WEIGHTS STAGNATION IN DYNAMIC LOCAL SEARCH FOR SAT

Abdelraouf Ishtaiwi

Faculty of Information Technology, University of Petra, Amman, Jordan
faishtaiwig@uop.edu.jo

ABSTRACT

Since 1991, tries were made to enhance the stochastic local search techniques (SLS). Some researchers turned their focus on studying the structure of the propositional satisfiability problems (SAT) to better understand their complexity in order to come up with better algorithms. Other researchers focused in investigating new ways to develop heuristics that alter the search space based on some information gathered prior to or during the search process. Thus, many heuristics, enhancements and developments were introduced to improve SLS techniques performance during the last three decades. As a result a group of heuristics were introduced namely Dynamic Local Search (DLS) that could outperform the systematic search techniques. Interestingly, a common characteristic of DLS heuristics is that they all depend on the use of weights during searching for satisfiable formulas.

In our study we experimentally investigated the weights behaviors and movements during searching for satisfiability using DLS techniques, for simplicity, DDFW DLS heuristic is chosen. As a results of our studies we discovered that while solving hard SAT problems such as blocks world and graph coloring problems, weights stagnation occur in many areas within the search space. We conclude that weights stagnation occurrence is highly related to the level of the problem density, complexity and connectivity.

1. INTRODUCTION

The propositional satisfiability (SAT) problem is at the core of many computer science and artificial intelligence problems. Hence, finding efficient solutions for SAT has far reaching implications. In this study, we consider propositional formulae in conjunctive normal form (CNF): $\mathcal{F} = \bigwedge_m \bigvee_n l_{mn}$ in which each l_{mn} is a literal (propositional variable or its negation), and each disjunct $\bigvee_n l_{mn}$ is a clause. The problem is to find an assignment that satisfies $\mathcal{F}$. Given that SAT is NP complete, systematic search methods can only solve problems of limited size. On the other hand, relatively simple stochastic local search (SLS) methods have proved successful on a wide range of larger and more challenging problems [10]. Furthermore, stochastic local search (SLS) techniques are proven to be effective in solving hard satisfiability boolean problems.

However, their performance is still arguably poor when compared to systematic search techniques. Therefore, and since the development of the first clause weighting dynamic local search (DLS) algorithms for SAT, namely he Breakout heuristic [16], tries were made to enhance the local search techniques in many different ways. Some researchers turned their focus on studying the structure of the satisfiability problems (as in 1.1) to better understand the complexity of it and to come up with algorithms that could solve the problems in an optimal way. Other researchers focused on investigating new heuristics that alter the search space based on some information gathered during searching for a solution (as in 1.2). Thus, many heuristics, enhancements and developments were introduced to improve SLS techniques performance in the last three decades. As a result, a group of heuristics were introduced that could outperformed the systematic search techniques, namely Dynamic Local Search (DLS). Recent DLS algorithms depend on the use of weights to alter the search space. In other words, weights are used when there are no moves that could decrease the search cost, to make it possible for the technique to take unattractive moves which could increase the search cost temporarily.

1.1 Propositional satisfiability (SAT) complexity and hardness

It is proven that hard combinatorial SAT problems are the benchmarks that are used to test the efficiency, accuracy and optimality of a given algorithm [12]. As easy problems could be solved by any algorithm in a reasonable manner [21], which in turn does not reflect the real performance of a solving techniques. Therefore, studies since almost three decades focused on studying the hardness, complexity, and density of a countless number of SAT problems [6, 8, 15, 9]. Thus, a large distribution of hard problems was produced. These hard problems are categorized into two main divisions: 1) satisfiable and 2) un-satisfiable instances. Furthermore, in The International SAT solver competition (http"//www.satcompetition.org/) there are three sub divisions of the two main divisions: a) Industrial, b) Crafted, and c) Random instances. In another field of studies, researchers not only investigated whether a problem is satisfiable or not, they also studied how hard a problem is, as in [17, 21, 1].

1.2 Propositional satisfiability (SAT) dynamic solving techniques

Since the development of the Breakout heuristic [16], clause weighting dynamic local search (DLS) algorithms for SAT have been intensively investigated, and continually improved [5, 7]. However, the performance of these algorithms remained inferior to their non-weighting counterparts (e.g. [13]), until the more recent development of weight smoothing heuristics [24, 19, 11, 23]). 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 [24], SAPS [11], PAWS [23]), EWS [4], COVER [18] and recently CScore-SAT [3]) 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. Except for COVER which updates the edge weights in every step of the search.

However, a key issue with DLS algorithms is that their performance depend mainly on the efficiency of modifying the weights during the search, regardless of some other factors which may play a crucial role in their performance when applied for solving large and hard SAT problems such as Blocks World and Graph Coloring problems. For Instance, the size of backbones [22][1], the phase transition occurrence, and the density of a given problem.

Our study focuses on another factor and investigate its impact on the performance of DLS solving techniques. The question addressed in the current paper is that what happens to the weights when a clause and its neighboring clauses are satisfied?. For instance, if clause c_i is connected to *n* number of clauses (neighboring area of clause c_i, as discussed in sub-section 2.2) and by assuming that clause c_i became satisfied, by flipping one of its literals l_{im}, where all its neighboring clauses are satisfied too, should clause c_i and its neighbors keep the weights?.

In the remainder of the paper we generally discuss the clause weighting most known algorithms such as SAPS, PAWS and DLM and DDFW. Then we discuss elaborately on DDFW technique as it is used for the purpose of the current study. Then, we show empirically the weights behaviors and movements during the search via an intensive experimentation on a broad range of benchmark SAT problems. Then we analyze the results and show the general outcome of the experiments. Finally, we conclude our work and some guidelines for future work are given.

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 neighborhood. 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 [24], SAPS [11] and PAWS [23]) share a similar structure in the way that weights are updated:2 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 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

[1]back in 2001, Slaney et. al. [22] studied the impact of backbones in optimization and approximation problems. He concluded that in some optimization problems, backbones are correlated with the problem hardness. He also suggested that heuristic methods when used to identify backbones may reduce problem difficulty.

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.

2.1 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 normalizing 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 normalize weight. In this respect, DDFW is similar to the predecessors of SAPS (SDF [19] and ESG [20]), which both adjust and normalize the weight distribution in each local minimum. Because these methods adjust weight across all clauses, they are considerably less efficient than SAPS, which normalizes weight after visiting a series of local minima.3DDFW escapes the inefficiencies of SDF and ESG by only transferring weights between pairs of clauses, rather than normalizing 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 normalized without repeatedly adjusting weights on all clauses.

Algorithm 1 DDFW($\mathcal{F}$, W_{init})

```
1: randomly instantiate each literal in $\mathcal{F}$;
2: set the weight $w_i$ for each clause $c_i \in \mathcal{F}$ to $W_{init}$;
3: while solution is not found and not timeout do
4:    find and return a list $\mathcal{L}$ of literals causing the greatest reduction in weighted cost $\Delta w$ when
      flipped;
5:    if ($\Delta w < 0$) or ($\Delta w = 0$ and probability $\leq 15\%$) then
6:       randomly flip a literal in $\mathcal{L}$;
7:    else
8:       for each false clause $c_f$ do
9:          select a satisfied same sign neighbouring clause $c_k$ with maximum weight $w_k$;
10:         if $w_k < Init$ then
11:            randomly select a clause $c_k$ with weight $w_k \geq W_{init}$;
12:         end if
13:         if $w_k > W_{init}$ then
14:            transfer a weight of two from $c_k$ to $c_f$;
15:         else
16:            transfer a weight of one from $c_k$ to $c_f$;
17:         end if
18:      end for
19:   end if
20: end while
```

[2]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.

[3]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.

DDFW's weight transfer approach also bears similarities to the operations research sub-gradient optimization techniques discussed in [20]. 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 sub-gradient optimization in that weight is only transferred between pairs of clauses and not across the board, meaning less computation is required.

2.2 Exploiting Neighborhood Structure

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

We term clause c_i to be a neighbor 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 $l_{im} = l_{jn}$ as in Fig 1. Furthermore, we term c_i to be a same sign neighbor 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 neighboring 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 neighboring clauses of 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 neighborhood 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 neighboring 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 neighbor 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 neighbors. 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 neighbors 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. EXPERIMENTAL EVALUATION AND ANALYSIS

Problem	CNF			clausSize		# min size	# max size
	Atoms	Clauses	literals	min	max		
bw-large.d	6325	131973	294118	2	20	102580	25
logistics.c	1141	10719	27978	2	14	5846	46
ais10	181	3151	7255	2	10	2322	10
g125.17	2125	66272	134419	2	17	66147	125
g125.18	2250	70163	142326	2	18	70038	125
bmc-ibm-5	9396	41207	103560	1	53	73	1
bmc-ibm-10	59056	323700	853193	1	32	447	1
bcsp5080-h	750	26510	53670	2	15	26460	50
bcsp5080-m	750	26510	53670	2	15	26460	50
bcsp5040-h	750	26456	53562	2	15	26406	50
bcsp5040-m	750	26510	53670	2	15	26460	50
bw-large.b	1087	13772	31767	2	12	9735	11
f400-h	400	1720	5160	3	3	1720	1720
bcsp3040-h	300	5598	11436	2	10	5568	30
uf250-h	250	1065	3195	3	3	1065	1065
bmc-ibm-1	9685	55855	149765	1	39	49	1
uf400-h	400	1700	5100	3	3	1700	1700
uf100-m	100	430	1290	3	3	430	430
rocket	433	2902	5839	1	5	42	14
par32-3	3176	10297	2758	1	3	141	7128
par16-m	334	1332	3874	2	3	122	1210
hanoi6.shuff	4968	39666	98346	1	9	1020	126
flat200-h	600	2237	4674	2	3	2037	200
flat100-h	300	1117	2334	2	3	1017	100
flat100-m	300	1117	2334	2	3	1017	100
f1600-h	1600	6880	20640	3	3	6880	6880
f1600-m	1600	6873	20622	2	3	4	6869
f800-h	800	3440	10320	3	3	3440	3440
f800-m	800	3440	10320	3	3	3440	3440
g250.15	3750	233965	471180	2	15	233715	250
g250.39	7250	454622	915994	2	29	454372	250
huge	459	7054	15567	2	10	5682	5
bw_large.c	3016	50457	114314	2	16	37493	17

Table 1. structural information about some of the the original DIMACS 2005 problem sets. the problem were carefully selected taking to consideration their size, density, and connectivity.

As we stressed in section 2.1 and section 2.2, DDFW can exploit the neighboring structure of a given clause c_i and identify the weight alliances of c_i. These weight alliances act as the source of weights donors when a clause within the alliance need more weights. Also, they stabilize the process of weight transfer as they can lead to keeping weights within each allies as long as no weight transfer is needed, thus all the clauses within the neighborhood are satisfied. However, this has further implications as it could lead the search to get into one of the following scenarios that we discovered while investigating the weight transfer process: i) weights within a neighborhood (Allie) could not be transferred to another sub area of the search space. ii) weights of a specific neighborhood may keep circulating within their neighborhood. iii) if the level of connectivity between any two neighboring allies is low, weight transfer may suffer from stagnation, hence it can make the search process longer than it suppose to be.

In order to show the above three scenarios and their impact on the search process of any given DLM technique, we first studied the general structure of some benchmark problems. Table 1 illustrates the structure of the benchmark problems that were carefully selected based on their size, complexity and hardness. We attempted to reproduce a 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 k3 problems from the SAT2004 competition benchmark. Secondly, we obtained the 10 SATLIB quasi-group existence problems used in [2]. These problems are relevant because they exhibit a balance between randomness and structure. Finally, we obtained the structured problem set used to originally evaluate SAPS [11]. These problems have been widely used to evaluate clause weighting algorithms (e.g. in [23]) 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.

For each selected problem we firstly show the number of atoms (variables) of the problem, the number of the clauses of the problem and the number of literals. Secondly, we show the minimum and the maximum number of literals that form a clause within the problem structure. Finally, we show the number of minimum sized clauses as will as the number of maximum sized clauses. We designed our experiment to be as follows:

- for each problem we ran DDFW 1000 run. Each run time out was set to 10,000,000 flips.
- in each run, we recorded the change of weights in every 10,000 flips.
- in every local minimum, we recorded whether DDFW heuristic selected a neighboring satisfied clause from the weight allie of the false clause, or it picked a satisfied clause, to be weight donor, randomly from outside the neighboring area of the false clause.
- plots were made for each problem to illustrates the change of weights of the false clauses from the starting point of the search process until the solution is found or it reaches a time out. This is done for all the figures included in the paper.

Table 2 show the detailed results of the runs. For each problem we firstly show the success rate (which reflect the percentage of whether a solution is found or not). Then we show the total number of local minima that DDFW heuristics found before reaching a global optima. Then we show the number of times the DDFW heuristic randomly picked a clause as a weight donor. Next we show the number of times that DDFW heuristic picked a neighboring satisfied clause as a weight donor from the false clause allies. Finally we show the the average number of times DDFW heuristic deterministically picked a clause as a weight donor.

In order to show the weights transfer and movements during the search we plotted the false clauses and their weights changes and the number of neighboring clauses of each false clause. This to show the relationship between the change of weights and the number of neighboring clauses of a false clause. Out of all problem sets we discuss four problems as they are of great importance to this work, namely, the Uniform Random 3SAT, the Parity problem, the Blocks World and the Graph Coloring problem because they explicitly show the previously mentioned three scenarios. These four problem sets are of different sizes and level of hardness. The Uniform Random 3SAT (uf100 and uf250) is the smallest set that has 100 variables and 430 clauses, where the Uniform Random 3SAT 250 has 250 variables and 1065 clauses. Fig 2 show the results for both problems. We can see that both problems were easy to solve. Also, the weight movement was smooth as their was enough neighboring clauses to donate weights to a false clause and more importantly when a sub area become satisfied, the connectivity between clauses allow the transfer of weights easily (no weight stagnations occur). This is also true with the second problem, The Parity problem even that the level of hardness of the Parity 16 and the Parity 32 is higher than the Uniform Random 3SAT, as in Fig 3. What was experimentally interesting is the Graph Coloring problem (both the g125.17 and the g125.18 problems as in Fig 4, where g125.17 has 2125 variables 66272 clauses and the g125.18 has 2250 variables and 70163 clauses) where firstly, DDFW could not reach 100% success rate on the g125.17. Secondly, the figure show a clear gap between the movement of the weights. Which means the occurrence of weights stagnation. Thus, the connectivity between the clauses is either very low which make transferring weights among the clauses is limited to the clauses that are directly connected, or the connectivity of the clauses is very high which means a larger number of neighbors that could keep the weights for longer time and prevent other false clauses some where else in the search space from using them. Finally the Blocks World, Fig 5, which has 6325 variables and 131973 clauses., which has a similar weight transfer behaviors as the parity problem and the uniform random 3SAT problem with the exception of the level of hardness as the block world problem was harder to solve and the weights were moving more often during the search space.

problem	sRate	nLocalMinima	nRandomDist	nDeterminDist	avgDetr
bw-large.d	100	31441.5	4634.3	410747.3	86.9
logistics.c	100	232398.7	5276.9	445695.7	84.2
ais10	100	96772.7	2791.7	237724.4	70.4
g125.17	47	23438576	10545491.5	333747500.3	35.9
g125.18	100	81307.3	9347.9	508783.1	27.7
bmc-ibm-5	32	803981.7	812007.5	13719359	16.3
bmc-ibm-10	30	70030.7	439221	14571277.9	32.7
bcsp5080-h	90	12966448.1	3444963.8	107933199	39.4
bcsp5080-m	90	14887277.3	3920599.3	123967018.9	38.7
bcsp5040-h	23	20892491.8	5119784.4	241167931.3	46.6
bcsp5040-m	25	21307522.8	5296308.5	232836213.3	43.5
bw-large.b	100	4409.5	454.8	31528.1	67.0
f400-h	100	329514.4	16071.4	1583429	97.9
bcsp3040-h	100	3565452.6	478237.2	20765453.3	44.1
uf250-h	100	248885.2	7706.3	757378.6	97.6
bmc-ibm-1	70	1100209.7	439064.5	17506230.3	39.5
uf400-h	100	99885.8	5420.5	527031.4	97.3
uf100-m	100	664.6	18.5	1570.9	85.5
rocket	68	2027948.8	2302130.7	76669187.7	33
par32-3	89	793067.8	299015.4	11539640.6	38.1
par16-m	100	805116.6	72261.5	7064764.5	97.3
hanoi6.shuff	54	1067162.8	166107.2	15080491.9	90.2
flat200-h	100	756983.1	1635948.9	4709022	2.1
flat100-h	100	6779.6	7037.9	23926.1	2.4
flat100-m	100	38778	45406.8	152577.6	3.2
f1600-h	100	811342.5	126567.6	12453381.6	97.9
f1600-m	100	139874.9	19431.5	1900084.9	97.9
f800-h	100	447944.9	38308.9	3748470.3	97.4
f800-m	100	191996.7	18452	1820062.8	98.2
g250.15	100	20.5	734.6	186.4	0
g250.29	0	13676053.1	6477528.2	306931973	47.6
huge	100	986.6	118.6	6070.7	55.3
bw_large.c	100	53580.9	5654.3	490703	83.8

Table 2. The table show the number of successful tries made by DDFW, the number of local minima faced the search, the number of random weights distribution during the search, the number of deterministic weight distribution and the average number of deterministc weight distribution. Weights behaviors and movements during the search.

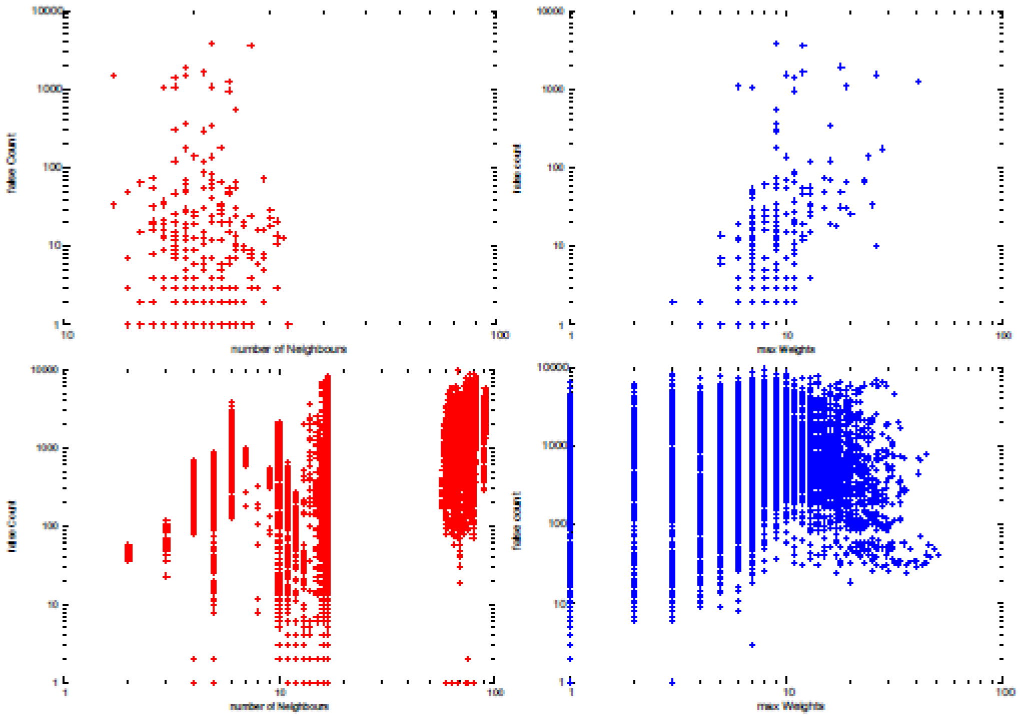

Fig. 2. false clauses and their weights during the search, the uf100 problem top and the uf250 bottom

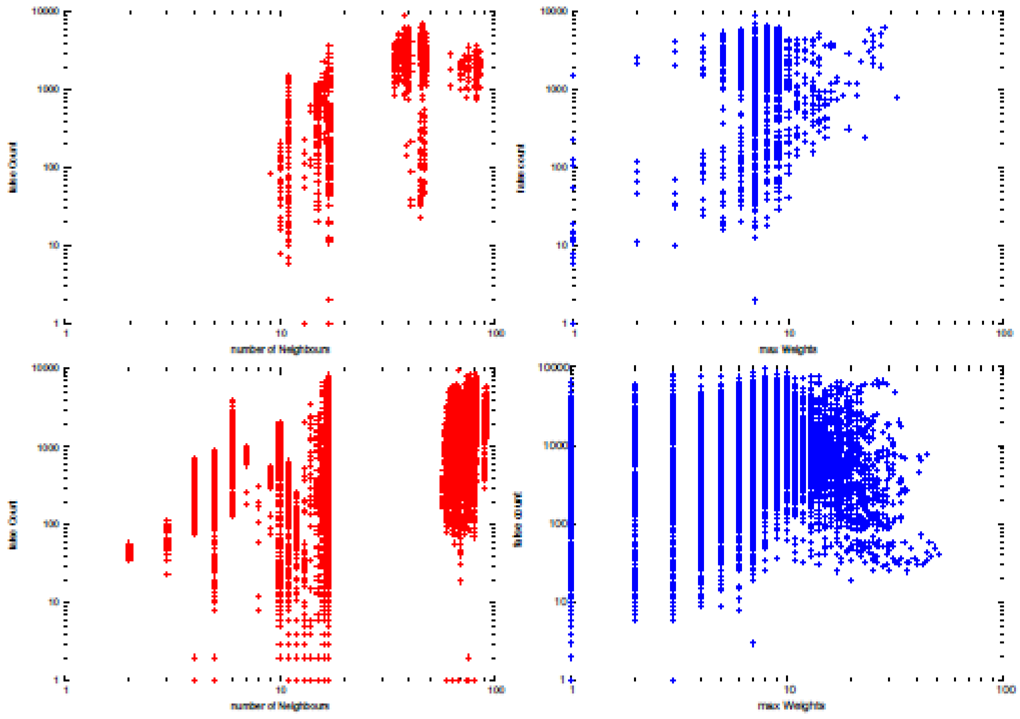

Fig. 3. false clauses and their weights during the search, the par16 top and par32 bottom

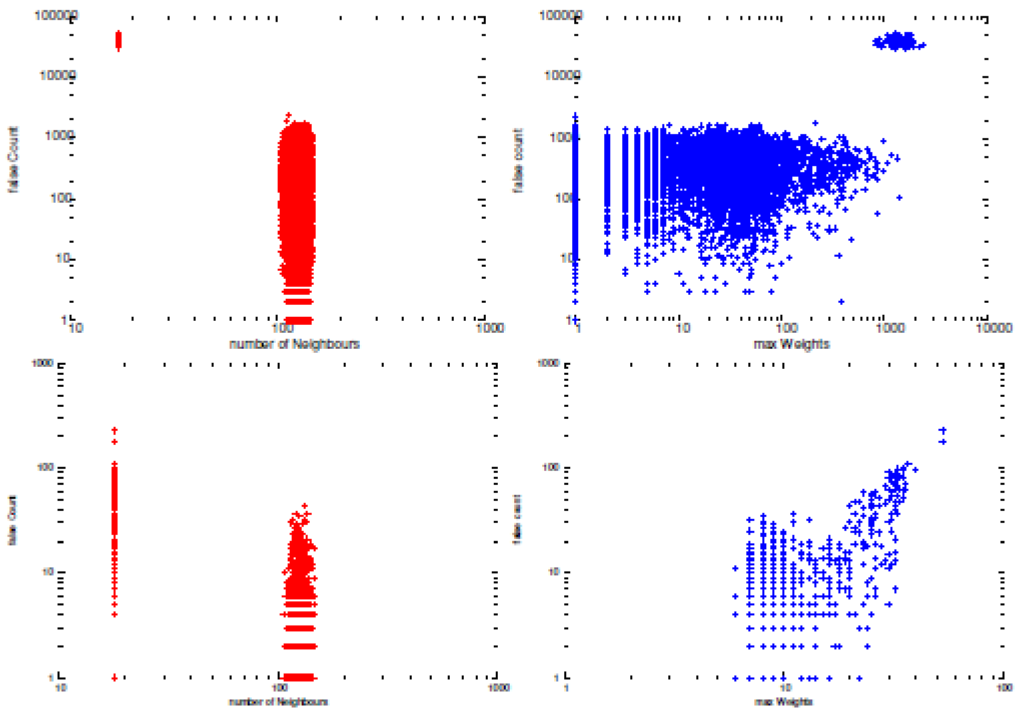

Fig. 4. false clauses and their weights during the search, the g125-17 top and the g125-18 bottom

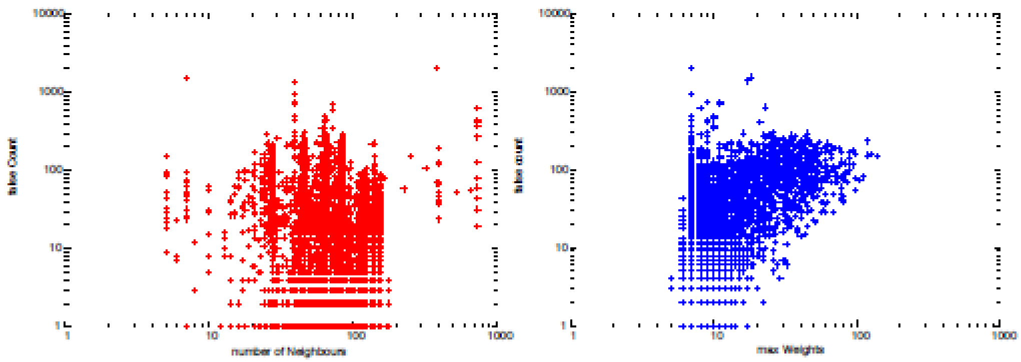

Fig. 5. false clauses and their weights during the search, the bw-d.large

4. CONCLUSION

As a conclusion, DLS weighting techniques performance could suffer from weight stagnation that leads to the slowness of a chosen techniques. Our experiments show that these weight stagnations are not general to all problems but rather they are a problem specific characteristic. We have looked into each problem characteristic such as its hardness, complexity, and density. As a result, each problem characteristics may contribute to the occurrence of weight stagnations in some stages during the search. The DDFW algorithm is a relatively simple application of neighborhood weighting, and further experiments (not reported here) indicate more complex heuristics can be more effective on individual problems. In particular, we have looked at adjusting the amount of weight that is redistributed and allowing DDFW to randomly pick donor clauses according to a noise parameter. However, we have yet to discover a general neighborhood heuristic as effective as DDFW over the range of problems considered. In future work we consider it will be promising to extend a DDFW-like approach to handle weight stagnations via adjusting weights in stagnated alliances regardless of whether they are satisfied or not. This could be done by improving the exploitation of neighboring areas.

ACKNOWLEDGEMENT

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.

REFERENCES

[1] D. Achlioptas, C. Gomes, H. A. Kautz, and B. Selman, \Generating satisfiable instances," in Proceedings of 17th AAAI, 2000, pp. 256{261.

[2] Anbulagan, D. Pham, J. Slaney, and A. Sattar, \Old resolution meets modern SLS," in Proceedings of 20th AAAI, 2005, pp. 354{359.

[3] S. Cai, C. Luo, and K. Su, \Scoring functions based on second level score for k-sat with long clauses," J. Artif. Intell. Res. (JAIR), vol. 51, pp. 413{441, 2014.

[4] 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.

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

[6] S. A. Cook, \The complexity of theorem-proving procedures," in Proceedings of the Third Annual ACM Symposium on Theory of Computing, ser. STOC'71. New York, NY, USA: ACM, 1971, pp. 151{158. [Online]. Available: http://doi.acm.org/10.1145/800157.805047

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

[8] M. R. Garey and D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co., 1990.

[9] I. P. Gent and T. Walsh, \The hardest random SAT problems," in KI-94: Advances in Artificial Intelligence, 18th Annual German Conference on Artificial Intelligence, Saarbrucken, Germany, September 18-23, 1994, Proceedings, 1994, pp. 355{366.

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

[11] F. Hutter, D. Tompkins, and H. Hoos, \Scaling and Probabilistic Smoothing: Efficient dynamic local search for SAT," in Proceedings of 8th CP, 2002, pp. 233{248.

[12] M. Jarvisalo, D. Le Berre, O. Roussel, and L. Simon, \The international SAT solver competitions," AI Magazine, vol. 33, no. 1, pp. 89{92, 2012.

[13] D. McAllester, B. Selman, and H. Kautz, \Evidence for invariants in local search," in Proceedings of 14th AAAI, 1997, pp. 321{326.

[14] P. Mills and E. Tsang, \Guided local search applied to the satisfiability (SAT) problem," in Proceedings of 15th ASOR, 1999, pp. 872{883.

[15] D. Mitchell, B. Selman, and H. Levesque, \Hard and easy distributions of sat problems," in ghiuyguyigy, 1992, pp. 459{465.

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

[17] P. Prosser, \Binary constraint satisfaction problems: Some are harder than others," in ECAI. PITMAN, 1994, pp. 95{95.

[18] 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, Osnabruck, Germany, September 10-13, 2007, Proceedings, 2007, pp. 412{426.

[19] D. Schuurmans and F. Southey, \Local search characteristics of incomplete SAT procedures," in Proceedings of 10th AAAI, 2000, pp. 297{302.

[20] D. Schuurmans, F. Southey, and R. Holte, \The exponentiated subgradient algorithm for heuristic boolean programming," in Proceedings of 17th IJCAI, 2001, pp. 334{341.

[21] B. Selman, D. Mitchell, and H. Levesque, \Generating hard satisfiability problems," Artificial Intelligence, vol. 81, pp. 17{29, 1996.

[22] J. Slaney and T. Walsh, \Backbones in optimization and approximation," in Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume1, ser. IJCAI'01, 2001, pp. 254{259.

[23] 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.

[24] 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