

Adaptive Clause Weight Redistribution

Abdelraouf Ishtaiwi^{1,2}, John Thornton^{1,2}, Anbulagan³, Abdul Sattar^{1,2}, and
Duc Nghia Pham^{1,2}

¹ IIS, Griffith University, QLD, Australia

² DisPRR, National ICT Australia Ltd, QLD, Australia

³ Logic and Computation Program, National ICT Australia Ltd, Canberra, Australia
{a.ishtaiwi,j.thornton,anbulagan,abdul.sattar,duc-nghia.pham}@nicta.com.au

Abstract. In recent years, dynamic local search (DLS) clause weighting algorithms have emerged as the local search state-of-the-art for solving propositional satisfiability problems. However, most DLS algorithms require the tuning of domain dependent parameters before their performance becomes competitive. If manual parameter tuning is impractical then various mechanisms have been developed that can automatically adjust a parameter value during the search. To date, the most effective adaptive clause weighting algorithm is RSAPS. However, RSAPS is unable to convincingly outperform the best non-weighting adaptive algorithm AdaptNovelty⁺, even though manually tuned clause weighting algorithms can routinely outperform the Novelty⁺ heuristic on which AdaptNovelty⁺ is based.

In this study we introduce R+DDFW⁺, an enhanced version of the DDFW clause weighting algorithm developed in 2005, that not only adapts the total amount of weight according to the degree of stagnation in the search, but also incorporates the latest resolution-based pre-processing approach used by the winner of the 2005 SAT competition (R+AdaptNovelty⁺). In an empirical study we show R+DDFW⁺ improves on DDFW and outperforms the other leading adaptive (R+AdaptNovelty⁺, R+RSAPS) and non-adaptive (R+G²WSAT) local search solvers over a range of random and structured benchmark problems.

1 Introduction

Since the development of the Breakout heuristic [1], clause weighting dynamic local search (DLS) algorithms for SAT have been intensively investigated, and continually improved [2, 3]. However, the performance of these algorithms remained inferior to their non-weighting counterparts (e.g. [4]), until the more recent development of weight smoothing heuristics [5–8]). Such algorithms now represent the state-of-the-art for stochastic local search (SLS) methods on SAT problems. Interestingly, the most successful DLS algorithms (i.e. DLM [5], SAPS [7] and PAWS [8]) have converged on the same underlying weighting strategy: increasing weights on false clauses in a local minimum, then periodically reducing weights according to a problem specific parameter setting. DLM mainly differs from PAWS by incorporating a plateau searching heuristic and PAWS mainly differs from SAPS by performing additive rather than multiplicative weight updates.

However, a key weakness of these approaches is that their performance depends on problem specific parameter tuning. This issue was partly addressed in the development of a reactive version of SAPS (RSAPS [7]) which used a similar adaptive noise mechanism to that used in AdaptNovelty⁺ [9]. Nevertheless, as the 2005 International SAT competition (SAT2005) has shown, DLS algorithms, including RSAPS, have not proved competitive with the best SLS techniques when they are constrained to use fixed parameter values. This is explained by the sensitivity of the control parameters and by the lack of a sufficiently effective adaptive mechanism to adjust these parameters to specific problem instances.

In 2005, a new approach to clause weighting was developed, known as Divide and Distribute Fixed Weight (DDFW) [10]. DDFW’s approach is to redistribute weight from satisfied to unsatisfied clauses in each local minimum, unifying the increase and decrease phases of weight control into a single action. This means there is no requirement for a problem specific parameter to decide when weight is to be reduced. In addition, DDFW only alters weights on those clauses that are false in a local minimum and an equal number of satisfied clauses. This makes it more efficient than earlier weight smoothing algorithms that also performed smoothing at each local minimum, but did so by adjusting weight on all the clauses in the problem (e.g. SDF [11]). However, DDFW still has a parameter (W_{init}) whose setting can effect performance by varying the amount of weight that is initially given to each clause. In the earlier empirical evaluation of DDFW this initial weight was fixed. However, the existence of such a parameter implies that DDFW could benefit from an adaptive mechanism to vary the amount of weight that is distributed according to the dynamic search conditions.

Also in 2005, it was shown that the performance of various SLS techniques can be significantly improved by the addition of a resolution-based preprocessing phase [12]. This work initially produced the winning algorithm in the SAT2005 satisfiable random problem category, R+AdaptNovelty⁺. However, in the subsequent paper [12], the largest performance gains were obtained for clause weighting algorithms solving *structured* problem instances. Here R+AdaptNovelty⁺ was convincingly outperformed by a R+RSAPS and a *tuned* version of R+PAWS on a range of quasigroup existence problems and standard structured SAT benchmarks.

The question we address in the current paper is which SLS SAT algorithm should be preferred in situations where parameter tuning is impractical and we have no other information that could guide us in choosing a particular approach. As this is exactly the situation we would expect to find in many real world applications, we take the relevance and importance of this question to be self evident. While the initial work on DDFW [10] showed that a fixed parameter version was able to outperform AdaptNovelty⁺ and RSAPS on a range of random and structured SAT benchmarks, the question still remains whether the performance of DDFW can be further improved by incorporating a similar adaptive mechanism to that used by AdaptNovelty⁺ and RSAPS to control the W_{init} parameter.

It also remains unanswered whether such an adaptive version of DDFW could derive enough benefit from resolution-based preprocessing to outperform the ex-

isting resolution-based versions of R+AdaptNovelty⁺ or R+RSAPS. In addition, in SAT2005 a new SLS algorithm was introduced, G²WSAT [13], which went on to win the silver medal in the random category of the competition. This algorithm has subsequently been improved and it too has yet incorporate a resolution-based preprocessor.

As a result of these considerations, our specific aim in the remainder of the paper is to introduce an adaptive resolution-incorporating version of DDFW (called R+DDFW⁺) and to compare it with the three other most promising general purpose SLS SAT solvers, namely R+AdaptNovelty⁺, R+RSAPS and an enhanced R+G²WSAT. On the basis of an empirical study that covers a range of problems from SAT2005, the quasigroup existence domain and the SATLIB benchmark library, we conclude that R+DDFW⁺ has the best overall performance of these methods, and that it derives significant benefits from its new adaptive mechanism.

2 Clause Weighting for SAT

Clause weighting local search algorithms for SAT follow the basic procedure of repeatedly flipping single literals that produce the greatest reduction in the sum of false clause weights. Typically, all literals are randomly initialized, and all clauses are given a fixed initial weight. The search then continues until no further cost reduction is possible, at which point the weight on all unsatisfied clauses is increased, and the search is resumed, punctuated with periodic weight reductions.

Existing clause weighting algorithms differ primarily in the schemes used to control the clause weights, and in the definition of the points where weight should be adjusted. Multiplicative methods, such as SAPS, generally adjust weights when no further improving moves are available in the local neighbourhood. This can be when all possible flips lead to a worse cost, or when no flip will improve cost, but some flips will lead to equal cost solutions. As multiplicative real-valued weights have much finer granularity, the presence of equal cost flips is much more unlikely than for an additive approach (such as DLM or PAWS), where weight is adjusted in integer units. This means that additive approaches frequently have the choice between adjusting weight when no improving move is available, or taking an equal cost (flat) move.

Despite these differences, the three most well-known clause weighting algorithms (DLM [5], SAPS [7] and PAWS [8]) share a similar structure in the way that weights are updated:⁴ Firstly, a point is reached where no further improvement in cost appears likely. The precise definition of this point depends on the algorithm, with DLM expending the greatest effort in searching plateau areas

⁴ Additionally, a fourth clause weighting algorithm, GLSSAT [14], uses a similar weight update scheme, additively increasing weights on the least weighted unsatisfied clauses and multiplicatively reducing weights whenever the weight on any one clause exceeds a predefined threshold.

of equal cost moves, and SAPS expending the least by only accepting cost improving moves. Then all three methods converge on increasing weights on the currently false clauses (DLM and PAWS by adding one to each clause and SAPS by multiplying the clause weight by a problem specific parameter $\alpha > 1$). Each method continues this cycle of searching and increasing weight, until, after a certain number of weight increases, clause weights are reduced (DLM and PAWS by subtracting one from all clauses with weight > 1 and SAPS by multiplying all clause weights by a problem specific parameter $\rho < 1$). SAPS is further distinguished by reducing weights probabilistically (according to a parameter P_{smooth}), whereas DLM and PAWS reduce weights after a fixed number of increases (again controlled by parameter). PAWS is mainly distinguished from DLM in being less likely to take equal cost or flat moves. DLM will take up to θ_1 consecutive flat moves, unless all available flat moves have already been used in the last θ_2 moves. PAWS does away with these parameters, taking flat moves with a fixed probability of 15%, otherwise it will increase weight.

However, as we have stressed in the introduction, the performance of these clause weighting algorithms remains very sensitive to the settings of their problem specific parameters (this has been shown in detail in [15]). While this sensitivity is also a problem for the non-weighting algorithms of the WalkSAT family, it has been somewhat counteracted by the use of heuristics that adapt parameter settings during the course of the search. The most successful of these algorithms, AdaptNovelty⁺, works by adapting a noise parameter that controls whether a move is selected randomly or deterministically [9]. In simplified terms, the likelihood of making a random choice is increased the longer the search continues without achieving an improvement in the objective function. A similar scheme was added to SAPS, producing reactive SAPS or RSAPS [7]. However, adapting SAPS was not as successful as adapting Novelty, for, while a tuned SAPS generally produces better performance than a tuned Novelty+, RSAPS has not been able to reach the consistent performance AdaptNovelty⁺ in the recent SAT competitions. One reason for this may be that SAPS requires the setting of *three* parameters to achieve its best performance, while RSAPS only adapts one of these parameters. Similarly, DLM requires the setting of at least three parameters before producing its best performance. In contrast, PAWS (like Novelty) only requires the tuning of a single parameter, but to date no successful heuristic has been discovered that can automatically adapt this value.

More recently, work has concentrated on *learning* empirical hardness models in order to predict the best parameter settings for SAPS [16]. This approach requires a set of training instances that are repeatedly solved by SAPS using different parameter settings. After this training phase, parameter settings can be generated for previously unseen instances taken from the same problem class. Results from this work are encouraging and could be generally applied to other local search algorithms. However, the weakness is that training is required on a representative test set before good predictions can be produced. It remains to be seen whether a general model can be devised that can predict good parameter settings for the SAT domain as a whole. In the meantime, if we are limited

to solving problems from an undisclosed problem distribution and if manual parameter tuning is ruled out of court, then the best available clause weighting algorithm is probably RSAPS (discounting DDFW for the moment).

3 Divide and Distribute Fixed Weights

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 then escapes local minima by *transferring weight from satisfied to unsatisfied clauses*. The other existing state-of-the-art clause weighting algorithms have all divided the weighting process into two distinct steps: i) increasing weights on false clauses in local minima and ii) decreasing or normalising weights on all clauses after a series of increases, so that weight growth does not spiral out of control. DDFW combines this process into a single step of weight transfer, thereby dispensing with the need to decide when to reduce or normalise weight. In this respect, DDFW is similar to the predecessors of SAPS (SDF [6] and ESG [11]), which both adjust *and* normalise the weight distribution in each local minimum. Because these methods adjust weight across all clauses, they are considerably less efficient than SAPS, which normalises weight after visiting a series of local minima.⁵ DDFW escapes the inefficiencies of SDF and ESG by only transferring weights between pairs of clauses, rather than normalising weight on all clauses. This transfer involves selecting a single satisfied clause for each currently unsatisfied clause in a local minimum, reducing the weight on the satisfied clause by an integer amount and adding that amount to the weight on the unsatisfied clause. Hence DDFW retains the additive (integer) weighting approach of DLM and PAWS, and combines this with an efficient method of weight redistribution, i.e. one that keeps all weight reasonably normalised without repeatedly adjusting weights on all clauses.

DDFW's weight transfer approach also bears similarities to the operations research subgradient optimisation techniques discussed in [11]. In these approaches, Lagrangian multipliers, analogous to the clause weights used in SAT, are associated with problem constraints, and are adjusted in local minima so that multipliers on unsatisfied constraints are increased and multipliers on satisfied constraints are reduced. This *symmetrical* treatment of satisfied and unsatisfied constraints is mirrored in DDFW, but not in the other SAT clause weighting approaches (which increase weights and then adjust). However, DDFW differs from subgradient optimisation in that weight is only transferred between pairs of clauses and not across the board, meaning less computation is required.

3.1 Exploiting Neighbourhood Structure

The second and more original idea developed in DDFW, is the exploitation of neighbourhood relationships between clauses when deciding which pairs of clauses will exchange weight.

⁵ Increasing weight on *false* clauses in a local minimum is efficient because only a small proportion of the total clauses will be false at any one time.

Algorithm 1 DDFW⁺(\mathcal{F})

```
1: randomly instantiate each literal in  $\mathcal{F}$ ;  
2: set the weight  $w_a$  of each clause  $c_a \in \mathcal{F}$  to two;  
3: set the minimum  $m$  to the number of false clauses  $c_f \in F$ ;  
4: set counter  $i$  to zero and boolean  $b$  to false;  
5: while solution is not found and not timeout do  
6:   calculate the list  $\mathcal{L}$  of literals causing the greatest reduction in weighted cost  $\Delta w$  when flipped;  
7:   if ( $\Delta w < 0$ ) or ( $\Delta w = 0$  and probability  $\leq 15\%$ ) then  
8:     randomly flip a literal in  $\mathcal{L}$ ;  
9:     if number of false clauses  $< m$  then  
10:      set counter  $i$  to zero and minimum  $m$  to the number of false clauses;  
11:     else  
12:       increment counter  $i$  by one;  
13:       if  $i \geq$  number of literals in  $\mathcal{F}$  then  
14:         set counter  $i$  to zero;  
15:         if  $b$  is false then  
16:           increase the weight  $w_a$  of each clause  $c_a \in \mathcal{F}$  by one;  
17:           set boolean  $b$  to true;  
18:         else  
19:           set the weight  $w_s$  of each satisfied clause  $c_s \in \mathcal{F}$  to two;  
20:           set the weight  $w_f$  of each false clause  $c_f \in \mathcal{F}$  to three;  
21:           set boolean  $b$  to false;  
22:         end if  
23:       end if  
24:     end if  
25:   else  
26:     for each false clause  $c_f \in \mathcal{F}$  do  
27:       select a satisfied same sign neighbouring clause  $c_n$  with maximum weight  $w_n$ ;  
28:       if  $w_n < 2$  then  
29:         randomly select a clause  $c_n$  with weight  $w_n \geq 2$ ;  
30:       end if  
31:       if  $w_n > 2$  then  
32:         transfer a weight of two from  $c_n$  to  $c_f$ ;  
33:       else  
34:         transfer a weight of one from  $c_n$  to  $c_f$ ;  
35:       end if  
36:     end for  
37:   end if  
38: end while
```

We term clause c_i to be a neighbour of clause c_j , if there exists at least one literal $l_{im} \in c_i$ and a second literal $l_{jn} \in c_j$ such that $l_{im} = l_{jn}$. Furthermore, we term c_i to be a *same sign* neighbour of c_j if the sign of any $l_{im} \in c_i$ is equal to the sign of any $l_{jn} \in c_j$ where $l_{im} = l_{jn}$. From this it follows that each literal $l_{im} \in c_i$ will have a set of same sign neighbouring clauses $C_{l_{im}}$. Now, if c_i is false, this implies all literals $l_{im} \in c_i$ evaluate to false. Hence flipping any l_{im} will cause it to become true in c_i , and also to become true in all the same sign neighbouring clauses of l_{im} , i.e. $C_{l_{im}}$. Therefore, flipping l_{im} will *help* all the clauses in $C_{l_{im}}$, i.e. it will increase the number of true literals, thereby increasing the overall level of satisfaction for those clauses. Conversely, l_{im} has a corresponding set of opposite sign clauses that would be *damaged* when l_{im} is flipped.

The reasoning behind the DDFW neighbourhood weighting heuristic proceeds as follows: if a clause c_i is false in a local minimum, it needs extra weight in order to encourage the search to satisfy it. If we are to pick a neighbouring clause c_j that will donate weight to c_i , we should pick the clause that is most able to pay. Hence, the clause should firstly already be satisfied. Secondly, it should be a same sign neighbour of c_i , as when c_i is eventually satisfied by flipping

l_{im} , this will also raise the level of satisfaction of l_{im} 's same sign neighbours. However, taking weight from c_j only increases the chance that c_j will be helped when c_i is satisfied, i.e. not all literals in c_i are necessarily shared as same sign literals in c_j , and a non-shared literal may be subsequently flipped to satisfy c_i . The third criteria is that the donating clause should also have the largest store of weight within the set of satisfied same sign neighbours of c_i .

The intuition behind the DDFW heuristic is that clauses that share same sign literals should form alliances, because a flip that benefits one of these clauses will always benefit some other member(s) of the group. Hence, clauses that are connected in this way will form groups that tend towards keeping each other satisfied. However, these groups are not closed, as each clause will have clauses within its own group that are connected by other literals to other groups. Weight is therefore able to move between groups as necessary, rather than being uniformly smoothed (as in existing methods).

3.2 Adapting DDFW

The new feature introduced in this study is the development of an adaptive mechanism that alters the total amount of weight that DDFW distributes according to the degree of stagnation in the search. This DDFW⁺ heuristic is detailed in lines 9-24 of Algorithm 1. Previously DDFW would have initialised the weight of each clause to W_{init} (which was fixed at 8 in [10]). Now this initialisation value is set at two in line 2 of Algorithm 1, but can be altered during the search as follows: if the search executes a consecutive series of i flips without reducing the total number of false clauses, where i is equal to the number of literals in the problem, then the amount of weight on each clause is increased by one 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 two and on each false clause back to three. 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 $c_a \in \mathcal{F}$ plus the total number of false clauses $c_f \in \mathcal{F}$.

The reasoning behind this adaptive heuristic is based on our observation that manually adjusting DDFW's original parameter W_{init} has a noticeable effect runtime performance, and that on several problems the default value of eight was not optimal. This is illustrated in Figure 1, which shows that on problem (a) $W_{init} = 8$ is near optimal whereas on problem (b) $W_{init} = 2$ is the better choice (if we consider the underlying trend). We conjectured that we could circumvent the need to initialise the clauses with more weight at the start of the search by allowing context sensitive weight increases during the search. Hence we developed a stagnation measure, much like the measures used in AdaptNovelty and RSAPS, that injects extra weight when no cost improvement occurs and made the frequency of this injection depend on the size of the problem. The unusual feature of the DDFW⁺ heuristic is that the search will only effect one

increase after which, if stagnation is observed again, the weights are reset. This reset mechanism was adopted after a series of empirical trials that tested various combinations of weight increase and decrease phases. Our main difficulty was to keep the weight growth within bounds and we could find no decrease scheme that worked well across a wide range of problems without requiring a further problem dependent parameter (which would obviously defeat the purpose of the study). We therefore settled on a simple reset strategy that places a strict limit on weight growth and avoids adding an additional parameter.

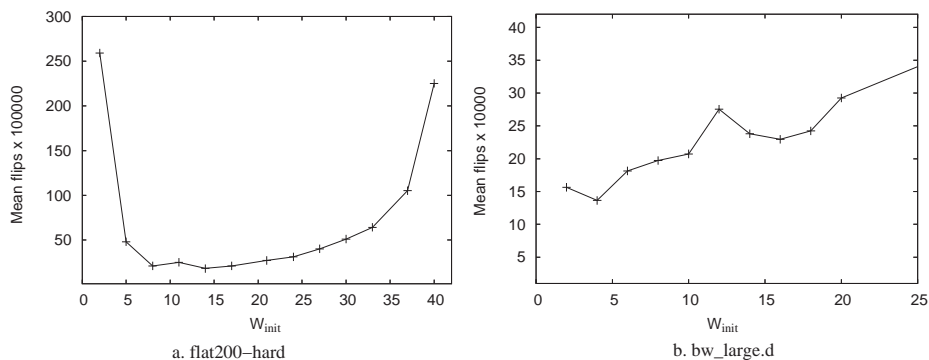


Fig. 1. Flip performance of DDFW for various settings of the W_{init} parameter

4 Resolution Based Preprocessing

As discussed in the introduction, significant performance benefits have been gained by preprocessing a problem using resolution before starting a search. This result is already well-known in the complete search community, where Satz [17] uses a restricted resolution procedure, adding resolvents of length ≤ 3 , as a pre-processor before running the complete backtrack search. The same procedure has now been added to AdaptNovelty⁺, PAWS, RSAPS and WalkSAT [12], and there is empirical evidence to suggest that clause weighting algorithms in particular benefit from this approach when solving structured real-world problems.

Resolution itself is a rule of inference widely used in automated deduction [18–20]. In the present study, as in [12], we implement the Satz resolution process (see Algorithm 2) as follows: when two clauses of a CNF formula have the property that some variable x_i occurs positively in one and negatively in the other, the resolvent of the clauses is a disjunction of all the literals occurring in the clauses except x_i and \bar{x}_i . For example, the clause $(x_2 \vee x_3 \vee \bar{x}_4)$ is the resolvent for the clauses $(\bar{x}_1 \vee x_2 \vee x_3)$ and $(x_1 \vee x_2 \vee \bar{x}_4)$ and is added to the clause set. The new clauses, provided they are of length ≤ 3 , can in turn be used to produce other resolvents. The process is repeated until saturation. Duplicate and subsumed

Algorithm 2 ComputeResolvents(\mathcal{F})

```
1: for each clause  $c_1$  of length  $\leq 3$  in  $\mathcal{F}$  do
2:   for each literal  $l$  of  $c_1$  do
3:     for each clause  $c_2$  of length  $\leq 3$  in  $\mathcal{F}$  s.t.  $\bar{l} \in c_2$  do
4:       Compute resolvent  $r = (c_1 \setminus \{l\}) \cup (c_2 \setminus \{\bar{l}\})$ ;
5:       if  $r$  is empty then
6:         return "unsatisfiable";
7:       else
8:         if  $r$  is of length  $\leq 3$  then
9:            $\mathcal{F} := \mathcal{F} \cup \{r\}$ ;
10:        end if
11:      end if
12:    end for
13:  end for
14: end for
```

clauses are deleted, as are tautologies and any duplicate literals in a clause. It is worth noting that this resolution phase takes polynomial time.

5 Experimental Evaluation

As the resolution process is encapsulated in a preprocessing phase, it can be added to an existing SAT solver as a separate module, leaving the original solver unaltered. In our experimental study we added this preprocessing phase (as defined in Algorithm 2) to DDFW, DDFW⁺, RSAPS, AdaptNovelty⁺ and G²WSAT, producing R+DDFW, R+DDFW⁺, R+RSAPS, R+AdaptNovelty⁺ and R+G²-WSAT. Of these algorithms, R+RSAPS and R+AdaptNovelty⁺ have already been entered into SAT2005 and reported in [12].⁶ However, R+DDFW, R+DDFW⁺ and R+G²WSAT are new algorithms whose performance has yet to be reported.⁷ We chose to compare DDFW with R+AdaptNovelty⁺ and R+G²WSAT because these two algorithms were the gold and silver medal winners in the SAT2005 satisfiable random category competition and achieved the best overall local search results in terms of the number of problems solved. We chose R+RSAPS because it was the best performing clause weighting algorithm in the competition. Together, therefore, these three algorithms can lay claim to being the state-of-the-art for general purpose local search SAT solving when manual parameter tuning is disallowed.

To evaluate the relative performance of these algorithms we divided our empirical study into four areas: firstly, we attempted to reproduce a reduced problem set similar to that used in the random category of the SAT competition (as this is the domain where local search techniques have dominated). To do this we selected the 50 satisfiable *k*3 problems from the SAT2004 competition

⁶ AdaptNovelty⁺ and RSAPS are available as part of the UBCSAT solver from <http://www.satlib.org/ubcsat/>

⁷ G²WSAT is available at <http://www.laria.u-picardie.fr/%7Eecli/g2wsat2005.c>. This latest version is described by the authors as generally more than 50% faster than the version entered in SAT2005.

benchmark. Secondly, we obtained the 10 SATLIB quasigroup existence problems used in [12]. These problems are relevant because they exhibit a balance between randomness and structure, while also producing clause sets to which resolution can be applied effectively. Thirdly, we obtained the structured problem set used to originally evaluate SAPS [7]. These problems have been widely used to evaluate clause weighting algorithms (e.g. in [8]) and contain a representative cross-section taken from the DIMACS and SATLIB libraries. In this set we also included 4 of the well-known DIMACS 16-bit parity learning problems. Finally, we used the 16 ferry planning problems from the SAT2005 competition that our local search techniques were able to solve. This was to give an indication of relative performance on the SAT2005 industrial problems.

Overall, the problem set is designed to show how R+DDFW⁺ compares in absolute terms to the other algorithms and to examine the relative effect of the adaptive mechanism on differing problem classes. For this reason we also include the results for R+DDFW (i.e. *without* the adaptive mechanism). All experiments were performed on a Dell machine with 3.1GHz CPU and 1GB memory, except for the quasigroup problems which were run on a Sun supercomputer with 8 × Sun Fire V880 servers, each with 8 × UltraSPARC-III 900MHz CPU and 8GB memory per node. Cut-offs for the various algorithms were set as follows: first R+DDFW was given 10 trials on each problem with a flip cut-off of 1,000,000. If it was unable to solve any trial then the cut-off was raised to 10,000,000, and then in steps of 10,000,000 until at least one solution was found. R+DDFW was then allowed 100 trials at the given flip cut-off for all instances except the ferry problems, where it was limited to 10 trials. The total time allowed for R+DDFW on each set of 10 or 100 trials was then recorded and all other algorithms were given this as a time cut-off on each problem. The following results detail the mean time in seconds (including the resolution preprocessing step), mean flips and the success rate for these cut-offs (results in bold indicate the best performance for a particular problem).

5.1 SAT Competition Problem Results

The results in Figure 2a graph the performance of R+DDFW⁺, R+DDFW, R+AdaptNovelty⁺ and R+G²WSAT after applying resolution on the 50 *k*3 problems from the SAT2004 competition (as R+RSAPS had very poor performance on the random instances it has been omitted from the figure and the following discussion). The graph shows the cumulative percentage of problems solved against runtime, assuming that each instance is solved in parallel (for example, in Figure 2a after 5 seconds approximately 71% of the 50 × 100 trials for R+DDFW will have terminated). Here R+DDFW⁺ and R+DDFW were the only solvers that could reach a 100% success rate over all trials. Although R+G²WSAT was competitive and could solve the easier problems faster than R+DDFW, it was unable to match R+DDFW as problem difficulty increased. Overall the graph shows that R+DDFW⁺ has the superior performance across the range of problem sizes, clearly dominating R+DDFW and

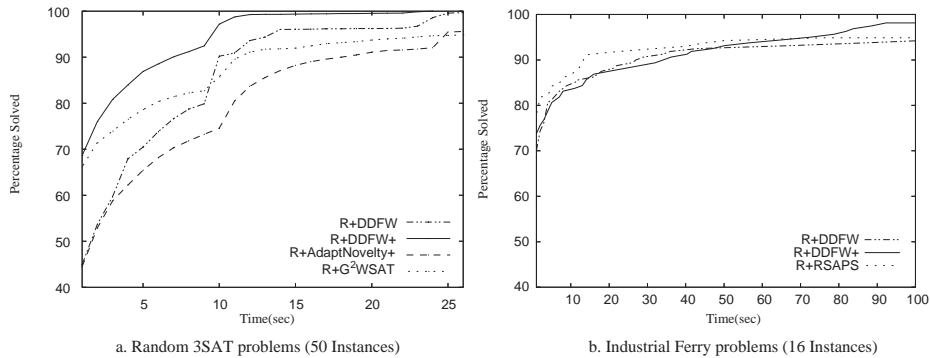


Fig. 2. Results for the SAT2004 random problems and SAT2005 industrial problems

thereby demonstrating that the new adaptive heuristic can positively affect runtime performance. Figure 2a also shows that R+G²WSAT generally dominates R+AdaptNovelty⁺, although R+AdaptNovelty⁺ does match R+G²WSAT’s success rate over the whole problem set.

The results for the SAT2005 industrial ferry problems are shown in Figure 2b and in Table 1 (as R+G²WSAT and R+AdaptNovelty⁺ were only able to solve 29% and 9% of the ferry instances respectively, they have been removed from the graphical analysis). Looking at Figure 2b we can see that R+RSAPS, after performing poorly on the random problems, is now able to dominate R+DDFW across the range of the ferry problems, but cannot quite reach R+DDFW⁺’s 97.5% success rate. However, Table 1 shows that R+RSAPS is able to solve 10 of the 16 ferry problems faster than either DDFW variant, and that R+DDFW⁺’s superior success rate is largely based on instance ferry4001. We must therefore conclude that there is little to choose between R+RSAPS and R+DDFW⁺ on these problems. Nevertheless, R+DDFW⁺ does more clearly outperform R+DDFW and again demonstrates that the adaptive heuristic can make noticeable improvements.

5.2 Quasigroup Problem Results

Table 2 shows the performance of the solvers on the quasigroup problems. Here we can see that R+DDFW and R+DDFW⁺ clearly emerge as the two best solvers, sharing the best results for each instance and both achieving an overall success rate of 100%. Comparing between the two DDFW methods, for the first time it becomes unclear whether the adaptive heuristic has made any difference, as, for most instances the results are comparable. However R+DDFW⁺ does exhibit noticeably better performance on instance qg1-08, whereas R+DDFW shows equally strong performance on qg7-13. We should therefore conclude that the adaptive mechanism does not change the overall performance of DDFW on this problem set, although it can make a difference, either positively or negatively, on individual instances.

Problems	R+DDFW ⁺			R+DDFW			R+AdaptNovelty ⁺			G ² WSAT			R+RSAPS		
	Time	Flips	%	Time	Flips	%	Time	Flips	%	Time	Flips	%	Time	Flips	%
ferry3994	3.48	2,073,195	100	1.1	786,967	100	n/a	n/a	0	n/a	n/a	0	0.6	530,501	100
ferry3995	1.54	933,726	100	0.6	458,302	100	n/a	n/a	0	n/a	n/a	0	0.1	89,730	100
ferry3996	0.0	7,903	100	0.0	13,942	100	3.9	8,204,511	20	0.1	275,547	100	0.0	7,741	100
ferry3997	10.3	8,238,690	60	10.3	5,055,539	90	n/a	n/a	0	n/a	n/a	0	9.2	6,742,006	50
ferry3998	0.0	6,526	100	0.0	8,586	100	2.1	3,344,936	100	0.1	180,334	100	0.0	5,070	100
ferry3999	9.81	5,312,170	100	3.2	1,908,547	100	n/a	n/a	0	n/a	n/a	0	0.6	304,680	100
ferry4000	0.0	31,774	100	0.0	19,280	100	n/a	n/a	0	1.8	2,442,300	80	0.0	12,771	100
ferry4001	63.1	24,392,288	100	99.4	40,117,368	90	n/a	n/a	0	n/a	n/a	0	90.0	54,061,467	80
ferry4002	0.0	9,637	100	0.0	20,336	100	4.8	7,535,284	30	2.1	1,958,552	90	0.0	3,852	100
ferry4003	21.2	10,395,968	100	21.2	7,773,439	50	n/a	n/a	0	n/a	n/a	0	7.2	2,884,301	100
ferry4004	0.0	30,348	100	0.1	40,547	100	n/a	n/a	0	2.4	2,437,826	50	0.0	20,394	100
ferry4006	0.0	14,640	100	0.0	17,697	100	n/a	n/a	0	4.9	2,616,491	20	0.0	9,160	100
ferry4008	0.0	33,192	100	0.1	51,796	100	n/a	n/a	0	3.2	2,655,066	20	0.1	42,938	100
ferry4009	0.0	23,163	100	0.1	24,015	100	n/a	n/a	0	n/a	n/a	0	0.1	17,612	100
ferry3992	0.1	60,525	100	0.2	102,413	100	n/a	n/a	0	n/a	n/a	0	0.2	92,346	100
ferry3993	0.0	26,878	100	0.1	43,595	100	n/a	n/a	0	7.2	3,399,169	10	0.2	54,742	100

Table 1. Results for the SAT2005 industrial ferry planning problems

Problems	R+DDFW ⁺			R+DDFW			R+AdaptNovelty ⁺			R+G ² WSAT			R+RSAPS		
	Time	Flips	%	Time	Flips	%	Time	Flips	%	Time	Flips	%	Time	Flips	%
qg1-07	0.0	4,388	100	0.1	11,375	100	0.2	14,840	100	0.1	9,600	100	0.1	4,901	100
qg1-08	10.2	352,276	100	21.8	601,271	100	33.8	1,076,689	100	28.8	2,818,904	100	64.6	2,153,008	99
qg2-07	0.0	2,361	100	0.0	2,035	100	0.1	9,094	100	0.1	5,073	100	0.1	2,478	100
qg2-08	57.5	1,556,545	100	60.0	1,346,438	100	77.1	1,906,196	20	79.8	4,569,088	50	71.5	1,879,019	70
qg3-08	0.1	16,867	100	0.1	21,986	100	0.6	78,849	100	0.1	24,534	100	0.2	11,049	100
qg4-09	0.2	25,311	100	0.2	26,123	100	1.5	169,169	100	0.7	142,619	100	1.2	54,920	100
qg5-11	0.2	7,303	100	0.2	6,797	100	2.3	131,924	100	0.4	29,992	100	0.6	11,014	100
qg6-09	0.0	478	100	0.0	466	100	0.0	3,644	100	0.0	686	100	0.6	11,753	100
qg7-09	0.0	292	100	0.0	299	100	0.0	698	100	0.0	412	100	0.0	295	100
qg7-13	9.3	229,258	100	3.2	122,091	100	16.3	5,351,459	56	n/a	n/a	0	24.9	373,456	10

Table 2. Results for Quasigroup SATLIB problems

5.3 Structured Problem Results

Table 3 shows the results for the structured problems taken from the original SAPS problem set [7] and the parity learning problems taken from the original PAWS study [8]. This set comprises of two blocks world planning (bw) problems, two logistics planning instances, two flat graph coloring problems (flat), two all-interval-series problems (ais) and four 16-bit parity learning problems (par16*). The results confirm our earlier observation from the random problem results that G²WSAT does not scale as well as DDFW. In this case R+G²WSAT is the best algorithm on the smaller ais, logistics and flat problems, but is outperformed by R+DDFW on each of the larger instances of these problems. In addition, R+RSAPS has stronger performance than R+DDFW on the ais and par16 problems.

However, the situation changes if we consider the performance of R+DDFW⁺. In comparison to R+DDFW, R+DDFW⁺ is better on the ais10, both logistics and all par16 problems, whereas R+DDFW is only better on the ais12 and flat200 problems (the two methods perform identically on the bw problems be-

cause the large number of literals mean the adaptive mechanism is not used). These results show that the R+DDFW⁺ adaptive mechanism has again produced noticeable performance benefits, and has improved the overall behaviour of R+DDFW on this problem set. In addition, if we take a simple count of the number of problems on which R+DDFW⁺ dominates we can see that it is also the best of the five algorithms considered.

	R+DDFW ⁺			R+DDFW			R+AdaptNovelty ⁺			R+G ² WSAT			R+RSAPS		
Problems	Time	Flips	%	Time	Flips	%	Time	Flips	%	Time	Flips	%	Time	Flips	%
ais10	0.0	298,650	100	0.5	498,911	100	1.4	1,214,321	100	0.0	112,044	100	0.0	25,459	100
ais12	5.0	4,036,866	100	2.3	1,934,170	100	10.1	7,328,426	51	2.4	1,854,652	100	0.2	187,743	100
logistics-c	0.0	242,540	100	0.3	414,645	100	0.0	26,696	100	0.0	23,623	100	0.0	5,364	100
logistics-d	0.1	16,708	100	0.1	25,869	100	0.1	109,650	100	0.5	350,711	100	0.1	20,918	100
flat200-m	0.3	262,905	100	0.2	161,902	100	0.2	351,563	100	0.1	150,588	100	0.4	362,786	100
flat200-h	3.2	2,814,221	100	1.0	1,014,878	100	3.6	8,166,964	36	2.4	5,535,185	100	3.5	3,517,562	94
bw_large.c	=	=	100	0.6	145,607	100	6.7	5,660,460	67	n/a	n/a	0	21.3	4,258,483	91
bw_large.d	=	=	100	1.4	184,874	100	13.4	7,974,818	38	n/a	n/a	0	n/a	n/a	0
par16-1	4.3	3,828,086	100	7.1	5,229,852	50	7.4	15,608,349	15	n/a	n/a	0	7.4	1,164,862	80
par16-2	23.2	21,670,517	100	27.9	20,542,514	60	36.8	54,634,563	10	n/a	n/a	0	16.0	17,581,843	100
par16-3	7.7	7,146,517	100	24.4	17,959,087	70	32.7	50,828,991	40	31.8	26,133,070	30	16.0	18,890,265	100
par16-4	2.9	2,699,444	100	11.4	12,800,152	100	26.8	41,099,634	50	26.5	51,205,540	60	8.1	9,445,556	100

Table 3. Results for structured problems from the SAPS and PAWS original studies, (the = symbol means that R+DDFW⁺ behaves identically to R+DDFW on these problems)

6 Analysis and Conclusions

Overall we can conclude that the addition of an adaptive mechanism has improved the performance of DDFW over the entire range of the problem sets we have considered. The strongest dominance was observed on the random 3-SAT and parity problems (shown in Figure 2a and Table 3 respectively). On the other problems R+DDFW⁺ improved over R+DDFW on 10 of the 16 ferry problems (in Table 1), 6 of the 10 quasigroup problems (in Table 2) and stays neutral on the remaining real-world problems (in Table 3).

We can further conclude that R+DDFW (i.e. even without the adaptive mechanism) has the better overall performance in comparison to AdaptNovelty⁺, G²WSAT and R+RSAPS. If we first look at R+G²WSAT, while it performed well on the smaller random problems, it could not match R+DDFW on the larger more difficult random problems. In the other categories R+G²WSAT was less competitive, again showing promise on the smaller structured problems in Table 3, but failing to scale up as well as R+DDFW on the more difficult problems. Interestingly, G²WSAT performed strongly on the quasigroup problems when no resolution was performed, but was uncompetitive after resolution (these results are not reported in the current paper). This confirms the findings in [12] that suggest clause weighting algorithms can gain more advantage from resolution

than non-weighting algorithms. In addition, R+G²WSAT was uniformly worse than R+DDFW on the ferry problems.

Turning our attention to R+RSAPS, this algorithm showed slightly better performance than R+DDFW on the structured and ferry problems, dominating on 10 of the 16 ferry problems and on all the parity problems, with R+DDFW showing the better performance on the remaining 6 ferry problems and on the other larger structured problems. However, R+RSAPS was outperformed by R+DDFW⁺ on the parity problems, was uniformly worse on the random problems and was uncompetitive with R+DDFW on the quasigroup problems, thereby failing to show the same robust performance as R+DDFW and R+DDFW⁺

is funded through the Australian Government's *Backing Australia's Ability* initiative and also through the Australian Research Council.

References

1. Morris, P.: The Breakout method for escaping from local minima. In: Proceedings of 11th AAI. (1993) 40–45
2. Cha, B., Iwama, K.: Adding new clauses for faster local search. In: Proceedings of 13th AAI. (1996) 332–337
3. Frank, J.: Learning short-term clause weights for GSAT. In: Proceedings of 15th IJCAI. (1997) 384–389
4. McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: Proceedings of 14th AAI. (1997) 321–326
5. Wu, Z., Wah, B.: An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In: Proceedings of 17th AAI. (2000) 310–315
6. Schuurmans, D., Southey, F.: Local search characteristics of incomplete SAT procedures. In: Proceedings of 10th AAI. (2000) 297–302
7. Hutter, F., Tompkins, D., Hoos, H.: Scaling and Probabilistic Smoothing: Efficient dynamic local search for SAT. In: Proceedings of 8th CP. (2002) 233–248
8. Thornton, J., Pham, D.N., Bain, S., Ferreira Jr., V.: Additive versus multiplicative clause weighting for SAT. In: Proceedings of 19th AAI. (2004) 191–196
9. Hoos, H.: An adaptive noise mechanism for WalkSAT. In: Proceedings of 19th AAI. (2002) 655–660
10. Ishtaiwi, A., Thornton, J., Sattar, A., Pham, D.N.: Neighbourhood clause weight redistribution in local search for SAT. In: Proceedings of 11th CP. (2005) 772 – 776
11. Schuurmans, D., Southey, F., Holte, R.: The exponentiated subgradient algorithm for heuristic boolean programming. In: Proceedings of 17th IJCAI. (2001) 334–341
12. Anbulagan, Pham, D., Slaney, J., Sattar, A.: Old resolution meets modern SLS. In: Proceedings of 20th AAI. (2005) 354–359
13. Li, C.M., Huang, W.: Diversification and determinism in local search for satisfiability. In: Proceedings of 8th SAT. (2005) 158–172
14. Mills, P., Tsang, E.: Guided local search applied to the satisfiability (SAT) problem. In: Proceedings of 15th ASOR. (1999) 872–883
15. Thornton, J.: Clause weighting local search for SAT. *Journal of Automated Reasoning* (2006) (to appear)
16. Hutter, F., Hamadi, Y.: Parameter adjustment based on performance prediction: Towards an instance aware problem solver. In: Technical Report: MSR-TR-2005-125, Microsoft Research, WA. (2005)
17. Li, C.M., Anbulagan: Look-ahead versus look-back for satisfiability problems. In: Proceedings of 3rd CP. (1997) 341–355
18. Quine, W.V.: A way to simplify truth functions. *American Mathematical Monthly* **62** (1955) 627–631
19. Davis, M., Putnam, H.: A computing procedure for quantification theory. *Journal of the ACM* **7** (1960) 201–215
20. Robinson, J.A.: A machine-oriented logic based on the resolution principle. *Journal of the ACM* **12** (1965) 23–41