Multi Level Weight Distribution Dynamic Local Search for SAT

Abdelraouf M. Ishtaiwi*, Marco J. Baron+, and Ghassan F. Issa*
*University of Petra, Faculty of Information Technology, Amman, Jordan
{aishtaiwi, gissa}@uop.edu.jo
+ Pedagogical and Technological University of Colombia Tunja-Boyaca, Colombia marco.suarez@uptc.edu.co

Abstract**:** *In recent years, dynamic clause weighting stochastic local search algorithms have emerged as the-state-of-the-art for solving satisfiability problems. In our study we experimentally investigated the weights behaviors and movements during searching for satisfiability. Firstly, We show that, second level neighboring clauses could be the culprit of causing the delay during the search. Secondly, we developed and experimentally investigated a new heuristic for enhancing Dynamic Local Search known as Multi Level Weight distribution (mulLWD). Our new heuristic is divided into two main phases. Phase one is to distribute weights within the same neighboring area. When phase one is no longer effective, phase two take place in which the second level neighboring area is exploited and its weights are used within the weight distribution process.*
To test our new heuristic, we conducted a detailed experimental study, comparing mulLWD with one of the leading clause weighting algorithm: Divide and Distribute Fixed Weight (DDFW). Experimental results showed that mulLWD heuristic has significantly better performance than DDFW (of up to an order of magnitude). This enhancement indicates that the multi level weight distribution is the key to further exploitation of local search SAT structures. Moreover, mulLWD could be easily generalized to be used with any dynamic local search weighting heuristic with some minor modifications

Keywords: *DLS, Local Search, mulLWD, SAT, Search Heuristics*

1. Introduction

The boolean satisfiability problems (also known as propositional satisfiability or SAT for short) are very crucial to many areas in computer science and aritficail intelligence. Thus, to find a complete assignment to a SAT problem has much deep signification. In our research we consider propositional conjunctive normal form formulas (CNF): F = $\wedge$m $\vee$n lmn in which each lmn is a literal (boolean variable or its negation), and each disjunct $\vee$n lmn is a clause. The task of any given search algorithm is to find a complete assignment that satisfies F . This task is beyond the reach of systematic search algorithms except for a limited small sized problems as SAT problems is NP complete. On the other hand, almost any simple stochastic local search (SLS) method could successfully solve a broad range of larger and challenging problems [5].

Since the development of the first SLS weighting algorithm for SAT, namely the Breakout heuristic [10], tries were made to enhance the performance of local search techniques ([2], [3]). However, the performance of these algorithms remained weak when compared to non-weighting SLS techniques. Which remained the case until the occurrence of Dynamic Local Search algorithms (DLS) such as DLM [13], SAPS [6], PAWS [12], DDFW [8] and more recently BalancedZ [9].

DLS algorithms depend on the use of weights to alter the search space in order to escape from local minima (or in some heuristics could prevent encountering local min- ima). Interestingly, the most successful DLS algorithms (i.e. DLM, SAPS, PAWS, EWS [1], COVER [11], DDFW, and recently, and BalancedZ [9]) follow the same underlying platform at which they increase weights of false clauses when the search get into local minima, then weights are reduced based on scoring functions that are calculated during the search. For instance, the scoring function used with most of DLS algorithms is as follows

$$score = Makes - Breaks \quad (1)$$

where the Makes is the number of clauses that will become true when literal l_{mn} flipped (to true if its current value is false or to false if its current value is true) and the Breaks is the number of clauses that will become false when l_{mn} is flipped. Except for the probSAT algorithm which does not use weights to

alter the search space but it depends on the score of the number of clauses that will become false (breaks) when a literal l_{mn} is flipped while in local minimum.

This study has emerged out of studying an algorithm called DDFW (Divide and Distribute Fixed Weights) which in turn, evolved out of Pure Additive Weighting Scheme (PAWS). PAWS retain the method of periodically smoothing weights after a number of weight increments with a difference that it uses an incremental function rather a multiplicative function as in Scaling and Probabilistic Smoothing (SAPS). It is worth mentioning that another methods older than PAWS used additive instead of multiplicative weight adjustment called DLM. The difference between PAWS and DLM is that PAWS has one parameter instead of 27 parameters (in the case of DLM it contains 27 parameters but only three need tuning) of DLM. Also, PAWS utilize a random flat moves. DDFW also while it retaining the additive weighting framework used by PAWS, it has a unique characteristic of combining the two main functions of weight adjustments in PAWS (weights increment, and weight decrements) into one function. Also DDFW is fully adaptive method and is a domain independent method that requires no parameter tuning.

DDFW was shown to outperform PAWS, SAPS, AdaptNovelty+ [4] in a wide range of problems, and was effective in handling weights for a wide range of hard satisfiability problems. However, In recent study [7], suffer from weight stagnation while distributing the weights when applied to large problems due to the complexity and hardness of these problems.

As previously mentioned, experiments show that DDFW could suffer from weight stagnations during the search steps. Among the factors that could cause such stagnation is that weights could be hold within the same neighborhood for longer than the search required. Thus, our current investigation of weight movements in DDFW addresses the question of whether there are an alternative method to handle weights to further achieve gains in the domain of SAT. In particular, we are interested in multi level weight distribution schemes, that exploit weighted clauses in multi levels fashion so that the problem of weight stagnation is prevented or dealt with when it occurs. The multi level weight distribution approach offers the advantage of exploiting weights that are indirectly connected. Thus, the limitation of weight distribution from within the same neighboring area where clauses share a variable or variables is avoided. Such an approach offers the a variation of weight distribution from other neighboring areas within the search domain. As a result the high degree of deterministic weight distribution from within the same neighboring area is relaxed without compromising the benefits gained by the original approach of one level weight distribution.

In the next section we provide a general background on the evolution of clause weighting algorithms. Then, provide further details of weights movements within the first level neighboring area (which is implemented in DDFW). We then introduce the mulLWD in more details, and provide an experimental comparison between mulLWD and its ancestor DDFW and some other clause weighting algorithms. Consequently, a significant category of problems is identified where mulLWD has remarkably better performance. Then we conclude our work by recommendations for the use and further improvements of multi level weight distribution local search for SAT.

2. Clause Weighting for SAT

In general, clause weighting local search heuristics for SAT repeatedly follow the same scheme. That is they start with a simple process of initializing randomly all literals of a given problem (given a boolean value for each literal $\in$ 0, 1), and all clauses get the same amount of weight at the beginning of the process. Then the search starts by changing the value of a literal (from 0 to 1 or vice versa). This changing of literal value must cause a reduction of the overall number of false clauses count that appear in the current stage of the search. When there are no literals that can cause the reduction of the sum of false clauses count by changing its boolean value, the search at this point increase the weights of all unsatisfied clauses. The search continue and periodically reduce the weights on the weighted satisfied clauses.

Basically there are two main methods on deciding on when to adjust the weights. This decision is a key factor that distinguishes the weighting heuristics from each others. Some heuristics use a multiplicative method such as SAPS. Some other heuristics adjust weights additively such as PAWS, DLM, DDFW, and BalancedZ.

2.1. Multi Level Weight Distribution

MulLWD heuristic uses the two uniquely implemented ideas that are built in DDFW. Firstly, both algorithms evenly distributes fixed amount of weights across all clauses at the beginning of the search process, escaping from the traps of local minima is performed by the weight transfer from satisfied clauses to unsatisfied (false clauses). The state-of-the-art-idea is that the weights increments and decrements are done in implicitly one step. That is, a neighboring satisfied clause will donate its weights (because weights are no longer needed since the clause has been satisfied) to unsatisfied (false) clause. In other words, weight normalization (decrements of satisfied clauses) is a sub-function of the weight increment step. This idea crucially important in the weight increment process as it waive the need of deciding at which point weights reduction should be performed. Other local search heuristics such PAWS, SAPS, BalancedZ, SDF, ESG, DLM, all use a separate step to decide at which point weights normalization (weights decrements) is performed. This separate step has been proven costly and it needs separate parameters tuning which consume tremendous amount of time. Second and more original idea is the exploitation of false clause neighboring area in order to search for weights donors.

Aside from the similarities between mulLWD and DDFW, mulLWD differ significantly in the way it picks a neighboring satisfied clause to be a weight donor to a false clause as discusses in the next subsection.

2.2. Exploiting 2nd Level Neighborhood Structure

A neighboring clause c_S to clause c_u is defined as: if there exists at leas a literal l_X that is $\in c_S$, and $\in c_u$. Furthermore, we term l_X a same sign literal in all clauses that l_X occur in, which in turn implies that its negation literal is $-l_X$. As a result, we term any two clauses $\in \mathbf{F}$, c_i and c_j neighbors if literal l_X is $\in$ clause c_i and clause c_j where $i \neq j$. Consequently, if clause c_i is false it means all literals l_{iX} $\in$ clause c_i evaluate to false (the boolean value of all literals $\in$ clause c_i = 0, or 1 if the literal is negated). Thus, flipping literal l_X (the boolean value of literal c_{iX}) will help clause c_{iX} and all its false neighbors. On the other hand, if clause c_n has a literal $\neg l_X$ and its current value is 0 then if we change the boolean value of literal l_X to 1 will make literal $\neg l_X$ evaluate to 0. Assuming that literal $\neg l_X$ was the only literal in clause c_n that evaluates to 1 before changing its value then clause c_n will be damaged by changing the value of literal l_X from 0 to 1 see Fig 1. Basically, if literal l_X occur in clause c_i, and clause

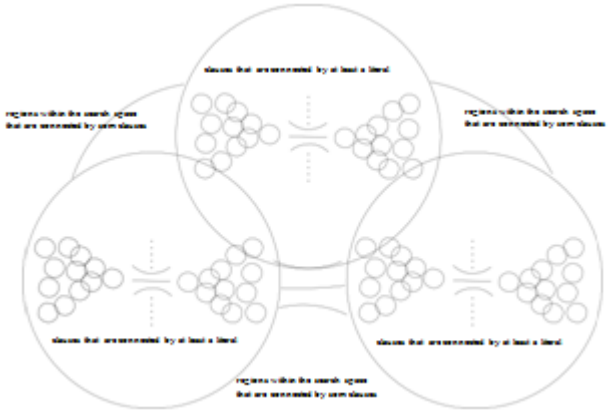

Fig. 1. illustration of neighboring areas of clauses

ci has another literal ly which occur in a subset of clauses S where literal lx $\neq$ ly then all the clause $\in$ S are a second level neighboring clauses of clause ci . The second level neighborhood area or (”neighbors of a neighbor”), came to life while observing the exploitation of the above first level neighborhood structure. To clarify our observation we reported in Fig 2 the weights movements within neighboring clauses. These weights movement are deterministic moves. Now, in case there is no deterministic move that could be taken, a random move (transferring weights in random fashion from outside the neighboring areas) is chosen. As illustrated in Fig 2, the random moves occur very frequently while searching for a solution. We assumed two

scenarios that may cause the search to pick a random clause, to be weight donor, from outside a false clause neighboring area:

- the neighboring clauses are all false.
- the neighboring clauses do not have enough weights to be transferred to their false neighbor.

We experimentally tested the first scenario, the results indicated that it is very unlikely to be the case so we excluded it. On the other hand, scenario two occurred in all the experiments that were conducted on our problem set as illustrated in Fig 3. The Figure shows the number of first level weight movements, the second level weights movements and the random weight movements. From that we assumed that if weights to be transferred from a second level neighboring area will improve the performance of the search process, which was the case as discussed in the section III.

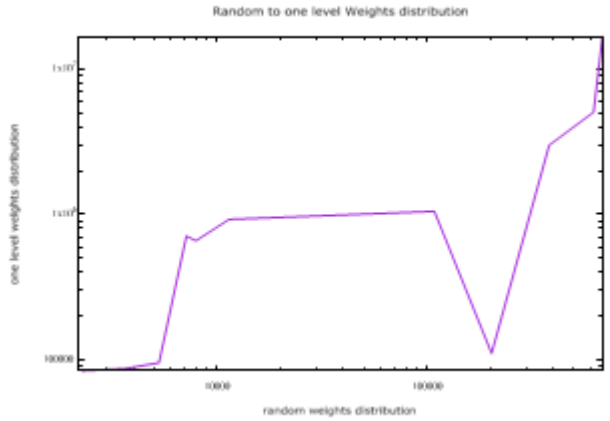

Fig. 2. illustration of search moves: random moves and one level deterministic moves

3. Experimental evaluation and analysis

As mentioned in the previous section, we empirically tested whether searching for a satisfying assignment that could evaluate $\mathbf{F}$ to true may be faced with the second scenario at which there is no single satisfied clause that is willing to give weights to its neighboring false clauses. Given the fact that the search would pick a clause randomly only when the satisfied neighboring clauses of a false clause do not have enough weights to donate (see algorithm 1), we designed our experiments to firstly report whether the weights movements are deterministic (first level weight distribution) or random (from outside the neighboring area of a false clause) as

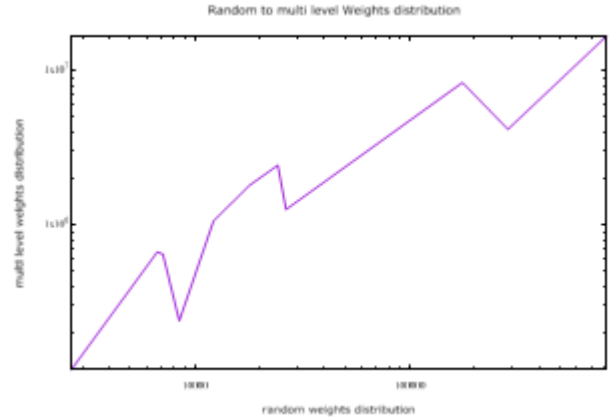

Fig. 3. illustration of search moves: random moves and multi level deterministic moves

Algorithm 1 mulLWD($\mathbf{F}$)

```
1: assign a boolean value ∈ 0, 1 to each literal in F;
2: set the weight of clauses in F = W_init;
3: while solution is not found and not timeout do
4:     find and return a list L of literals causing the greatest reduction in
                                        weighted cost Δw when flipped;
5:     if (Δw < 0) or (Δw = 0) then
6:         flip a literal randomly in L;
7:     else
8:         for each false clause c_f do
9:             find a satisfied clause neighbouring clause c_k with the highest
                                                    weight w_k;
10:            if C_K found then
11:                move a weight of one from c_k to c_f;
12:            else
13:                select a satisfied same sign neighbour of neighbouring
                                                clause c_k with maximum weight w_k;
14:            end if
15:            if w_k < Init then
16:                randomly select a clause c_k with weight w_k ≥ W_init;
17:            end if
18:            if w_k >= W_init then
19:                transfer a weight of one from c_k to c_f;
20:            end if
21:        end for
22:    end if
23: end while
```

illustrated in Table I.

The random moves reported in Table I confirm that, in many search steps random moves is chosen which support our assumption. For instance, we have noticed that in all of our problem set, random occurs very frequently. The ratio of random moves to deterministic moves varies depending on the size of the problem, and in some cases the problem complexity. For instance, problem ais10 random moves count was 11,437, while the search picked a neighboring satisfied clause in 928,457, which is a ratio of almost 82% of the over all weights moves. That implies, the search had to pick a satisfied clause from outside the neighbouring area of a false clause with a ration of 18%. This was also the case with small to medium sized problems. For the larger problems, we can see the number of random weight moves is much less. This is due to the fact that, large problems have much more clauses and connectivity than smaller problems, therefore the search in most cases would find a neighbouring satisfied clause in the neighbourhood of a false clause, so no random moves is selected. This is clearly appear in the case of problem uni-k6-r43.37-v135.

TABLE I
Experimental results that show the count of random weights moves, and the count of deterministic moves

Problem	DDFW		MulLWD		
	randPick	DtrPick	randPick	1stlevel	2ndlevel
ais10	11437	928457	7109	644419	1906
uf400-hard	108632	1053788	17910	1408277	390409
f800-hard	623753	5055439	6684	508460	160837
f1600-hard	383177	3000368	286299	3913840	244149
bw_large.c	3648	87143	8463	239620	303
bw_large.d	2287	84444	2653	115438	70
flat200-hard	203556	111172	175883	341382	8015928
logistics	7950	660082	12194	1060692	49
g125.18	5307	95541	26581	1260285	63
unif-k6-r43.37-v130	685888	16777216	826258	16777216	0
unif-k6-r43.37-v135	7157	712198	24309	2435869	0

TABLE II
Comparison results of DDFW, BalancedZ and MulLWD. For each algorithm the time and the % solved is reported.

Problem	DDFW		BalancedZ		MulLWD	
	time	%solv	time	%solv	time	%solv
ais10	72.80	100	29.03	100	42.31	100
uf400-hard	70.06	100	76.29	100	30.21	100
f800-hard	222.11	97	95.91	100	171.76	100
f1600-hard	1001.86	95	186.95	100	405.54	100
bw_large.c	32.89	100	Timed-out	0	28.13	100
bw_large.d	55.18	100	Timed-out	0	44.72	100
flat200-hard	88.93	100	682.12	100	27.80	100
logistics	40.80	100	341.14	100	42.61	100
g125.18	31.72	100	Timed-out	0	24.67	100
unif-k6-r43-v130	14,747	93	Timed-out	0	11,118	100
unif-k6-r43-v135	6371	100	44.32	91	3847	100

Based on the above observation, we designed the next experimental phase at which we ran the first level weight distribution algorithm (DDFW), the multi level weight distribution (mulLWD) and we have also included in our runs BlanacedZ. BalansedZ starts with generating a random instantiation of the given problem, all the weights of the clauses are assigned a value of one nit of weight. Then, the search starts with a target of reaching a zero value of the sum of weights of the unsatisfied clauses. The score of a variable is the amount of weights of the unsatisfied clauses reduction, if the variable is flipped. If after flipping a variable, the weights of the unsatisfied clauses is not zero then BalancedZ flip a literal that will cause the major reduction, if no such variable is found then the variable with the highest score is flipped. If no variable is found a random flip is chosen that is least recently flipped. Balancedz algorithm was the winner the gold medal of The International SAT solver competition (***http"//www.satcompetition.org/***) in the year 2014. The results are reported in Table II.

Overall, the problem set is designed to show how mulLWD compares in absolute terms to the other algorithms and to examine the relative effect of the multi level weight distribution mechanism on different problem classes. For this reason we also include the results for DDFW. All experiments were performed on a iMAC machine with i5 multiCore 2.5. GHz CPU and 8GB memory. Cut-offs for the various algorithms were set as follows: for the three algorithms, DDFW, BalancedZ and mulLWD the cutoff was set 50,000,000 on all the problem set. All the algorithms were allowed 100 tries on each problem. For each run the *time* and % of solution, if found, is reported.

Overall we can conclude that the addition of a multi level weight distribution has shown a significant enhancements over the first level weight distribution over the entire range of the problem sets we have considered (except for the *logistics* problem where DDFW and mulLWD performed similarly). on some problems mulLWD performed better by an order of two as the case with *ais*10, $f800-hard$ and $uf400$. The strongest dominance was observed on the $f1600-hard$where mulLwd performed better by an order of magni- tude. Also, mulLWD was twice better than DDFW on $flat200-hard$ and $unif-k6-r43.37-v135$. The mulLWD also was superior to BalancedZ on 70% of the runs. It performed twice better on $uf400$ and the strongest performance was observed on the *bw−c*,*bw−d*, *logistics*, $flat200$, $g125.18$, $unif-k6-r43.37-v130$ and $unif-k6-r43.37-v135$. On the other hand, BalancedZ performed better than DDFW and mulLwd on the problems *ais*10, $f800-hard$, and $f1600-hard$.

4. Acknowledgements

The authors would like to acknowledge the financial support of the Scientific Research Committee at Petra University. Also we would like to thank all faculty members of the Information Technology faculty who contributed directly and indirectly to this work.

5. References

[1] S. Cai, K. Su, and Q. Chen, "EWLS: A new local search for minimum vertex cover," in Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010, 2010.

[2] B. Cha and K. Iwama, "Adding new clauses for faster local search," in Proceedings of 13th AAAI, 1996, pp. 332–337.

[3] J. Frank, "Learning short-term clause weights for GSAT," in Proceedings of 15th IJCAI, 1997, pp. 384–389.

[4] H. Hoos, "An adaptive noise mechanism for WalkSAT," in Proceedings of 19th AAAI, 2002, pp. 655–660.

[5] H. Hoos and T. Stulze, Stochastic Local Search. Cambridge, Massachusetts: Morgan Kaufmann, 2005.

[6] F. Hutter, D. Tompkins, and H. Hoos, "Scaling and Probabilistic Smoothing: Efficient dynamic local search for SAT," in Pro- ceedings of 8th CP, 2002, pp. 233–248.

[7] A. Ishtaiwi, "Weight stagnation in dynamic local search for sat," in Proceedings of 3rd International Conference on Artificial Intelligence and Applications AIAP, 2016, pp. 79–86.
https://doi.org/10.5121/csit.2016.60607

[8] A. Ishtaiwi, J. Thornton, A. Sattar, and D. N. Pham, "Neigh- bourhood clause weight redistribution in local search for SAT," in Proceedings of 11th CP, 2005, pp. 772 – 776.

[9] C. Li, C. Huang, and R. Xu, "Balance between intensification and diversification: two sides of the same coin," in Proc. of SAT Competition 2013, Solver description, pp. 10–11, 2013.

[10] P. Morris, "The Breakout method for escaping from local minima," in Proceedings of 11th AAAI, 1993, pp. 40–45.

[11] S. Richter, M. Helmert, and C. Gretton, "A stochastic local search approach to vertex cover," in KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany, September 10-13, 2007, Proceedings, 2007, pp. 412–426.
https://doi.org/10.1007/978-3-540-74565-5_31

[12] J. Thornton, D. N. Pham, S. Bain, and V. Ferreira Jr., "Additive versus multiplicative clause weighting for SAT," in Proceedings of 19th AAAI, 2004, pp. 191–196.

[13] Z. Wu and B. Wah, "An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems," in Proceedings of 17th AAAI, 2000, pp. 310–315.

Abdelraouf M. Ishtaiwi Abdelraouf Ishtaiwi received the M.S degree (in 2001) & Ph.D. degree (in 2008) both in Information & Com- munication Technology from Griffith Univer- sity, Brisbane, Australia. Currently, he is an Assistant Professor at University of Petra in Amman, Jordan. His interests are in artificial intelligence, distributed systems & handling BigData

Marco J. Baron Marco J. Baron is re- searcher at UNITEC University from Bogota- Colombia. He was born in Duitama Colom- bia. His works focused on machine learning, semantic web and knowledge discovery. He hold a Ph.D. in Strategic and Management technology.

Ghassan F. Issa Ghassan Farid Issa is the Dean of Information Technology Faculty at University of Petra in Amman, Jordan. He re- ceived his BS degree in Electrical engineering from University of Toledo in 1983, and the BS in Computer engineering from Trine Uni- versity, Indiana in 1984. Ghassan Issa received his M.S and PhD in Computer Science from Old Dominion University, Norfolk Virginia in 1987 and 1992.