for the DIMACS problems in table 1, however, we can say with certainty that neither method dominates. Overall, PAWS's run-length distribution is slightly better than PAWS+US's, whereas it is not possible to determine either method's dominance in regards to the run-time distribution, as demonstrated in Figure 2.

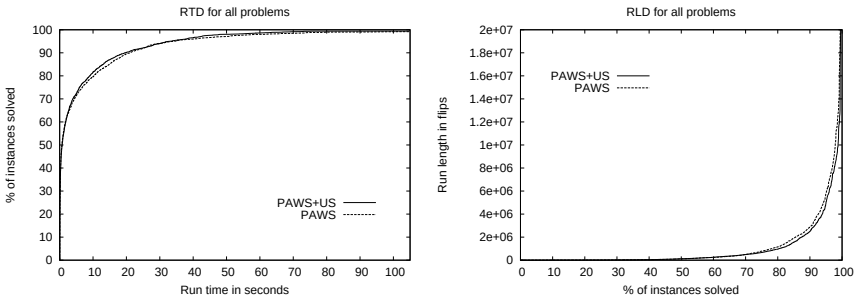


Fig. 2. The performance of PAWS and PAWS+US on the whole problem set, with optimal parameter settings for both methods as shown in tables 1 and 2.

For those problems where PAWS+US significantly outperformed PAWS, we observed that the improvement was generally not limited to optimal (Inc, LL) settings only, as was the case of problem bw_large.c for example, as shown in figure 3. This result, together with the others where PAWS+US gave significant improvements, can be interpreted as evidence that the use of the global knowledge afforded by the special treatment dispensed to the US clauses does

indeed render the instance easier for the CWLS method, irrespective of the rate of weight increase dictated by the setting of Inc.

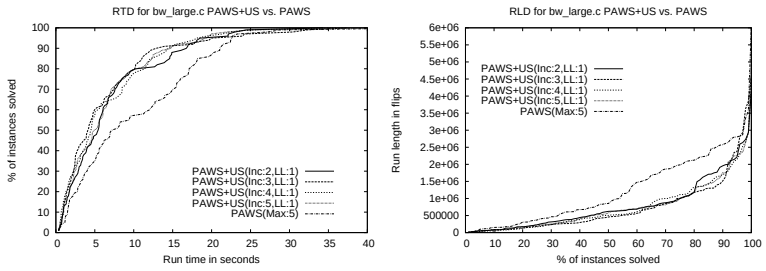


Fig. 3. RTD and RLD comparisons between PAWS($Max : 5$) and PAWS+US($Inc : \{2, ..., 5\}, LL : 1$) for the bw_large.c problem showing the method's stability *w.r.t.* the different settings of Inc.

We also decided to investigate the existence of an inter-parametric dependency between a (LL, Inc) pair and $MaxInc$. We re-ran our experiments, this time using 4 additional settings for $MaxInc$ (we used optimal $MaxInc$ ± 2), which allowed us to investigate $(4 \times 10 \times 5)$ Inc, LL and $MaxInc$ combinations. For all but three problems we observed that modifying the value of $MaxInc$ generally resulted in performance degradation, which indicates that the US heuristic generally works *in combination with* already optimally tuned values of $MaxInc$, and that its introduction does not create the need for re-tuning the host method's parameter. Two problems for which improvements were observed were flat100-med, and flat200-med, as for these we found at least one $(Inc, LL, MaxInc)$ triple that resulted in a reduction in the P value derived from using the Wilcoxon rank-sum test to less than 0.05, indicating that should these settings be used, the method would give a significant improvement over PAWS. However, we concluded that this improvement does not justify the expensive search required to find these triples. The third problem, f1600-hard, was the exception to the rule as most triples with a $MaxInc = 10$ (instead of PAWS's optimal setting of 11) resulted in significant improvements over the already significantly better performance obtained by PAWS+US($Inc : 2, LL : 10$). Furthermore, we found that the setting of $MaxInc = 10$ only works well in combination with the US and not in isolation. Figure 4 is used to illustrate these findings.

As previously mentioned, and according to observations not reported here, we found that generally PAWS+US gives the best performance when combined with the best known setting for $MaxInc$, settings which were found in [7] by testing 72 distinct values between 3 and 75. For this study, finding the optimal (Inc, LL) pair for PAWS+US involved searching on a space of (4×10) possible combinations, and then using this optimal pair in combination with the best setting for $MaxInc$. Therefore, in practice, the cost of tuning PAWS+US

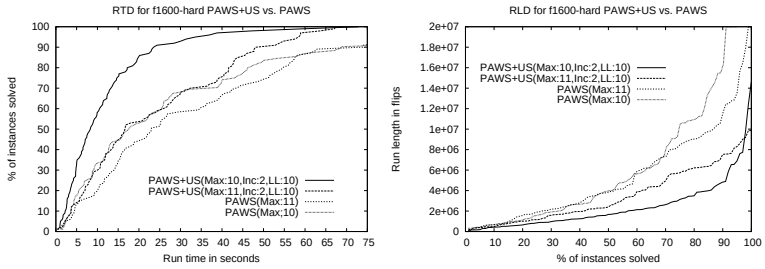


Fig. 4. RTD and RLD comparisons between PAWS($Max : \{10, 11\}$) and PAWS+US($Max : \{10, 11\}, Inc : 2, LL : 10$) for the f1600-hard problem, where a one step change in the weight decrease parameter resulted in yet another significant performance increase by PAWS+US.

is equivalent to searching on the space of the 40 possible combinations. This compares positively against most multiplicative CWLS methods such as SAPS, where the tuning of highly sensitive multipliers typically requires searching on a larger space of possible combinations[7].

5 Conclusion

Our results challenge the conclusions reached in previous works [8], which stated that no meaningful information could be derived from clause weights in CWLS methods. These works attempted to acquire meaningful information by examining clause weights at the end of a search, despite existing evidence [1] that information acquired in this fashion is of limited applicability due to its contextual nature. Furthermore, these methods did not consider using the acquired information to alter the way clauses are treated.

In contrast, we propose that our results support our hypothesis that CWLS methods can learn from clause weights, and that the significant performance improvement offered by PAWS+US can be attributed to (a)its ability to learn which clauses are globally hardest to satisfy, and (b) its ability to use this information to treat these clauses differentially.

As is the case with most empirical evaluations of local search methods, we found that our heuristic can offer performance improvements at the problem level, but this advantage tends to disappear as we shift the perspective to the overall level and consider all problems in combination. However, given that PAWS+US offered significant improvements over the state-of-the-art performance for approximately 40% of the problems in our test set, we believe that these initial results represent a well-founded motivation for future work. One interesting research path is the development of a better understanding of the

[7] The search space of SAPS's α, ρ and P_{smooth} parameters in our investigation of additive vs. multiplicative methods [7] was approximately $(20 \times 20 \times 5)$.

factors underlying our method's successes and failures. Another is on extending the US heuristic to incorporate neighbourhood weighting.

References

1. Jeremy Frank. Learning short-term weights for GSAT. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 384–391, 1997.
2. Ian Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pages 28–33, 1993.
3. Frank Hutter, Dave Tompkins, and Holger Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *Proceedings of CP-02*, volume 2470 of *LNCS*, pages 233–248. Springer Verlag, 2002.
4. Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)*, pages 40–45, Menlo Park, CA, 1993. AAAI Press.
5. Steven Prestwich. Local search on sat-encoded csps. In *Proceedings of the 6th International Conference on Theory and Applications of satisfiability Testing*, S. Margherita Ligure, Portofino, Italy, May 2003.
6. Bart Selman and Henry Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 290–295, 1993.
7. John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'04)*, 2004.
8. Dave Tompkins and Holger Hoos. Warped landscapes and random acts of SAT solving. In *Proc. of the Eighth International Symposium on Artificial Intelligence and Mathematics - AMAI, AI&M 1-2004*, Fort Lauderdale, Florida, USA, January 2004.
9. Benjamin Wah and Yi Shang. Discrete lagrangian-based search for solving MAX-SAT problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 378–383, 1997.
10. Zhe Wu and Benjamin Wah. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI'99)*, pages 673–678, Orlando, Florida, 1999.