Chapter 9

ADAPTIVE MEMORY SEARCH GUIDANCE FOR SATISFIABILITY PROBLEMS

Arne Løkketangen[1] and Fred Glover[2]
[1]*Molde College, Molde, Norway, arne.lokketangen@himolde.no;*
[2]*Leeds School of Business, University of Colorado at Boulder, USA, fred.glover@colorado.edu*

Abstract: Satisfiability problems (SAT) are capable of representing many important real-world problems, like planning, scheduling, and robotic movement. Efficient encodings exist for many of these applications and thus having good solvers for these problems is of critical significance. We look at how adaptive memory and surrogate constraint processes can be used as search guidance for both constructive and local search heuristics for satisfiability problems, and how many well-known heuristics for SAT can be seen as special cases. We also discuss how adaptive memory learning processes can reduce the customary reliance on randomization for diversification so often seen in the literature. More specifically, we look at the tradeoff between the cost of maintaining extra memory search guidance structures and the potential benefit they have on the search. Computational results on a portfolio of satisfiability problems from SATLIB illustrating these tradeoffs are presented.

Keywords: Adaptive Memory, Local Search, Satisfiability Problems

1. Introduction

Many important real-world problems can be represented as satisfiability problems. These include planning, scheduling and robotic movement, and efficient encodings exist for many of these. Efficient solvers for these problems are thus of critical significance. SAT has thus received substantial attention in recent years, and efficient SAT solvers exist.

What sets SAT apart from other combinatorial optimization problems is that SAT is basically a feasibility problem. Once a variable assignment is

found that satisfies all the clauses (see Section 2), the problem is solved, and this condition is readily detected. In SAT there is thus no guidance from the normal objective function. Guidance is customarily instead based on the amount of infeasibility, usually by counting the number of unsatisfied clauses for a given solution, possibly modified by the clause length.

There are many approaches to solving the SAT problem. Constructive methods range from complete tree-search (DPL - Davis-Putnam-Loveland , see Davis , Logemann and Loveland, 1962, Davis and Putnam, 1960), to constructive heuristics like GRASP (Resende and Feo, 1996) and surrogate constraint based learning heuristics (Løkketangen and Glover, 1997). Most heuristic solvers for SAT are based on local search starting from randomly or otherwise constructed starting solutions. For a nice overview of many of the heuristics for SAT, see Hoos (1998). See also Section 4.

The work presented in this paper is based on previous work by the authors on the satisfiability problem, where the basic framework and search mechanisms was developed . For details, see Løkketangen and Glover (1997).

We will show how the judicious use of surrogate constraint based local search guidance, with the augmentation of adaptive memory structures for short and long-term learning and forgetting, provides superior search guidance, at an extra computational cost per iteration. Many of the popular heuristics for SAT can be derived as special cases, and we show that additional heuristic power results by considering more general forms of this guidance framework. We also discuss how adaptive memory processes can reduce the customary reliance on randimazation for diversification so often seen in the literature.

We report computational tests that compare solution attempts both in terms of execution time and number of local search steps, using a set of state-of-the-art local search heuristics that are augmented by varying degrees of search guidance, and adaptive memory capabilities. The tradeoffs between the increased solution time required by fuller reliance on adaptive memory, and the reduced numbers of iterations that are required to obtain feasible solutions, are illustrated on a portfolio of satisfiability problems taken from SATLIB (see SATLIB).

The layout of this extended abstract is as follows. This introduction is followed in Section 2 by a description of the SAT problem. In Section 3 we look at surrogate constraints, while a brief outline of SAT solvers is presented in Section 4. Our choice of search guidance mechanisms is described in Section 5, and the computational results in are in Section 6, followed by the conclusions in Section 5.

2. The SAT Problem

The Satisfiability problem originates from the realm of logic theorem proving, and was the first problem proven to be NP-Complete (Cook 1971). All other NP-Complete problems can be reduced to SAT in polynomial time. The SAT problem can be defined as follows. Given the logical function, consisting of combinations of disjunctions, conjunctions and negations of a set of variablesals (x_1, …, x_N), then the SAT problem is to find a set of truth assignments to the literals that will make $\Phi(x)$ *true* (or false):

$$SAT: \quad \Phi(x) = \Phi(x_1, ..., x_N) = \begin{cases} true \\ false \end{cases}$$

The logical function $\Phi(x)$ is usually represented in CNF, *Conjunctive Normal Form*. $\Phi(x)$ then consists of a set of conjunctions of clauses $c_i(\boldsymbol{x})$, written $\Phi = c_1 \wedge c_2 \cdots \wedge c_M$, where each clause is a disjunction of complemented and uncomplemented variables, called literals, with M being the number of clauses. As a simple example, let $\Phi(x)$ be the following formula containing 3 variables and 5 clauses:

$$\Phi(x)=(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_3)$$

2.1 Mathematical Formulation

To get to the more customary mathematical formulation, replace true/false with 1/0, disjunction with +, representing each conjunction as a separate constraint row. Let literals be represented by x_j and their complements by $1 - x_j$. This gives us:

$\mathbf{A}x \geq b$
x binary
where $\mathbf{A}$ is an $m*n$ matrix of 0´s, 1´s and –1´s
and b and x are n*1 column vectors.

The i[th] constraint of the system,

$$\mathbf{A}_i x \geq b_i$$

has the property that the number of –1´s in the row vector $\mathbf{A}_i$ equals $1 - b_i$, where b_i is an integer ≤ 1.

To get a more convenient representation, we split each variable x_i into its complemented and uncomplemented occurrences, we get the following constraint set for the example, with the variable pair z_i and $z_{i\#}$ represents x_i:

$$
\begin{array}{llllll}
z_1 + z_2 & & & & \geq 1 & (w_1) \\
z_1 \quad + z_3 & & & & \geq 1 & (w_2) \\
\quad z_2 \quad + z_3 & & & & \geq 1 & (w_3) \\
& & z_4 \quad + z_5 & & \geq 1 & (w_4) \\
z_1 & & & + z_6 & \geq 1 & (w_5)
\end{array}
$$

(The w's are weights used for learning purposes in surrogate constraint evaluations, see the next section):

Our final model is then

$$Dz \geq 1 \qquad (2.1)$$
$$z_i + z_{i\#} = 1 \qquad (2.2)$$

where D is the 0-1 matrix obtained by substituting the z's for the x_i's. The last constraint (2.2) is handled implicitly in the search heuristics we describe.

3. Surrogate Constraints

A Surrogate Constraint (SC) is a weighted linear combinations of the original problem constraints (Glover, 1977), and provides a powerful way to capture constraint information to be used for search guidance. The basic use of SC methods for both constructive and local search heuristics for SAT is described in Løkketangen and Glover (1997).

Given the transformed constraint set of (2.1), we introduce a nonnegative vector w of weights w_i, to generate a surrogate constraint

$$sz \geq s_o$$

where $s = \mathbf{wD}$ and $s_o = \Sigma w$. The surrogate constraint therefore results by weighting each constraint row j by w_i and summing. Assuming (as is the case initially in our searches) that all the w_i's are 1, we get the following surrogate constraint from the example (other weightings will of course result in a different SC):

$$3z_1 + 2z_2 + 2z_3 + z_4 + z_5 \quad + z_6 \quad \geq 5$$

This surrogate indicates that the best would be to set z_1 to 1, and thus select x_1 to be *true*. If one also considers that the pair (z_1, z_4) really represents the same variable, x_1, and both cannot simultaneously be set, we can form a *derived* surrogate constraint by replacing s_j with $s_j - \text{Min}(s_j, s_{j\#})$ in the surrogate constraint evaluation. We then get:

$$2z_1 + z_2 + z_3 \geq 2$$

indicating even stronger that x_1 should be set to *true*. In the event of ties, we choose the variable with the maximum $s_j - s_{j\#}$ value.

We will use (derived) surrogate constraint based search guidance for the local searches, augmented by various levels of adaptive memory capabilities.

4. On Solving SAT

In this section we will look briefly at the most common constructive and iterative local search paradigms for solving SAT, and some of the existing solvers. For a nice overview of many of the heuristics for SAT, see Hoos (1998), and the bibliography at SATLIB. There are myriads of different solvers for SAT, but most falls into one of the broader categories, and shares most features with other solvers.

4.1 Constructive Methods

Within the constructive solver class, there is a big distinction between *complete* methods, guaranteeing to prove that a formula is satisfiable or not, and *heuristic methods* designed to try to find a solution if one exists. The best known complete method for SAT is DPL – Davis-Putnam-Loveland (Davis, Logemann and Loveland, 1962, Davis and Putnam, 1960). This is a deterministic tree-search with backtracking. The problem with this approach is the limited size of the problem instances that can be solved in reasonable time.

A constructive search usually contains the following elements and search flow:

1. All variables initially unassigned
2. Construct solution by assigning a truth-value to one variable at the time. (Neighborhood is the set of remaining unassigned variables).
3. When no feasible assignments can be made:
   - Full Backtrack (complete method - DPL)
   - Limited backtracking with restart - (DPL with restarts)
4. Finish construction and:
   - Submit to (limited) local search and restart – (GRASP, SC-Learn)
5. Need move evaluation guidance. This is usually based on change in feasibility..
6. For restart-methods, guidance should be modified by history (SC-Learn)

Among constructive heuristic methods are GRASP (Resende and Feo, 1996), DPL with restarts (Gomes, Selman and Kautz, 1998), and SC-Learn, a surrogate constraint based learning heuristics (Løkketangen and Glover, 1997).

DPL with restart (Gomes, Selman and Kautz, 1998) is a DPL-based tree-search with limited backtracking, and only in the bottom of the search tree. This work is inspired by the phenomenon of heavy-tailed cost distributions, in that at any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been solved before (Gomes et. al. 1998). Instead of risking spending such a long time futilely searching, the search is restarted, but different, controlled randomized choices are made in the new search.

GRASP - Greedy Randomized Adaptive Search Procedure (Resende and Feo, 1996),. This is a constructive heuristic followed by a short greedy local search, trying all combinations of improving flips. It can be called a shotgun method, as its aim is generate a diverse set of solutions quickly, some of which might be the solution. The basic heuristic for assigning one variable value is:

– For each unassigned variable, count the number of clauses that are satisfied by assigning it True (and similarly for False).
– Sort the values. Select randomly among the top half evaluations (or max 50).

This corresponds to a basic Surrogate Constraint using uniform weighting,, and no normalization. There is also no learning, or use of memory, between restarts.

SC-Learn (Løkketangen and Glover, 1997) uses adaptive memory structures to learn between runs. More specifically, it gives added focus on the clauses that have been difficult to satisfy so far. Surrogate constraints are used for move evaluations.

4.2 Iterative Local Search Methods

All of the iterative local search methods for SAT are incomplete methods, in that the non-existense of a solution can not be proven. An iterative local search usually contains the following elements and search flow:

1. All variables are assigned a truth-value at all times
2. The starting solution (or starting point) is usually based on a random assignment to the variables or based on a construction heuristic.
3. A move is the flip of a variable. A flip means assigning the opposite value to a variable. (i.e. change $1 \rightarrow 0$ or $0 \rightarrow 1$).
4. The search neighborhood is either the full set of variables, or just those that appear in unsatisfied clauses.
5. Move evaluation is based on changes in feasibility. I.e. select moves that reduce the number of unsatisfied clauses. This measure can be modified by history.

6. The move selection is greedy (i.e. take the best move according to the move evaluation).
7. A random restart is applied after a certain number of moves, to diversify the search after stagnation
8. The stopping criterion is a simple time limit, a cutoff on the number of allowable flips or the identification of a solution.

There are extremely many iterative local search methods for SAT. Among the first,and most well-known, are GSAT (Selman, Levesque and Mitchell, 1992), and the whole family of search methods derived from it. (Walksat, GSAT+Tabu, Novelty,…). For an overview, see Hoos (1998). These methods are generally very simple and have fast iterations. Random restarts are usually employed when restarting.

GSAT starts from a randomly generated starting solution. The moves are variable flips. Move evaluation is based on the change in the number of satisfied clauses. (Choose randomly among ties). Don't allow downhill (worsening) moves. Do a random restart after a certain number of flips.This corresponds to using the derived surrogate constraint, without the SC choice rule (for ties).

Novelty (McAllester, Selman and Kautz, 1997). This is considered one of the best local search heuristics for SAT. Each iteration a violated clause is selected randomly. Then the best (in terms of improved infeasibility) variable to flip in this clause is identified. (In the case of ties, select the least recently flipped variable). If this variable is not the most recently flipped, flip it. Otherwise select the next best variable with probabibility p, and with probability 1-p select the best variable. This heuristic works very well on random 3-sat.

SC-Learn (Løkketangen and Glover, 1997) starts from a randomly generated starting solution. The moves are variable flips. A simple tabu search is added to avoid move reversals. Diversification is with the modification of clause weights used in the surrogate constraint based move evaluations.

5. Search Guidance Structures and Mechanisms

In this section we will look at enhancements to the iterative SC-Learn heuristics (Løkketangen and Glover, 1997). More specifically we will look at adding forgetting to the learning, sensitivity to learning weights and the introduction of controlled randomization in the move selection. We will also look at examples of how the different search mechanisms are far from independent, and that when adding a new search mechanism, the already implemented ones can change behaviour.

All the gathering of information about the search development during the search process, updating of the adaptive memory structures, and the processing of the gathered information takes additional computing time. The purpose of this is to provide better search guidance, thus needing fewer iterations to get to the solution. (Note that the solution can be reached in at most N steps, where N is the number of variables. The actual number of flips needed by many of the local search methods are often several orders of magnitude larger). It is therefore of interest to look at this trade-off between randomization and use of adaptive memory structures for search guidance and diversification. A new heuristic, SC-RN is developed, and will be described below.

5.1 Learning

We use frequency based information to modify the clause weights in a judicious way. Given the current solution vector, we know that at least one of the variables in one of the violated clauses has the wrong value, and hence place an emphasis on changing values of these variables. We do this in the SC framework by increasing the weights of the *violated clauses* every iteration. (This was also used in Løkketangen and Glover, 1997, and a different weighting scheme was tried in Frank, 1996). We have found that the increment used is not important, and a value of 1 is used in the tests. Preliminary testing has also shown that resetting these weights at fixed intervals has no discernible effect.

5.2 Forgetting

The accuracy of the information embedded in the clause weights vanes over time, and should have decreasing impact. This is accomplished by increasing the weight used in the learning process slightly every iteration. This leads to a discounting of the oldest values. Preliminary testing have shown that the value of the forgetting (or discounting) increment likewise is not important as long as it is significantly smaller than the actual weights.

5.3 Tabu Tenure and the Non-independence of Search Mechanisms

Our search uses the basic tabu criterion of not to flip a variable that has recently been flipped. (A good treatment of tabu search is in Glover and Laguna, 1997). One problem is to determine the optimal, or best, tabu tenure in terms of some problem parameter, like the number of variables. Mazure, Saïs and Grégoire (1997) added a simple tabu criterion as described above to GSAT (naming the new method TSAT). One of their findings was a linear

relationship between the optimal tabu tenure and problem size, according to the following formula:

$$TT_{OPT} = 0.01875 * N + 2.8125$$

with N being the number of variables for random hard 3-SAT instances.

We similarly tried different values of TT combined with the basic learning scheme on the test instance *aim-50-2_0-yes-2* taken from SATLIB. This problem has 50 variables, and is not very difficult. Table 9.1 shows the rate of success for 5 runs from random starting positions with a maximum of 5000 flips, and varying TT, using a weight increment of 1. The table indicates a best TT of 10.

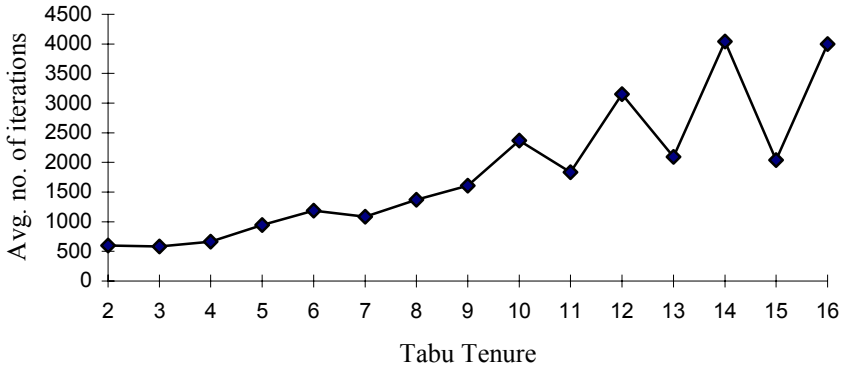

Figure 9.1. Search performance for TT with discounted learning

When rerunning the same test, but with discounting of the learning weights (using an increment of 0.1), we get the results of Table 9.1. The solution is now found for all values of TT between 2 and 16. In Figure 9.1 is shown the average search effort (in terms of flips) for each of the TT's where a solution was found for all runs. This graph shows clearly that even though the search always finds the optimal solution for a TT value in the range 2 to 17, it uses far more iterations for the larger values of the TT. The best values for the TT are 2 or 3, while without discounting this value is much larger. The addition of the discounting mechanism clearly changes the effective TT values, and . has a great stabilizing effect.

Table 9.1. Effect of TT with simple learning

TT	Rate of Success
5	2/5
10	5/5
15	4/5
20	3/5

Table 9.2. Effect of TT with discounted learning

TT	Rate of success	Avg. no of iterations
1	9/10	-
2	10/10	599
3	10/10	584
4	10/10	664
5	10/10	946
6	10/10	1189
7	10/10	1082
8	10/10	1372
9	10/10	1606
10	10/10	2367
11	10/10	1832
12	10/10	3152
13	10/10	2095
14	10/10	4040
15	10/10	2041
16	10/10	3996
17	8/10	-
20	6/10	-

For the tests in Section 6 a short, fixed tabu tenure of value 4 was used. This choice avoids stumbling back in the previously visited search path, given that the local neighborhood evaluations typically consist of many equal values and forms a local plateau, even when learning and forgetting is employed. The value of the tabu tenure does not appear sensitive to changes within a modest range. The tabu mechanism thus has a very local function, while the learning and forgetting provides the necessary diversification.

5.4 Probabilistic Move Acceptance

The only diversification present in our SC guided search is the random selection between the moves that tie for the best evaluation. More than one move may tie for this, but many fewer ties occur when the learning and

forgetting mechanisms are employed than when only the SC evaluations are used. This leads to less diversification in the search.

Since our guidance is not perfect, there is no guarantee that the actual best move gets the best evaluation. Assuming our move evaluations have some merit, we select probabilistically among the best moves, disproportionately favoring those with higher evaluations (see Løkketangen and Glover, 1996). The process selects the best move with probability p, and if the move is rejected then applies the same selection rule to the next best, etc. This gives rise to an exponentially decreasing move acceptance.

5.5 Choice of Starting Solution

We ran preliminary tests comparing starting from randomly generated truth assignments to the variables, and the best solution found after 10 iterations of the constructive search with learning as described in Løkketangen and Glover (1997). With 10 runs on each problem, we did not find any significant difference between the results. One reason why the constructed starting point did not yield better results might be that the different solutions from the constructive phase for the same problem tend to be similar, such that bad trends might be amplified.

5.6 A New Method – SC-RN

Our full method with learning and forgetting spend a lot of time in maintaining the adaptive memory structures.

Move evaluation is very costly in the SC Learn method with added forgetting. The problem is the "forgetting" part, as this increments the learning weights. This requires a full evaluation of the SC's are after every flip. (Preliminary testing using incremental update, ignoring the slightly changed weight values, gave very bad results).

Considering that in every unsatisfied clause at least one of the variables has the wrong value, we form a reduced neighbourhood consisting only of the variables in the unsatisfied clauses. The new method can be labelled SCLFPRN – ***S**urrogate **C**onstraint based move evaluation with **L**earning and **F**orgetting, **P**robabilistic move acceptance and **R**educed **N**eighborhood.* For short we call it SC – RN.

6. Computational Results

We investigate the tradeoff issues when applying different levels of adaptive memory mechanisms, and also when using guidance based on surrogate constraints for various neighborhood sizes. The testing reported in this section is intended to illustrate these tradeoffs. For purposes of

comparison we chose three heuristics. As a representative for a method using rather myopic search guidance, but with very fast iterations, we chose Novelty (McAllester, Selman and Kautz, 1997), hereafter called *NOV*. The iterative SC-Learn heuristic (Løkketangen and Glover, 1997) was augmented with forgetting and probabilistic move acceptance and called *SC-W*. As an intermediary we used the method described in Section 5.6, *SC-RN*, as it uses a smaller neighbourhood than SC-W, thus basing its decisions on less information.

6.1 Test Setup

Testing has been done on benchmark SAT-problems, both structured and randomized, taken from SATLIB (see SATLIB). The problem sizes range from 48 variables ∗ 400 clauses to 4713 variables ∗ 21991 clauses.

As all the methods include a probabilistic parameter, p, we chose one of the simpler test cases from SATLIB, *jnh*, to tune this parameter individually for each heuristic, and keep it fixed for all subsequent tests. The best values (and thus the values used) for p are shown in Table 9.3.

Table 9.3. Best values for p

	p
SC-W	0.8
SC-RN	0.6
NOV	0.8

We tested each heuristic from random starting solutions, using different random seeds, with 10 runs per test case. We allowed in general up to 10^7 iterations for Novelty, 10^6 for SC-RN and 10^5 for SC-W. Restarts were not used. A fixed TT of 4 was used. The results are both in term of flips and time. The tests have been run on a 300 MHz Pentium III running Windows 98 (The sub-second timing is thus somewhat inaccurate).

6.2 Test Results

The results are shown in Tables 9.4, 9.5 and 9.6. For each test case is shown the size (in terms of variables and clauses), and the average number of flips and the corresponding time used for each of the three heuristics.

A dash in the flips column indicates that the heuristic failed to find a solution for at least on of the tries, with the actual number of successful runs are shown in the flips column.

Table 9.4 shows the results for some of the smaller test cases. Not surprisingly NOV is very good on the random 3-SAT instance uf100-100, spending virtually no time in finding the solution. SC-RN spends more flips

and time, while SC-W is comparable with NOV in terms of flips, but not on time.

The rest of the test cases in the table are structured, par-8-1 is a parity problem encoding, and par-8-1-c is a compressed version of he same problem. The ais test cases are taken from a problem in music theory (all interval series). On these problems SC-W is clearly most successful, only having 2 unsuccessful runs, while NOV does rather badly.

Table 9.4. Smaller test-cases

Problem	**vars**	**clauses**	**NOV f**	**NOV t**	**SC-RN f**	**SC-RN t**	**SC-W f**	**SC-W t**
max-flips			10^7		10^6		10^5	
uf100-100	100	430	371	0	1090	0.1	397	0.01
par8-1-c	64	254	1149	0	4484	0.5	1070	.13
par-8-1	350	1149	1/10	-	8/10	-	9/10	-
ais6	61	581	0/10	-	3326	0.1	465	0.1
ais8	265	5666	0/10	-	187850	9.5	6269	1
ais12	1141	10719	0/10	-	0/10	-	9/10	-

Table 9.5. Intermediate test-cases

Problem	**vars**	**clauses**	**NOV f**	**NOV t**	**SC-RN f**	**SC-RN t**	**SC-W f**	**SC-W t**
max-flips			10^7		10^6		10^5	
uf200-100	200	860	8860	0.2	64608	0.1	397	0.1
flat100_3_0	300	1117	9060	0.2	5/10	-	3882	1.5
sw100-10-p4	500	3100	2/10	-	89502	3.7	11075	10
sw100-10-p6	500	3100	2/10	-	5/10	-	4/10	-
anomaly	48	261	124	0	164	0	135	0
medium	116	953	852	0.1	331	0	445	0.1

Table 9.5 shows the next set of results, for small to intermediate test cases. Interestingly, SC-W uses many fewer flips on the random 3-SAT instance uf200-100 than NOV, even being faster. In general SC-W needs fewer flips on most of the instances. NOV does reasonably well on these instances, while SC-RN is slightly worse.

Table 9.6. Large structured test-cases

Problem	vars	clauses	NOV f	NOV t	SC-RN f	SC-RN t	SC-W f	SC-W t
max-flips			10^7		10^6		10^5	
bw_large.a	459	4675	60299	1.4	5571	1	5132	6.5
bw_large.b	1087	13772	4.79 M	105	31048	10.4	56611	198
logistics.a	828	6718	0/10	-	2/10	-	18922	50
logistics.b	843	7301	0/10	-	102465	14	23565	50
logistics.c	1141	10719	0/10	-	91739	59	21248	103
logistics.d	4713	21991	0/10	-	88091	26	72600	470

In Table 9.6 are results for the runs on large structured instances. SC-W seems very good on these, while NOV fails on the logistics instances. SC-RN does quite well, only failing on logistics.a.

As can be seen from the results, SC-W solves most test cases, but spends a long time per iteration. It seems particularly good on the large structured instances. This is according to expectations, as the learning mechanisms learn *structure*.

SC-RN needs more flips, and is more unstable. It solves most structured instances, while showing bad performance on some random problem classes. Each iteration is much faster than SC-W. This is the behaviour we would expect, as it bases its search guidance on a smaller neighbourhood, and thus less information, than SC-W

NOV fails on bigger instances, and on instances having a lot of structure. This is not very surprising, as NOV does not have any memory mechanisms to capture structure. It does use recency information, but only within a clause, and then only to choose between the two (locally) best variables. In clauses with only 3 variables, this seems to work very well, but NOV clearly has problems with problems having longer clauses and problems with structure.

7. Conclusions

The heuristics we describe rely on a set of advanced mechanisms for their working. Testing clearly shows that care should be taken when combining mechanism, often necessitating changes in their customary settings.

The computational testing clearly illustrates that the use of surrogate constraints provides good guidance. The addition of simple learning gives greatly improved results. Discounted learning (forgetting) is effective, and has a stabilizing effect. As is expected, best results are obtained for the structured test cases. The extra cost in maintaining memory and guidance structures thus seems well spent on several classes of test instances.

Appropriate guidance structures based on surrogate constraint evaluations, incorporating adaptive memory guidance based on recency, frequency, learning and forgetting – thus yield results that compare favourably with state-of-the-art randomized local search heuristics for SAT.

References

Cook, S.A. (1971) "The Complexity of Theorem-Proving Procedures," *Proceedings of the Third ACM Symposium on Theory of Computing*. 151–158.

Davis, M., G. Logemann and D. Loveland (1962) "A Machine Program for Theorem Proving," *Comm. ACM*, 5:394–397.

Davis, M. and H. Putnam (1960) "A Computing Procedure for Quantification Theory," *Journal of ACM*, 7:201–215.

Frank, J. (1996) "Weighting for Godot: Learning Heuristics for Gsat," *Proceedings of the13th International Conference on Artificial Intelligence,* 338–343.

Hoos, H. (1998) "Stochastic Local Search - Methods, Models, Applications," Ph.D. Dissertation, Fachbereich Informatik, Technische Universität Darmstadt.

Glover, F. (1977) "Heuristics for Integer Programming using Surrogate Constraints," *Decision Sciences* 8:156–166.

Glover, F. and M. Laguna (1997) *Tabu Search*. Kluwer Academic Publishers.

Gomes, C., B. Selman, K. McAloon and C. Tretkoff (1998) "Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems". In Proceedings AIPS-98.

Gomes, C., B. Selman and H. Kautz (1998) "Boosting Combinatorial Search Through Randomization," In Proceedings AAAI98.

Løkketangen, A. and F. Glover (1997) "Surrogate Constraint Analysis–New Heuristics and Learning Schemes for Satisfiability Problems," Satisfiability Problem: Theory and Applications. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 35:537–572.

Løkketangen, A. and F. Glover (1996) Probabilistic Move Selection in Tabu Search for 0/1 Mixed Integer Programming Problems. Metaheuristics: Theory and Applications, Kluwer Academic Publishers, 467–489.

McAllester, D., B. Selman and H. Kaut (1997) "Evidence for Invariants in Local Search," In Proceedings AAAI97.

Mazure, B., L. Saïs and É. Grégoire (1997) Tabu Search for SAT. In Proceedings AAAI 97.

Resende, M. and T. Feo (1996) "A GRASP for Satisfiability," in Cliques, Coloring and Satisfiability. The Second DIMACS Implementation Challenge, D.S. Johnson and M.A. Trick (eds.), DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 26:499–520, American Mathematical Society.

SATLIB – The Satisfiability Library. http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/

Selman, B., H.J. Levesque and D. Mitchell (1992) "A New Method for Solving Hard Satisfiability Problems," *Proceedings AAAI 92,* 440–446.