Backbone Guided Dynamic Local Search for Propositional Satisfiability

Valnir Ferreira Jr.

Institute for Integrated and Intelligent Systems
Griffith University, PMB50, Gold Coast Mail Centre, QLD 9726, Australia
v.ferreira@griffith.edu.au

Abstract. This comparative study examines the impact of backbone guided heuristics on the performance of dynamic local search methods. We study alternatives to the backbone membership estimation problem, discuss how our proposed estimation phase addresses it, and discuss how this information is integrated in the host methods. Backbone guidance results in significantly faster dynamic local search on the large problems tested, but it is of questionable use for small problem domains where the cost of backbone estimation alone represents a significant part of the total search cost.

1 Introduction

The propositional satisfiability (SAT) problem is of significant interest to the artificial intelligence community because many interesting AI application domains can be formulated in this way. A SAT problem consists of finding an assignment for the Boolean variables in a propositional formula that makes the formula true.

Stochastic local search methods [1] are amongst the most successful approaches to solving SAT problems. Since the introduction of GSAT [2], the development of efficient local search methods for SAT has been an active area of research. However, GSAT's major drawback was its inability to escape from local minima, and the introduction of clause weighting methods started to address this problem. In its purest form [3], clause weighting mechanisms work by assigning weights to every clause in the SAT problem and by incrementing the weights of false clauses every time the search reaches a local minimum. Different implementations vary in the way clause weights are incremented. Some methods use integer arithmetic, whereas others use floating point arithmetic, and hence are commonly termed additive and multiplicative clause weighting local search methods, respectively [4].

More importantly, these methods need to effect clause weight decrements (or rescaling) in order to keep the clause weight profile relevant to the current search context. This problem can be seen as trying to maintain an optimal balance between long- and short-term clause weight memory [5]. It is now widely accepted that the degree of a method's ability to handle this problem is positively correlated with its ability to efficiently solve hard SAT problems. The body of work pertaining to clause weighting local search is termed dynamic local search (DLS), and its methods are amongst the best for solving hard SAT from a range of domains [1, 6–9]. The DLS method of particular relevance to this study is the pure additive weighting scheme (PAWS) [4], as well as three of its most successful variants [10–12]. Figure 1 shows PAWS's pseudo-code.

PAWS has two parameters: P_{flat} (line 13) and WDP (line 17). The former controls the probability of taking an equal-cost flip that leaves the weighted cost of the solution unchanged, while the latter determines the number of weight increases (line 16) allowed before a weight decrease takes place (lines 17-18). In practice only WDP has its value set on a problem-per-problem basis, whereas a P_{flat} setting of 0.15 works well for most domains [4].

The backbone of a SAT problem consists of those variables for which logical values cannot vary from solution to solution [13]. Hence, if the backbone is known, its variables can have their values fixed, leading to a reduction in the search space. Unfortunately, computing the backbone involves solution enumeration, making it impractical for all but the smallest problem instances. An alternative is to estimate backbone membership by sampling variable assignments in near-optimal solutions. Recently, positive results and

```
PAWS procedure
begin
    generate random starting point
    for each clause $c_i$ do set clause weight $w_i \leftarrow 1$
    while solution not found and not terminated do
        $best \leftarrow \infty$
        for each literal $x_{ij}$ in each false clause $f_i$ do
            $\Delta w \leftarrow$ change $\sum w$ in $f_i$ caused by flipping $x_{ij}$
            if $\Delta w < best$ then $L \leftarrow x_{ij}$ and $best \leftarrow \Delta w$
            else if $\Delta w = best$ then $L = L \cup x_{ij}$
        end for
        if $best < 0$ then
            randomly flip $x_{ij} \in L$
        else if $best = 0$ and probability $\leq P_{flat}$ then
            randomly flip $x_{ij} \in L$
        else
            for each false clause $f_i$ do $w_i \leftarrow w_i + 1$
            if # times clause weights increased % $WDP = 0$ then
                for each clause $c_i \mid w_i > 1$ do $w_i \leftarrow w_i - 1$
            end if
        end if
    end while
end
```

Fig. 1. PAWS.

further insights have been obtained from using local search with backbone estimation heuristics for the maximum satisfiability (MAX-SAT)[1] and travelling salesperson problem (TSP) domains [14, 15].

To our knowledge, no studies have yet investigated backbone guidance in the context of DLS for SAT, despite recent positive results indicating the usefulness of backbone guided search [15–18], and despite the superior performance of DLS methods on large and hard SAT problems. Our work represents a contribution to addressing this void. In the subsequent sections, we examine related work, discuss the implementation details pertaining to our approach, analyse and discuss the results from our empirical evaluation, and present our conclusive remarks.

2 Related Work

Zhang *et al.* [14] introduced a backbone guided implementation of Walksat [19]. BGWalksat splits K tries into two phases: (a) an estimation phase of length K_1 tries (each starting from a distinct randomly generated initial assignment) used to collect local minima and to estimate the backbone, and (b) an actual search phase of length $K - K_1$ tries that uses the collected backbone information to guide the otherwise random Walksat choices.

Walksat makes five random choices. The first is the randomly generated initial assignment of values to literals. BGWalksat considers replacing this with a heuristic backbone sampling (HBS): the higher a literal's pseudo-backbone frequency, the higher its probability of being selected into the initial assignment. Our approach uses HBS.

Walksat's second random choice is the selection of a false clause. BGWalksat considers collecting and using pseudo-backbone clause frequencies (PBCF) to replace this random decision. PBCF measure the frequencies that clauses are *satisfied* in all local minima. If this feature is used, then a false clause with higher frequency is selected with a higher probability. We note that because typical local minima have a small number of false clauses and a relatively much greater number of true clauses, it would be more efficient for PBCF to measure the frequencies that clauses are *unsatisfied* and then reverse the decision process. Our approach does not use PBCF, and so we do not discuss this feature further.

[1] When not all clause are satisfiable, SAT becomes MAX-SAT, an optimisation problem where the goal is to maximise the number of satisfied clauses.

The final three possible random decisions relate to the selection of a variable from the false clause previously picked. Walksat can perform a *freebie*, a *greedy* or a *noise* pick, as seen in Figure 2.

```
Walksat procedure
generate random initial assignment
repeat
    $c \leftarrow$ an unsatisfied clause chosen at random
    if there exists a variable $x$ in $c$ with break value = 0
        flip the value of $x$   //freebie pick
    else
        with probability p
            $x \leftarrow$ a variable in $c$ chosen at random
            flip the value of $x$   //noise pick
        with probability (1-p)
            $x \leftarrow$ a variable in $c$ with smallest break value
            flip the value of $x$   //greedy pick
until a satisfying assignment is found
```

Fig. 2. The main loop of the Walksat procedure [20].

Rather than randomly picking a variable, BGWalksat introduces a bias based on the variables' pseudo-backbone frequency estimation (PBFE). It works as follows: consider Walksat selected a false clause C with literals a, $\neg b$, and c with PBFEs 0.2, 0.3, and 0.5, respectively. The backbone bias is introduced by selecting a, $\neg b$, and c with probabilities $p(a) = 0.8/(0.8 + 0.7 + 0.5)$, $p(\neg b) = 0.7/2.0$, and $p(c) = 0.5/2.0$. This is because literal $\neg a$ has a higher frequency (0.8) than either b (0.7) or $\neg c$ (0.5). In other words, it estimates that $\neg a$ appears more often in the backbone than either b or $\neg c$, and so it biases the decision towards flipping a.

Run-time efficiency depends on obtaining accurate PBFEs, and on deciding when to use the backbone bias instead of Walksat's random choices. Whilst both of these factors are of relevance to BGWalksat, a discussion of the former is of greater relevance to our work, whereas the latter is only relevant to BGWalksat, and space limitations preclude us from discussing it further. Obtaining accurate PBFEs is also crucial to the performance of other backbone guided methods [15, 16].

BGWalksat's authors considered two PBFE schemes, namely average counting (AC), and cost reciprocal average counting (CRAC). AC considers samples from all local minima S, and takes the frequency of a variable-value pair $l = (x_i, v_i)$ that appears in S as its pseudo-backbone frequency, i.e., $pbfe(l) = \sum_{\forall s_i \in S, l \in s_i} 1/\,|S|$. All local minima sampled are considered to be of the same quality. CRAC also samples from all local minima S encountered, but discounts the contribution of a local minima s_i based on its cost c_i, i.e., the pseudo-backbone frequency of a variable-value pair $l = (x_i, v_i)$ is computed as $pbfe(l) = (\sum_{\forall s_i \in S, l \in s_i} 1/c_i)/(\sum_{\forall s_i \in S} 1/c_i)$. Our approach uses CRAC.

For BGWalksat, the best PBFE scheme is relative to the target problem domain. PBFE efficiency also relies on: (a) the choice of local minima to sample and how to account for minima of different costs, and (b) the number of sampling tries. We explore these two issues in our implementations of backbone guided dynamic local search discussed next.

3 Backbone Guided Dynamic Local Search

Our approach runs in two phases. The estimation phase runs the host DLS method to collect PBFE data. The execution phase runs the host method using the collected PBFEs for breaking ties amongst equally ranked candidate flips.

For the estimation phase, we empirically tested a number of PBFE schemes, including BGWalksat's AC and CRAC (the best performing scheme in our experiments), as well as a CRAC variant that considers a local minimum's contribution to the PBFE to be directly proportional to its cost. This variant was only marginally worse than CRAC. We have therefore elected to use CRAC, i.e., the pseudo-backbone frequency of a variable-value pair $l = (x_i, v_i)$ is computed as $pbfe(l) = (\sum_{\forall s_i \in S, l \in s_i} 1/c_i)/(\sum_{\forall s_i \in S} 1/c_i)$. The cost c_i is the number of false clauses in s_i. We considered 1, 10 and 100 learning tries (lt), each starting from a unique randomly generated initial assignment. The purpose of multiple learning tries is to reduce the bias in the estimation procedure. Learning tries never find a solution due to the introduction of a learning depth threshold (ldt): the try is halted when the number of false clauses reaches the ldt. We have experimented with settings of 4, 8, 16, and 32, and set a learning try's cutoff to 1 million flips.

For the execution phase, we applied our approach to PAWS's most recent and successful extensions: resPAWS [10], and mvPAWS [11], calling the resulting methods BGresPAWS, and BGmvPAWS, respectively. Backbone guidance was achieved using insights gained from our recent positive results on the usefulness of tie breaking in additive DLS methods [12], and so BGresPAWS and BGmvPAWS use PBFE information to break ties amongst equally ranked candidate flips. Both BGresPAWS and BGmvPAWS use PAWS's standard flip selection strategy (see Figure 1). If more than one candidate flip is available, i.e., $|L| > 1$, the tie is broken using BGWalksat's formula (see Section 2), with the members of L in PAWS corresponding to the literals in the false clause selected by BGWalksat. We do not update PBFEs during execution time. All of our experiments were performed on a Sun cluster with 8 Sun Fire V880 servers, each with 8 UltraSPARC-III 900 MHz CPU and 8 GB memory per node. The cutoff for the execution phase was set to 50 million flips (250 million for the bqwh problems). As a key motivation for developing incomplete local search methods is to solve problems beyond the reach of complete solvers, we include results for three prominent complete solvers: Satz215.2[2], zChaff 2004.11.15[3], and MiniSat-C_v1.14[4]. Cut-off time for complete solvers in all experiments was set to 4 hours.

The resPAWS method [10] adds Satz's resolution procedure (adding resolvent clauses of length ≤ 3) [21] as a preprocessing step to PAWS, Walksat [19], AdaptNovelty+ [22], and RSAPS [9]. However, it is well documented [23] that neither Walksat nor RSAPS can match PAWS's performance on large instances, and AdaptNovelty+ is not DLS, and so we chose not to include these methods in our empirical study. The most notable improvements observed for resPAWS were on a set of ten quasigroup existence (qge) problems [24]. For these problems (qg1-07, qg1-08, qg2-07, qg2-08, qg3-08, qg4-09, qg5-11, qg6-09, qg7-09, qg7-13), the addition of resolvent clauses allowed for the truth values of up to half of the variables to be determined, and for up to approximately 90% of the original clauses to be deleted from the formula, resulting in significant run-time performance gains for resPAWS even when considering the additional preprocessing time. Results for the performance of complete solvers on these qge problems were not given in [10].

However, the level of gain obtained by resPAWS on the qge problems did not map to real-world domains such as parity learning, blocks world and logistics (available from www.satlib.org), possibly due to the fact that resolution fails to achieve the same reductions in instance size observed for the qge instances [10]. Additionally, the results from our experiments, shown in Table 1, indicate that all three complete solvers find solutions in less than 4 seconds on average with 100% success while resPAWS takes more than 50 seconds on average and performs sub-optimally *w.r.t.* success ratio, highlighting the absolute superiority of complete solvers in the qge problem domain.

	Time (secs)		Success	Time (secs)		Success	Time (secs)		Success		Time (secs)		Success
	mean	median	%	mean	median	%	mean	median	%		mean	median	%
	lt 1			lt 10			lt 100						
ldt 4	44.44	0.23	96.40	106.04	1.36	93.33	210.18	9.03	80.00	resPAWS	51.92	0.10	94.8
ldt 8	44.26	0.16	96.80	70.86	0.78	96.80	143.18	6.12	92.00	Satz	3.29	0.23	100.0
ldt 16	36.01	0.15	97.60	60.73	0.42	97.60	87.72	1.21	91.00	zChaff	3.64	0.04	100.0
ldt 32	41.16	0.13	97.20	67.22	0.28	96.40	98.63	0.80	91.00	MiniSat	3.78	0.06	100.0

Table 1. Run-time data obtained from 250 runs (25 per problem) showing the effect of different lt and ldt settings on BGresPAWS's performance on ten quasigroup existence problems. The results for BGresPAWS include estimation phase times. PAWS's clause weight decrement parameter was set to 4. These results also illustrate the superior performance of complete solvers in the qge problem domain.

[2] http://www.laria.u-picardie.fr/ cli/englishpage.html
[3] http://www.princeton.edu/ chaff/zchaff.html
[4] http://www.cs.chalmers.se/cs/research/formalmethods/minisat/minisat.html

We compared the performance of resPAWS and BGresPAWS on these problems and found that BGresPAWS's best performance is obtained when using one learning try (*lt* 1) and a *ldt* setting of 16. With these settings, BGresPAWS has better mean run-time performance, but compares negatively against resPAWS when we consider the median run-times. The explanation for this relates to the disparate problem hardness in this set: eight out of ten problems are solved by resPAWS with 100% success under five seconds, with the remaining two problems, qg2-08, and qg7-13, being significantly more challenging. Put simply, the additional computational time resulting from BGresPAWS's estimation phase is not justifiable for trivial problems that can be solved within fractions of a second, as is the case with most problems in this set. Conversely, backbone guidance results in significant performance gains as problems become harder. To illustrate this point, a comparison between resPAWS and BGresPAWS on the two hardest problems is shown in Table 2, indicating the superiority of the backbone guided method both in terms of run-time and success probability.

	Time (secs)		Success	Time (secs)		Success
	mean	median	%	mean	median	%
	qg2-08			qg7-13		
resPAWS	188.54	29.80	68.00	323.44	218.47	80.00
BGresPAWS	135.57	25.02	84.00	213.98	124.60	92.00

Table 2. Run-time data for the hard qge problems obtained from 25 runs. The results for BGresPAWS (*lt* 1 and *ldt* 16) include estimation phase times.

Although both resolution and PBFE are preprocessing techniques, backbone guidance provides the advantage of being more generally applicable. For some problems, resolution can actually increase problem size without bringing any simplification [10]. In other cases, resolution results in no changes to the formula under consideration, as was the case with the balanced quasigroup with holes problems used in our next experiment. In summary, we see these first results as promising given that BGresPAWS outperforms resPAWS on the harder problems, although we also highlight the need for carefully considering the trade off between performance gains and additional run-time resulting from the backbone estimation phase.

Our second experiment applied backbone guidance to another recent PAWS variant. The mvPAWS method uses a feature extraction mechanism that recovers and exploits variable and constraint structure hidden in the SAT encoded formulation of certain problems. In doing so, the method allows for significant improvements in solution times on a series of well-known benchmarks [11]. The balanced quasigroup with holes (bqwh) problems are amongst the best benchmarks for testing incomplete and complete solvers. We generated three sets of orders 30, 33, and 36 from the suggested ($number\, of\, holes/N^{1.55} = 1.7$) hardness region, and encoded these into SAT using the 3-D encoding method [25][5].

		Time (secs)	Success %	Time (secs)	Success %	Time (secs)	Success %		Time (secs)	Success %
		lt 1		lt 10		lt 100				
order 30	ldt 4	2.70	98.4	10.09	100.0	93.12	100.0	mvPAWS	4.05	100.0
	ldt 8	2.16	99.2	4.49	100.0	29.44	100.0	Satz	41.87	100.0
	ldt 16	3.00	100.0	2.89	100.0	9.81	100.0	zChaff	36.21	100.0
	ldt 32	4.34	100.0	2.52	100.0	5.17	100.0	MiniSat	86.70	100.0
		lt 1		lt 10		lt 100				
order 33	ldt 4	14.51	98.4	48.95	99.2	424.00	100.0	mvPAWS	35.69	100.0
	ldt 8	10.23	98.8	17.75	100.0	112.14	99.6	Satz	79.85	100.0
	ldt 16	17.13	99.6	12.18	100.0	27.97	99.6	zChaff	8,423.06	50.0
	ldt 32	44.55	98.4	17.61	100.0	17.76	99.6	MiniSat	238.07	100.0
		lt 1		lt 10		lt 100				
order 36	ldt 4	484.10	58.8	168.75	82.8	995.42	98.8	mvPAWS	90.88	98.8
	ldt 8	276.01	68.8	58.08	95.6	234.72	98.8	Satz	1,708.87	100.0
	ldt 16	357.95	70.4	49.66	95.6	78.83	96.0	zChaff	14,400.00	10.0
	ldt 32	388.80	69.6	57.07	92.8	58.71	97.6	MiniSat	10,975.60	50.0

Table 3. Median run-time and completion success data for the balanced quasigroup with holes problems.

[5] The author would like to thank Duc Nghia Pham for generating these instances.

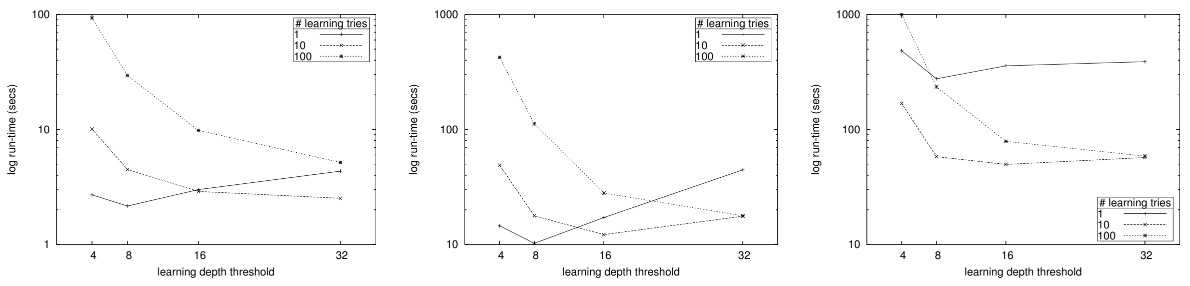

Fig. 3. mvPAWS vs. BGmvPAWS on the balanced quasigroup with holes order 30 (left), 33 (centre), and 36. Each data point is the median run-time from 250 runs (25 per problem). BGmvPAWS results include estimation phase times. PAWS's clause weight decrement parameter was set to 4, 4, and 3, for orders 30, 33, and 36, respectively.

We ran mvPAWS and BGmvPAWS on these bqwh problems, again investigating a series of *lt* and *ldt* settings. Our results, shown in Figure 3 and Table 3, indicate that BGmvPAWS benefits from the backbone guidance on these large instances, particularly when using 10 learning tries and a *ldt* setting of 16. With these settings, BGmvPAWS improves on mvPAWS's run-time by approximately 29%, 66% and 45% for the orders 30, 33, and 36, respectively.

Intuitively, by sampling from local minima with a lower number of false clauses (i.e., 4 or 8), the PBFE scheme can more accurately approximate the actual backbone. However, lower *ldt* settings result in longer estimation phase run-times, and our results reveal that the setting of 10 learning tries and a maximum depth of 16 is sufficient to obtain useful PBFEs while still maintaining BGmvPAWS's superior run-time performance. For the complete solvers, Satz proves to be the best alternative in this domain, with the performance of MiniSat and zChaff deteriorating significantly with problem size. Moreover, these results demonstrate the superior performance of DLS methods over complete methods in the bqwh domain.

4 Discussion and Conclusions

Our work has investigated the usefulness of backbone guidance to the performance of DLS methods for SAT. We have studied and implemented efficient ways to obtain and integrate pseudo-backbone frequency estimations to bias the decision process that is crucial to the performance of DLS methods: flip selection.

The problem of obtaining accurate pseudo-backbone frequency estimations has been highlighted, and we demonstrated how our estimation phase successfully addresses it. We have shown, for the first time, empirical evidence indicating the positive impact of backbone guidance on the run-time of state-of-the-art DLS. However, we have also identified the need for carefully considering the trade off between performance gains and additional run-time resulting from the estimation phase. Specifically, our results show that for small problems, the additional overheads brought about by the backbone estimation phase are likely to degrade run-time performance because in these quick searches the cost of backbone estimation corresponds to a relatively large part of the total search cost.

Recent work [18] has investigated the feasibility of computing and approximating the backbone of the TSP. Despite some negative theoretical results, it has been suggested that much of the backbone appears to be present in close to optimal solutions, and that approximation methods based on good heuristics should be successful in obtaining and using such information to guide the search. Our work serves as additional empirical evidence supporting this view.

For future work, it would be interesting to investigate computationally cheaper alternatives to the PBFE scheme presented here, given the relevance of accurate PBFEs to the performance of backbone guided dynamic local search. Also, we chose to use PBFEs to bias the tie breaking of candidate flips because of our recent success in using tie breaking heuristics for additive DLS methods. However, multiplicative DLS methods rarely encounter tie breaking situations due to the finer granularity of their clause weight profiles. Therefore, it would be interesting to extend the work presented here by integrating PBFE directly into the DLS method's evaluation function, thus making our approach useful for multiplicative methods as well. In conclusion, we believe this work further illustrates the usefulness of backbone heuristics to improve local search performance in general, and now, DLS performance in particular.

References

1. Hoos, H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, San Francisco, California (2005)
2. Selman, B., Levesque, H.J., Mitchell, D.: A new method for solving hard satisfiability problems. In Rosenbloom, P., Szolovits, P., eds.: Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92), Menlo Park, California, AAAI Press/The MIT Press (1992) 440–446
3. 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., AAAI Press/The MIT Press (1993) 40–45
4. 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, AAAI Press/The MIT Press (2004) 191–196
5. Tompkins, D., Hoos, H.: Warped landscapes and random acts of SAT solving. In: Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics, AI&M 2004, Fort Lauderdale, Florida (2004)
6. 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
7. Schuurmans, D., Southey, F.: Local search characteristics of incomplete SAT procedures. In: Proceedings of the 1tth National Conference on Artificial Intelligence (AAAI'00), Austin, Texas, AAAI Press/The MIT Press (2000) 297–302
8. Schuurmans, D., Southey, F., Holte, R.C.: The exponentiated subgradient algorithm for heuristic Boolean programming. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle, Washington, Morgan Kaufmann (2001) 334–341
9. Hutter, F., Tompkins, D., Hoos, H.: Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Proceedings of CP'02. Volume 2470 of Lecture Notes in Computer Science., Ithaca, New York, Springer (2002) 233–248
10. Anbulagan, Pham, D.N., Slaney, J., Sattar, A.: Old resolution meets modern SLS. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'05), San Jose, California, AAAI Press/The MIT Press (2005) 354–359
11. Pham, D.N., Thornton, J., Sattar, A., Ishtaiwi, A.: SAT-based versus CSP-based constraint weighting for satisfiability. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'05), San Jose, California, AAAI Press/The MIT Press (2005) 455–460
12. Ferreira Jr., V., Thornton, J.: Tie breaking in clause weighting local search for sat. In: 18th Australian Joint Conference on Artificial Intelligence, Sydney, Australia (to appear)
13. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity from characteristic phase transitions. Nature **400** (1999) 133–137
14. Zhang, W., Rangan, A., Looks, M.: Backbone guided local search for maximum satisfiability. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco, Mexico, Morgan Kaufmann (2003) 1179–1186
15. Zhang, W., Looks, M.: A novel local search algorithm for the travelling salesman problem that exploits backbones. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05). (to appear)
16. Dubois, O., Dequen, G.: A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle, Morgan Kaufmann (2001) 248–253
17. Zhang, W.: Configuration landscape analysis and backbone guided local search. Part I: Satisfiability and maximum satisfiability. Artificial Intelligence (158) (2004) 1–26
18. Kilby, P., Slaney, J., Walsh, T.: The backbone of the travelling salesperson. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05). (to appear)
19. 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, Washington, AAAI Press (1994) 337–343
20. Wei, W., Selman, B.: Accelerating random walks. In: Proceedings of CP'02. Volume 2470 of Lecture Notes in Computer Science., Ithaca, New York, Springer (2002) 216–232

21. Li, C.M., Anbulagan: Look-ahead versus look-back for satisfiability testing. In: Proceedings of CP'97. Volume 1330 of Lecture Notes in Computer Science., Springer (1997) 341–355
22. Hoos, H.: An adaptive noise mechanism for Walksat. In: Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02), Edmonton, Alberta, Canada, AAAI Press (2002) 655–660
23. Thornton, J.: Clause weighting local search for SAT. Journal of Artificial Intelligence Research (to appear)
24. Fujita, M., Slaney, J., Bennett, F.: Automatic generation of some results in finite algebra. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'93), Chambery, France, Morgan Kaufmann (1993) 52–57
25. 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