

# Tie Breaking 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  
{v.ferreira, j.thornton}@griffith.edu.au  
Phone: +00617 55528502

**Abstract.** Clause weighting local search methods are widely used for satisfiability testing. A feature of particular importance for such methods is the scheme used to maintain the clause weight distribution relevant to different areas of the search landscape. Existing methods periodically adjust clause weights either multiplicatively or additively. Tie breaking strategies are used whenever a method's evaluation function encounters more than one optimal candidate flip, with the dominant approach being to break such ties randomly. Although this is acceptable for multiplicative methods as they rarely encounter such situations, additive methods encounter significantly more tie breaking scenarios in their landscapes, and therefore a more refined tie breaking strategy is of much greater relevance. This paper proposes a new way of handling the tie breaking situations frequently encountered in the landscapes of additive constraint weighting local search methods. We demonstrate through an empirical study that when this idea is used to modify the purely random tie breaking strategy of a state-of-the-art solver, the modified method significantly outperforms the existing one on a range of benchmarks, especially when we consider the encodings of large and structured problems.

**Content Areas: Search, Constraint Satisfaction**

## 1 Introduction

Local search methods are of considerable interest to the AI community due to their ability to efficiently find solutions to combinatorial problems that are beyond the reach of complete search methods. The satisfiability (SAT) problem is of significant practical and theoretical interest as many application domains can be formulated in this way. The SAT problem consists of finding an assignment for the Boolean variables in a propositional formula that makes the formula true. Typically, local search methods for SAT work by iteratively changing (flipping) the value of one Boolean variable in the problem in order to minimise an evaluation function that maps any given variable assignment to the number of unsatisfied clauses under this assignment. This heuristic is followed 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 problem, thus dynamically changing the evaluation function and the search landscape as the search progresses. As a consequence, successful CWLS methods need efficient ways to adjust clause weights, so they can maintain the clause weight distribution relevant to the context in which they are searching. To this end, most methods can be divided into those that adjust weights multiplicatively, and those that do so additively. Multiplicative methods use floating point clause weights and increase/decrease multipliers that give the clause weight distribution a much finer granularity. Additive methods, on the other hand, assign integer values to clause weights and increase/decrease amounts, resulting in a coarser weight distribution.

Since their introduction [1, 2], several improvements have been proposed to CWLS methods, such as DLM [3] and SAPS [4]. While DLM uses additive weighting, SAPS adjusts clause weights multiplicatively. Recently, the pure additive weighting scheme (PAWS) was introduced [5] and shown to give significant performance improvements over SAPS on a range of challenging SAT problems from the SATLIB<sup>1</sup> and DIMACS<sup>2</sup> libraries, as well as on a set of SAT-encoded random binary CSPs from the phase transition region.

## 2 Tie Breaking and Search Landscapes

Consider a search landscape  $L$  for an instance  $\pi$ ,  $L(\pi) := (S, N, g)$ , where  $S$  is the space of all candidate solutions,  $N$  is a given neighbourhood relation, and  $g$  is an evaluation function. Now consider the following definitions of landscape position taken from [6]. For a position  $s \in S$  the following functions determine the number of upwards, sideways, and downwards flips from  $s$  to its direct neighbours<sup>3</sup>, respectively:  $upw := \#\{s' \in N(s) \mid g(s') > g(s)\}$ ,  $sidew := \#\{s' \in N(s) \mid g(s') = g(s)\}$ , and  $down := \#\{s' \in N(s) \mid g(s') < g(s)\}$ ; and the following landscape positions of interest:  $SLMIN(s) :\Leftrightarrow downw(s) = sidew(s) = 0$ , and  $LMIN(s) :\Leftrightarrow downw(s) = 0 \wedge sidew(s) > 0 \wedge upw(s) > 0$ .

It is well known that multiplicative methods encounter negligible numbers of tie breaking situations in their search landscapes [7, 5] due to the finer granularity of their clause weight distributions. Furthermore, by not taking equal-cost flips, multiplicative methods make no distinction between strict local minima ( $SLMIN$ ) and local minima ( $LMIN$ ), treating such positions as generic local minima. Specifically, SAPS randomly breaks ties amongst cost-improving flips, but if the candidate flips are either cost-increasing, or equal-cost, then it performs a random walk step with 1% probability by randomly selecting a variable for flipping from the domain of all variables in the problem.

<sup>1</sup> <http://www.satlib.org>

<sup>2</sup> <http://dimacs.rutgers.edu/Challenges/Seventh/PC>

<sup>3</sup> A neighbour  $s'$  of  $s$  is a position that differs from  $s$  on at most one variable assignment.

In contrast, additive methods such as DLM and PAWS tend to encounter significantly more *LMIN* positions in their landscapes, and hence having an efficient mechanism to deal with such positions appears to be of crucial importance for their performance. Additionally, they distinguish between *SLMIN* and *LMIN*, and execute an equal-cost flip strategy whenever the latter is encountered. Like SAPS, PAWS also randomly breaks ties amongst cost-improving and equal-cost flips, but in the equal-cost case it will take such a flip with 15% probability, otherwise it will adjust the clause weights<sup>4</sup>. It never takes a cost-increasing flip.

This work’s main motivation is to investigate an alternative way for breaking the ties frequently encountered in the landscapes of additive CWLS methods. After observing that a vast proportion of flips are randomly selected from a list of tied candidates, we hypothesise that the performance of these methods can be significantly enhanced by replacing the purely random tie breaking mechanism with one that also incorporates a heuristic that considers information about the landscape being searched. This paper reports on the implementation of this idea on a state-of-the-art additive CWLS method. We will demonstrate that our resulting method is able to significantly outperform the original one on a range of benchmark SAT problems.

### 3 Random versus Heuristic Tie Breaking

Figure 1 shows the PAWS method extended to accommodate heuristic tie breaking (HTB). Of interest here are its two parameters:  $P_{flat}$  (line 17) and  $WDP$  (line 25). The former controls the probability with which PAWS takes an equal-cost flip, while the latter determines the number of weight increases (line 24) allowed before a weight decrease takes place (lines 25-26). In practice only  $WDP$  has its value set on a problem-per-problem basis, whereas a  $P_{flat}$  setting of 0.15 was found to generally work well for all problems [5]. PAWS is suitable as a host method as it achieves state-of-the-art performance for satisfiable SAT instances, and because it represents the purest implementation of an additive CWLS method. As a result, the insights gained from this study are sufficiently general and thus can be of practical relevance for the design of new methods.

In a preliminary study only partially reported here, we developed twenty alternatives to PAWS’s purely random strategy for breaking ties of *equal-cost* candidate flips (Figure 1 lines 17-22). We chose to initially only deal with equal-cost flips for two reasons. Firstly, recent methods’ reliance on purely random tie breaking mechanisms to achieve current levels of performance has meant that little attention has been given to the investigation of novel heuristic-based alternatives for dealing with equal-cost flips. Secondly, as the setting of  $P_{flat}$  regulates the frequency of equal-cost flips, we were able to place an upper bound on the usage of candidate heuristics without having to introduce and tune a new parameter.

---

<sup>4</sup> Both SAPS’s random walk and PAWS’s equal-cost probabilities are parameters in the corresponding algorithms, but are in practice set to 0.01 and 0.15, respectively.

```

PAWS+HTB procedure
1. begin
2. generate random starting point
3. for each clause  $c_i$  do set clause weight  $w_i \leftarrow 1$ 
4. while solution not found and not terminated do
5.    $best \leftarrow \infty$ 
6.   for each literal  $x_{ij}$  in each false clause  $f_i$  do
7.      $\Delta w \leftarrow$  change  $\sum w$  in  $f_i$  caused by flipping  $x_{ij}$ 
8.     if  $\Delta w < best$  then  $L \leftarrow x_{ij}$  and  $best \leftarrow \Delta w$ 
9.     else if  $\Delta w = best$  then  $L = L \cup x_{ij}$ 
10.  end for
11.  if  $best < 0$  then
12.    if HTB and probability  $\leq 1 - P_{flat}$  then
13.      call BOCM procedure
14.    else
15.      randomly flip  $x_{ij} \in L$ 
16.    end if
17.  else if  $best = 0$  and probability  $\leq P_{flat}$ 
18.    if HTB then
19.      call TB procedure
20.    else
21.      randomly flip  $x_{ij} \in L$ 
22.    end if
23.  else
24.    for each false clause  $f_i$  do  $w_i \leftarrow w_i + 1$ 
25.    if # times clause weights increased %  $WDP = 0$  then
26.      for each clause  $c_i \mid w_i > 1$  do  $w_i \leftarrow w_i - 1$ 
27.    end if
28.  end if
29. end while
30. end

```

**Fig. 1.** The pure additive weighting scheme extended to accommodate heuristic tie breaking for equal-cost and cost-improving flips. This pseudo-code corresponds to the HTB method used in our empirical study.

We implemented each alternative tie breaking heuristic and tested the resulting method using a diverse problem set. The experimental conditions were identical to those reported later in the main part of this study. Control data was obtained by running the unmodified PAWS under the same conditions.

In order to estimate the performance variation arising from the introduction of each candidate heuristic, we computed the average median run-time for PAWS and each of the variants across all problems. A measure of variation in performance was then calculated as a percentage of PAWS's performance, and a ranking of candidate heuristics was obtained based on this measure. Scores for the best, median, and worst performing heuristics were 86.25%, 94.87%, and 110.18%, respectively. The main idea behind our best performing heuristic, TB (Figure 2(a)), lies on biasing the selection towards those candidates appearing in clauses that have been most rarely weighted during the search. For example, consider the CNF formula:  $(a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge (a \vee b \vee c) \wedge (a \vee \neg b \vee \neg c)$ , the corresponding clause weight  $C_w = \{3, 3, 3, 4\}$ , and number of weight up-

<pre> <b>TB procedure</b> 1. <b>begin</b> 2. <math>best \leftarrow \infty</math> 3. <b>for each</b> <math>x_{ij} \in L</math> 4.   <b>for each</b> true clause <math>c_i</math> in which <math>x_{ij}</math> appears 5.     <math>thisValue += \#WeightUpdates_{ci}</math> 6.   <b>end for</b> 7.   <b>if</b> <math>thisValue &lt; best</math> <b>then</b> 8.     <math>L' \leftarrow x_{ij}</math> and <math>best \leftarrow thisValue</math> 9.   <b>else if</b> <math>thisValue = best</math> <b>then</b> 10.    <math>L' = L' \cup x_{ij}</math> 11.   <b>end if</b> 12. <b>end for</b> 13. <b>if</b> <math>\#</math> candidate literals in <math>L' &gt; 1</math> <b>then</b> call BOCM 14. <b>else</b> return <math>x_{ij} \in L'</math> 15. <b>end</b> </pre>	<pre> <b>BOCM procedure</b> 1. <b>begin</b> 2. <math>best \leftarrow 0</math> 3. <b>for each</b> literal <math>x_{ij}</math> in <math>L'</math> <b>do</b> 4.   <b>if</b> <math>clauseMake_{ij} &gt; best</math> <b>then</b> 5.     <math>L'' \leftarrow x_{ij}</math> and <math>best \leftarrow clauseMake_{ij}</math> 6.   <b>else if</b> <math>clauseMake_{ij} = best</math> <b>then</b> 7.     <math>L'' = L'' \cup x_{ij}</math> 8.   <b>end if</b> 9. <b>end for</b> 10. randomly flip <math>x_{ij} \in L''</math> 11. <b>end</b> </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(a) TB.

(b) BOCM.

**Fig. 2.** TB and BOCM.

dates distributions  $C_{wu} = \{10, 20, 10, 10\}$ ; and a complete candidate solution  $S = \{a := false, b := true, c := false\}$ .

In this example, only the first clause is falsified, so when selecting a literal to flip, PAWS calculates  $\Delta w$  for each literal appearing in it (Figure 1 line 7):  $\Delta w = \{a := 0, -b := 0, c := 1\}$ . At this point, PAWS would then randomly break the tie amongst the two best candidates  $a$  and  $-b$  with 15% probability, given this is an equal cost flip.

However, if TB is used, then the tie breaking is biased towards the selection of those flips that appear in clauses that are (a) currently satisfied, and (b) have had their weight updated the least number of times during the search. Continuing with our example, the tie-breaker value for the two tied candidates is calculated (Figure 2(a) lines 4-6), resulting in the tie being broken in favour of  $a$ , as its tie-breaker value (20) is smaller than that of the other candidate (30). Although space limitations preclude us from discussing the other candidate heuristics in detail, we note that in some of the less successful variants the bias favoured the candidate flips with the *highest* tie breaking value. We also tested versions that took into account both satisfied and *unsatisfied* clauses in their tie breaking computation. If after using TB there was still a tie between  $a$  and  $-b$ , then BOCM (the procedure shown in Figure 2(b) for breaking ties on *clauseMake*<sup>5</sup>) would have been used and the tie would have been broken in favour of the candidate that, if flipped, would satisfy the greater number of clauses. Any remaining ties would be broken at random.

<sup>5</sup> *clauseMake* refers to a data structure introduced in the Walksat framework [8] and now commonly used in CWLS methods that counts how many clauses would be made true for any given variable flip.

For all heuristics tested, we observed that the number of flips resulting from heuristic tie breaking amounted to an average of approximately 3% of all search flips. This corresponds to a reduction of the same magnitude in the number of random flips performed by the host method. Therefore, the most striking finding of this preliminary study was that the reduction in the number of random flips resulted in a relatively larger improvement in overall performance when we consider our best performing heuristics.

Hence, we investigated the effects of allowing the *heuristic-to-random* flip ratio to increase by extending heuristic tie breaking to cost-increasing flips. In an initial attempt to implement this, the TB heuristic was used to break ties of cost-improving flips as well, with probability  $1-P_{flat}$ . However, the improvement in performance in terms of number of flips as observed for several problems compared poorly against the decrease in run-time performance brought about by the additional computational overheads. In other words, obtaining a further decrease in randomness in this fashion proved to be too expensive.

In an alternative implementation, BOCM was used in place of TB for cost-improving flips (maintaining the  $1-P_{flat}$  setting), while continuing to use TB for breaking ties of equal-cost flips. TB remained the chosen heuristic for equal-cost flips because when we tested BOCM as a standalone heuristic for this purpose, its performance was inferior to the performance obtained while using TB. Finally, as we observed that TB’s usage of BOCM was almost negligible at 0.02%, we decided to switch TB’s BOCM off in our resulting method.

As a result, we obtained a competitive method with which to test our hypothesis that an increase in the heuristic-to-random flip ratio can result in significant performance improvements for additive CWLS methods. We call this resulting method PAWS with heuristic tie breaking, or HTB, and use the empirical study reported next to compare its performance against the unmodified PAWS.

## 4 Empirical study

### 4.1 Problem Set and Parameter Setting

Our diverse test set draws problems from four different domains: uniform random 3-SAT, SAT-encoded graph colouring, parity learning, and planning. The -med and -hard instances from the original SATLIB sets flat100, flat200, uf100 and uf250 correspond to the median and hardest instances from these sets as found in a previous study [5]. 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, 3 sets of problems (400, 800 and 1600 variable sets) were generated from the 4.3 clause-to-variable ratio hard region. To these sets, the f400, f800, and f1600 problems from DIMACS were added and determined the median and hardest instances, resulting in the 6 random 3-SAT problems (f400, f800 and f1600 -med and -hard). A range of random binary CSPs (also from the accepted hard region) were generated and transformed into SAT instances using the multi-valued

encoding procedure described in [9]. These problems were divided into 4 sets of 5 problems each according to the number of variables ( $v$ ) the domain size ( $d$ ) and the constraint density ( $c$ ) from the originating CSP, which resulted in the 30v10d40c, 30v10d80v, 50v15d40c, and 50v15d80c problem sets from which the hardest problem in each set was chosen. We obtained three sets of balanced quasigroup with holes (BQWH) problems with orders 30, 33, and 36, sampled from the backbone phase transition region ( $number\ of\ holes/N^{1.55} = 1.7$ ) suggested in [10], and using the encoding method proposed in [11]<sup>6</sup>. We then selected the easy, median, and hard instances used in this study according to the number of flips (averaged from 20 runs) taken by PAWS to find a solution.

PAWS and HTB only require the  $WDP$  parameter to be set in practice, as  $P_{flat}$  can be treated as a constant and for all experiments reported here was set to the suggested 0.15. The PAWS  $WDP$  settings were taken from [12], except the settings for the BQWH problems, found according to the same thorough empirical evaluation used for finding the optimal HTB  $WDP$  settings for each problem.

## 4.2 Testing for Significance

Local search run-times on the same problem can vary greatly due to different starting points and subsequent randomised decisions. For this reason, empirical studies have traditionally reported statistics like mean, median and standard deviation obtained from many runs on the same problem 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 run-length distributions are usually not normally distributed, the non-parametric Wilcoxon rank-sum test can be used to measure the confidence level of these assertions. The test requires that the run-times or number of flips 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 used to obtain, using the normal approximation to the Wilcoxon distribution, the  $z$  value giving the probability  $P$  that the null hypothesis  $H_0 : A \geq B$  is true.

The Wilcoxon values presented below the mean time and number of flips for each problem 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 run-times (or number of flips) 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  $B$ . Significant performance difference is granted if the Wilcoxon test on run-times *and* number of flips is significant for  $P < 0.05$ . Using run-time and flips in combination allows us to capture any significant performance degradation in run-time arising from the introduction of the heuristic tie breaking that would otherwise be missed if we only used the Wilcoxon test on the distribution of flips.

---

<sup>6</sup> The authors would like to thank Duc Nghia Pham for generating these instances.

## 5 Results and Analysis

All statistics were obtained from 1,000 runs (100 runs for the bqwh-33-384-hard and bqwh-36-440-med and -hard) with a 20 million flip cut-off (50 million for 50v15d40c, and 250 million for all BQWH problems). We used a Sun supercomputer with 8 Sun Fire V880 servers, each with 8 UltraSPARC-III 900 MHz CPU and 8 GB memory per node.

	HTB					PAWS				
	WDP	% Solved	Mean Time (secs)	Flips	Random Flips %	WDP	% Solved	Mean Time (secs)	Flips	Random Flips %
Random 3-SAT										
uf100-hard	20	100.0	0.01	2,838	24.48%	15	100.0	0.01 (0.1339)	2,795 (0.2925)	51.01%
uf250-med	18	100.0	0.03	6,940	33.14%	15	100.0	0.02 (*0.0054)	5,604 (0.0029)	64.31%
uf250-hard	27	100.0	0.93 (*0.0000)	234,333 (*0.0000)	33.71%	18	100.0	1.26	319,366	64.67%
f400-med	17	100.0	0.16	38,007	36.36%	9	100.0	0.14 (*0.0447)	36,359 (0.2993)	61.62%
f400-hard	19	100.0	4.98 (*0.0084)	1,112,995 (*0.0024)	41.96%	11	100.0	5.95	1,363,348	71.35%
f800-med	15	100.0	0.90	180,202	47.30%	9	100.0	0.62 (*0.0000)	125,647 (*0.0000)	74.31%
f800-hard	17	100.0	8.27	1,801,078	40.38%	10	100.0	4.99 (*0.0000)	1,089,711 (*0.0000)	68.59%
f1600-med	18	100.0	4.56	859,664	45.37%	10	100.0	2.28 (*0.0000)	461,533 (*0.0000)	69.07%
f1600-hard	17	97.8	18.61 (*0.0000)	3,795,765 (*0.0000)	42.79%	11	98.1	29.59	4,752,810	79.20%
Random Binary CSPs										
30v10d80c	12	100.0	0.10	11,808	39.14%	7	100.0	0.08 (*0.0010)	10,323 (*0.0076)	59.94%
30v10d40c	9	100.0	0.19 (0.3219)	23,913 (0.3671)	39.54%	5	100.0	0.19	24,130	64.11%
50v15d80c	9	100.0	2.88	195,496	43.93%	7	100.0	2.24 (*0.0000)	155,804 (*0.0000)	64.61%
50v15d40c	8	98.6	178.36 (*0.0000)	11,406,521 (*0.0000)	46.82%	6	95.4	216.33	13,556,479	72.73%

**Table 1.** Random instances.

An initial inspection of Tables 1 and 2 reveals two interesting results. Firstly, HTB was significantly better on sixteen problems, whereas PAWS gave significantly better performance on five. Secondly, all five problems where PAWS was superior are randomly generated instances shown in Table 1. Should our analysis be restricted to the arguably more relevant domain of structured encodings shown in Table 2, then HTB is significantly better in twelve problems, whereas PAWS fails to give significant performance improvements over HTB on any of the twenty-six structured problems. Our results also show the number of random flips as a percentage of the total number of flips, performed by each method on each problem. HTB used approximately 18% less random flips, on average, across the whole problem set.

These observations indicate that the reduction in the number of random flips introduced by HTB may not be as useful for random problems, i.e., randomly generated search landscapes may be more efficiently searched by a method that utilises proportionally more random flips. However, the significant performance improvement afforded by the heuristic tie breaking method on the two

	HTB					PAWS				
	WDP	% Solved	Mean Time (secs)	Flips	Random Flips	WDP	% Solved	Mean Time (secs)	Flips	Random Flips %
Blocks World										
bw_large.a	35	100.0	0.02	3,133	29.66%	34	100.0	0.02	3,066	52.64%
			(0.3835)	(0.2809)						
bw_large.b	60	100.0	0.31	43,163	29.10%	50	100.0	0.30	41,985	56.16%
			(0.4179)	(0.4336)						
bw_large.c	6	100.0	7.71	898,743	49.82%	5	100.0	10.06	1,211,756	66.00%
			(*0.0000)	(*0.0000)						
bw_large.d	5	100.0	13.88	1,270,084	56.76%	4	100.0	16.90	1,501,005	72.47%
			(*0.0001)	(*0.0014)						
AIS, Logistics, Par										
ais10	70	100.0	0.12	17,595	25.64%	52	100.0	0.15	20,211	41.25%
			(*0.0195)	(*0.0319)						
ais12	80	100.0	1.08	123,640	28.14%	148	100.0	1.12	137,461	36.11%
			(0.1204)	(*0.0060)						
logistics.c	∞	100.0	0.04	6,375	32.47%	∞	100.0	0.05	6,676	51.22%
			(*0.0001)	(*0.0042)						
logistics.d	∞	100.0	0.26	20,263	40.43%	∞	100.0	0.27	21,531	60.54%
			(*0.0148)	(*0.0001)						
par16-2-c	39	98.9	15.95	4,051,732	66.74%	36	98.5	18.13	4,845,096	76.40%
			(*0.0003)	(*0.0000)						
par16-3-c	39	99.4	15.26	3,879,903	66.29%	40	98.9	14.10	3,711,505	77.26%
								(0.1112)	(0.2967)	
Graph Colouring										
flat100-med	20	100.0	0.02	7,699	45.51%	16	100.0	0.03	9,462	69.91%
			(*0.0000)	(*0.0000)						
flat100-hard	35	100.0	0.10	33,940	47.34%	46	100.0	0.10	34,822	63.08%
				(0.3395)				(0.3807)		
flat200-med	16	100.0	0.46	134,334	56.66%	9	100.0	0.57	183,618	74.82%
			(*0.0016)	(*0.0000)						
flat200-hard	78	100.0	10.80	3,270,296	47.94%	74	99.8	10.52	3,261,605	66.15%
				(0.3608)				(0.4595)		
g125.17	10	100.0	12.87	674,462	54.92%	4	100.0	12.78	737,912	66.36%
			(0.1190)	(*0.0008)						
g250.29	11	100.0	33.17	324,516	62.29%	4	100.0	30.23	353,746	70.98%
				(*0.0056)				(*0.0046)		
QWH										
bqwh-30-332-easy	4	100.0	1.69	292,089	59.10%	3	100.0	1.92	347,602	67.75%
			(*0.0207)	(*0.0027)						
bqwh-30-332-med	5	100.0	15.18	2,317,550	66.02%	4	100.0	16.08	2,509,285	74.09%
			(0.2773)	(0.1618)						
bqwh-30-332-hard	5	100.0	118.06	17,301,209	67.46%	4	100.0	122.50	18,534,584	74.86%
			(0.1442)	(*0.0485)						
bqwh-33-384-easy	5	100.0	23.62	3,672,471	64.97%	4	100.0	24.72	3,970,927	73.05%
			(0.1262)	(*0.0399)						
bqwh-33-384-med	4	100.0	55.66	10,199,421	58.00%	4	100.0	83.14	11,719,052	75.67%
			(*0.0000)	(*0.0001)						
bqwh-33-384-hard	5	98.0	301.25	39,903,182	69.81%	4	98.0	371.90	50,428,307	76.75%
			(*0.0494)	(*0.0366)						
bqwh-36-440-easy	4	100.0	41.73	6,857,439	61.91%	3	99.8	54.13	9,566,874	69.56%
			(*0.0000)	(*0.0000)						
bqwh-36-440-med	4	100.0	286.585	45,866,880	63.24%	3	100.0	339.58	59,211,063	72.12%
			(0.1925)	(0.0929)						
bqwh-36-440-hard	4	80.0	560.55	91,481,218	62.01%	3	66.0	608.76	105,869,465	70.18%
			(0.1464)	(*0.0065)						

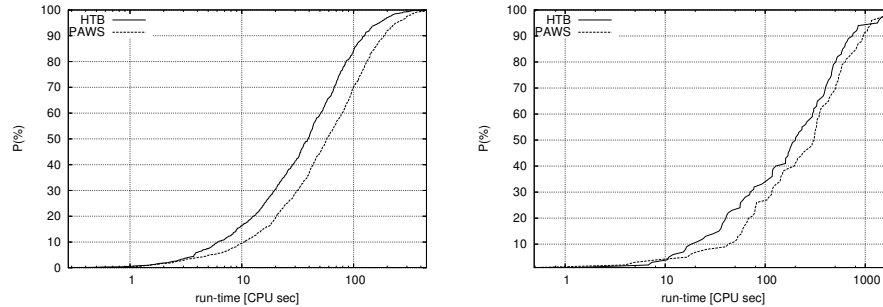
Table 2. Structured instances.

largest random problems, 50v15d40c and f1600-hard, suggests that in addition to problem structure, problem size also plays a significant part in determining the usefulness of heuristic tie breaking for additive CWLS methods.

For all but three problems in our test set, heuristic tie breaking resulted in an increase in the optimal setting of *WDP*. Therefore, the additional computational overhead caused by the introduction of heuristic tie breaking is compensated by the less frequent clause weight updates. Whether these higher settings are indicative of a desirable increase in the robustness of the *WDP* settings is not yet clear to us, and is therefore an issue that requires further investigation.

When comparing performance, it is also relevant to show that one method can frequently give higher solution probabilities than the other. Such probabilis-

tic domination can be ascertained by visually inspecting a plot of the run-time distribution (RTD) data for the different methods on any given problem. If these RTDs do not cross then the method whose RTD appears to the left of the other probabilistically dominates for *all* solution probabilities [6]. In practice, we normally relax this definition to allow for some crossing for low solution probabilities, i.e.,  $P \leq 0.1$ . The RTD in Figure 3 (left) is an example of probabilistic domination for all probabilities  $P > 0.05$ .



**Fig. 3.** Run-time behaviour analysis of HTB and PAWS using the RTDs for problems bqwh-33-384-med (left) and -hard (right).

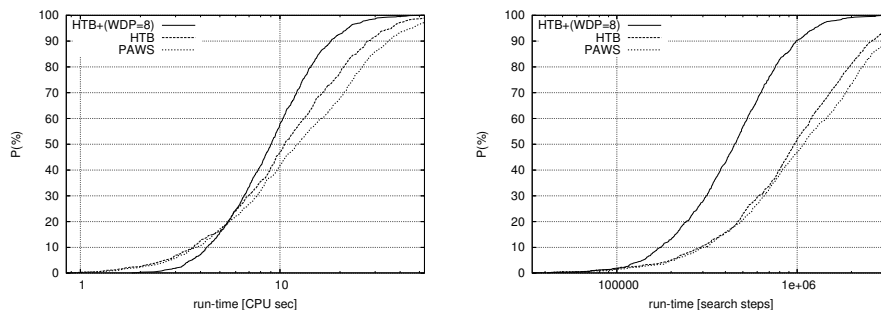
Run-time and run-length distribution (RLD) plots are also useful for detecting erratic run-time behaviour. For example, the RTD in Figure 3 (right) reveals that although HTB dominates PAWS for all  $0.05 \leq P < 0.95$ , the crossing of the distributions at  $P = 0.95$  indicates that PAWS was faster on the 3 longest runs sampled. This could be a symptom of search stagnation on HTB’s behalf. Stagnation invariably arises from poor search diversification. As additive methods rely on the strong use of random flips to diversify their search [13], any changes likely to result in a substantial reduction in the number of random flips must also consider adjustments to the method’s diversification strategy. In our host method, effective diversification relies on an adequate equal-cost flip strategy. Consequently, we see two possible ways to address this problem in the HTB method. One is to fine-tune the setting of  $P_{flat}$ . Another, to replace PAWS’s equal-cost strategy with the random-walk mechanism typically found in implementations of less randomised multiplicative CWLS methods. We continue to work on this problem and intend to report on our findings in forthcoming work.

As pointed out earlier, we initially considered using the TB heuristic for breaking the ties of cost-improving flips as well, but opted for using BOCM because it offered less computational overheads while still delivering a method suitable for the goals of our study. However, given HTB’s performance in our empirical evaluation, we became interested in investigating whether we could obtain further gains by handling tied cost-improving flips with a more sophisti-

cated heuristic. Using HTB, we replaced BOCM with TB for handling equal-cost flips, naming the resulting method HTB+.

Figure 4 shows the performance of this method on the largest blocks world problem. As expected, the run-time performance (Figure 4 (left)) is impaired by the additional computational overheads, although this is only the case for solution probabilities below 25%. For higher solution probabilities (i.e. longer runs), this difference is noticeably outweighed by the method’s superior performance.

An RLD analysis (Figure 4 (right)) shows that HTB+ clearly dominates both HTB and PAWS for all solution probabilities above 5%, solving 80% more instances than either of the other two methods within 1 million flips. Similar results were also observed on the RTDs and RLDs for the `bw_large.c` problem, as well as on several BQWH instances. This evidence suggests that it is worthwhile



**Fig. 4.** RTD and RLD analysis of PAWS, HTB, and HTB+ on `bw_large.d`, 1,000 runs.

investigating the use of more sophisticated heuristic tie breaking techniques, although a more thorough investigation in this direction is needed before any conclusive findings can be reached.

## 6 Conclusion

This study advances our understanding of the usefulness of heuristic tie breaking for the performance of constraint weighting local search methods. We have used an empirical study to demonstrate that when heuristic tie breaking is used in combination with the random tie breaking mechanism of an existing state-of-the-art solver, the resulting method gives significant performance improvements, especially for the large and structured search spaces.

Our proposed approach is conceptually simple, and does not require the use of any additional parameters. Our findings therefore serve to motivate the adoption of heuristic tie breaking as a worthwhile avenue to be explored for the further development of additive CWLS methods.

## References

1. Morris, P.: The breakout method for escaping from local minima. In: Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93), Washington, D.C., MIT Press (1993) 40–45
2. Selman, B., Kautz, H.: Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'93), Chambéry, France, Morgan Kaufmann (1993) 290–295
3. Wah, B., Shang, Y.: Discrete lagrangian-based search for solving MAX-SAT problems. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97), Nagoya, Japan, Morgan Kaufmann (1997) 378–383
4. Hutter, F., Tompkins, D., Hoos, H.: Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Proceedings of CP-02. Volume 2470 of LNCS., Ithaca, New York, Springer Verlag (2002) 233–248
5. Thornton, J., Pham, D.N., Bain, S., Ferreira Jr., V.: Additive versus multiplicative clause weighting for SAT. In: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'04), San Jose, California, MIT Press (2004) 191–196
6. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, San Francisco, California (2005)
7. Schuurmans, D., Southey, F.: Local search characteristics of incomplete SAT procedures. In: Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'00), Austin, TX, MIT Press (2000) 297–302
8. Selmann, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94), Seattle, WA, AAAI Press (1994) 337–343
9. Prestwich, S.: Local search on sat-encoded CSPs. In: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), Portofino, Italy, Springer (2003)
10. Achlioptas, D., Gomes, C., Kautz, H., Selman, B.: Generating satisfiable problem instances. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00), Austin, TX, MIT Press (2000) 256–261
11. Kautz, H., Ruan, Y., Achlioptas, D., Gomes, C., Selman, B., Stickel, M.: Balance and filtering in structured satisfiable problems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle, Morgan Kaufmann (2001) 351–358
12. Thornton, J.: Clause weighting local search for SAT. *Journal of Artificial Intelligence Research* (to appear)
13. Tompkins, D., Hoos, H.: Warped landscapes and random acts of SAT solving. In: Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics - AMAI, AI&M 2004, Fort Lauderdale, Florida (2004)