Guided Local Search for solving SAT and weighted MAX-SAT Problems

Patrick Mills (millph@essex.ac.uk) and Edward Tsang (edward@essex.ac.uk)
Department of Computer Science, University of Essex

24th February 1999

Abstract. In this paper, we show how Guided Local Search (GLS) can be applied to the SAT problem and show how the resulting algorithm can be naturally extended to solve the weighted MAX-SAT problem. GLS is a general, penalty-based meta-heuristic, which sits on top of local search algorithms to help guide them out of local minima. GLS has been shown to be successful in solving a number of practical real life problems, such as the travelling salesman problem, BT's workforce scheduling problem, the radio link frequency assignment problem and the vehicle routing problem. We present empirical results of applying GLS to instances of the SAT problem from the DIMACS archive and also a small set of weighted MAX-SAT problem instances and compare them against the results of other local search algorithms for the SAT problem.

Keywords: SAT problem, Local Search, Meta-heuristics, Optimisation

1. Introduction

Guided Local Search [41] has been applied to a number of real life problems, including BT's workforce scheduling problem [40], the travelling salesman problem [43], the vehicle routing problem [20], function optimization [42] and the radio link frequency assignment problem [44].

GLS is a meta-heuristic, which sits on top of local search procedures for helping them escape from local minima. GLS is a generalisation of the GENET neural network [4], for solving constraint satisfaction and optimisation problems. Recently, Lau and Tsang [22, 23] have shown how GLS can be sat on top of a specialised Genetic Algorithm, resulting in the Guided Genetic Algorithm (GGA, [21]) for solving constraint satisfaction and optimisation problems. They apply GGA to a number of problems including the processor configuration problem [23] and the generalised assignment problem [22]. In this paper, we show how GLS can be successfully applied to the SAT problem and weighted MAX-SAT problem.

The SAT problem is an important problem in mathematical logic, inference, machine learning, VLSI engineering, and computing theory [14], and has been the focus of a lot of successful research into local search algorithms. The SAT problem is a special case of a constraint

satisfaction problem (CSP) where the variables take boolean values and the constraints are disjunctions (logical OR) of literals (variables or their negations). The weighted MAX-SAT problem is an extension of the basic SAT problem, where each clause has a weight associated with it, and the goal is to minimise the sum of the weights of violated clauses.

In this paper, we show how GLS can be used to solve the SAT and weighted MAX-SAT problems, by sitting it on top of a local search algorithm, which is similar to GSAT [35] without restarts. We give an evaluation of GLS's performance on benchmarks from the DIMACS archive, and a set of weighted MAX-SAT problems, comparing the results against the results of similar local search algorithms.

2. The SAT and weighted MAX-SAT problem

The SAT problem is an important class of constraint satisfaction problem [39], where the domain of a variable is always { false, true }, and each constraint is a clause. The SAT problem is important in solving many practical problems in mathematical logic, constraint satisfaction, VLSI engineering and computing theory [14].

The SAT problem and it's extension, the weighted MAX-SAT problem are defined as follows:

- A set of m (boolean) variables, $Z = \{x_1, x_2, .., x_m\}$, each of which may take the values true or false.
- A set of n clauses, $C = C_1, C_2, .., C_n$, each of which is a disjunction of a set of literals (a variable or it's negation), e.g. $x_1 \vee \neg x_2 \vee x_3$. In the weighted MAX-SAT problem, a weight w_{C_i} is associated with each clause.
- The goal of the SAT problem is to find an assignment of values to variables, if one exists, where all the clauses are satisfied (evaluate to true) or prove it is unsatisfiable if no valid assignment exists (currently only complete algorithms can prove unsatisfiability). In the weighted MAX-SAT problem the goal is to minimise the sum of weights of violated clauses.

Both complete and incomplete algorithms have been used to solve the SAT problem. Of the complete algorithms, one of the best known is the Davis-Putnam procedure [6], which is based on resolution. Of the incomplete stochastic algorithms, the best known is probably GSAT (first reported in [35]), based on steepest gradient descent and the related

WalkSAT [33] based on random walk with greedy variable selection heuristics.

Whilst a lot of work has been done on the basic SAT problem, only a small amount of work has been done on the weighted MAX-SAT problem by comparison. Examples include [18], where GSAT is extended to solve instances based network steiner trees, and their results compared against non-MAX-SAT techniques for the native network steiner tree problems. Resende et al. ([29]) show how GRASP can be applied to the instances of the weighted MAX-SAT problem based on soluble and insoluble instances of the jnh* problems from the DIMACS benchmarks archive with random weights associated with each clause. Borchers et al. ([1]) show how a branch and cut algorithm can be used to solve the MAX-SAT and weighted MAX-SAT problems. Recently, Wah and Shang ([45]) have shown how DLM how can be applied to the same set of instances of the weighted MAX-SAT problem as GRASP with excellent results. Later, in this paper we also give results compared with GRASP and DLM for these problems.

3. Other methods related to GLS

GENET ([46], [5], [4]), the direct ancestor of GLS, is a Min-Conflicts ([26]) based repair algorithm with a simple weight update scheme for solving constraint satisfaction problems (CSPs), inspired by neural networks and the earliest such scheme we know of which has been applied to solving CSPs, where the weights of all constraints are increased each time the algorithm converges to a local minimum. Morris ([27]) describes a very similar scheme called Breakout, which increases the weights of all violated constraints, every time there are no local moves available which will decrease the objective function. Selman and Kautz ([32]) modify GSAT to include clause weights, increasing the weights of all clauses which are unsatisfied at the end of each try (when the maximum number of flips of variable values has been exhausted and the algorithm is restarted from a new point in the search space) of GSAT. Hampson and Kibler ([16]) suggest a heuristic for deciding when to restart, based on the number of flips required to reach the current plateau, according to how many flips it took to reach the current plateau, from the initial solution. Later, Cha and Iwama ([2]) suggest a scheme called weight, which is the equivalent to breakout. Frank ([8]) increases the weights of violated clauses after each flip, and later ([9]) refines his scheme, with decaying weights, where after each flip, weights of all clauses are multiplied by a constant very close to 1.0. Recently, Shang and Wah ([38]) have applied DLM to the weighted MAX-SAT

and SAT problem with excellent results. DLM could be seen to be quite similar to GLS, although it does not use selective penalisation (see Section 5.2), and originated from theoretical mathematical foundations, whereas GLS is the result of many years of research based on empirical results. The two methods are related by the use of some kind of augmented objective function, with extra terms to guide the search out of local minima (Lagrangian multipliers in DLM, and penalties in GLS). So far DLM has been applied to a number of problems (e.g. nonlinear integer programming problems, nonlinear mixed integer programming problems, and the design of multiplierless filter banks), as well as the SAT problem (see [36] and [49] for a full description), whilst the GLS framework is more general and has also been applied to a wide variety of other problems (and local search algorithms, for example in the travelling saleman problem with 2-opting, [43]) as well (see Section 1).

4. Local Search for the SAT problem

To cast the SAT problem as a local search problem, we simply want to minimise the number of unsatisfied clauses, which we will give as our basic objective function, g, given in (1), below.

$$g(\mathbf{s}) = \sharp\{c_i \mid c_i \text{ is violated by the truth assignment } \mathbf{s}\} \qquad (1)$$

The local search algorithm we use is based on ideas reported by Gent and Walsh in [12], who conduct extensive experiments on lots of different variations of the basic GSAT algorithm ([35]), and find HSAT to be the best hill climbing algorithm to use with GSAT at that time (although other schemes have now been divised which may outperform HSAT). Following Gent and Walsh's HSAT, we use a history mechanism in our local search algorithm, to help diversify the search as much as possible. We sort the variables into buckets of downward (if the variable is flipped the objective function will decrease) and sideways moves (if the variable is flipped the objective function remains the same). We then select the least recently flipped variable, which will result in a decrease in the objective function if one is available and flip it. Otherwise we select a so-called sideways move and flip that variable. We allow the maximum number of consecutive sideways moves to be a parameter ($smax$) to the local search algorithm. Pseudo code for our local search algorithm is given in Figure 1. In this paper $smax$ is always set to 20, unless otherwise stated.

In our implementation, we incrementally update the changes in the objective function, which would result when a variable is flipped, and

```
Function LocalSATSearch(s, g, smax)
{
        while(g(s) > 0)
        {
                Modify s, by flipping the least recently flipped
                variable, which will decrease the objective function
                g(s), if one exists.

                Otherwise, if no such variable exists, which will
                decrease the objective function g(s), modify s by,
                flipping the least recently flipped variable, which
                does not increase or decrease the objective function.

                If the last smax moves were sideways moves or no
                downwards or sideways move is available return s.
        }

        return s
}
```

Figure 1. Pseudo code for local search for the SAT problem

also the buckets storing the variables which if flipped result in downwards and sideways moves for the current search state, as this is more efficient than re-calculating every time.

5. Guided Local Search

Guided local search, (See [41] for more examples of applications of GLS) sits on top of a local search algorithm. Given a candidate solution **s** and a cost function g, a local search algorithm is expected to return a candidate solution **s'** according to its neighbourhood function. **s'** is hopefully better than **s** (i.e. hopefully $g(\mathbf{s'}) < g(\mathbf{s})$ in a minimisation problem).

5.1. Solution features

Solution features are used to distinguish between solutions with different characteristics, so that undesirable features can be penalised by GLS. The choice of solution features therefore depends on the type of

problem, and also to a certain extent on the local search algorithm. Each feature, f_i defined must have the following components:

- An Indicator function, indicating whether the feature is present in the current solution $\mathbf{s}$ or not:

$$I_{f_i}(\mathbf{s}) = \begin{cases} 1 & \text{if feature present in solution } \mathbf{s} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

- A cost function $c_{f_i}(\mathbf{s})$, which gives the cost of having the feature present in the solution $\mathbf{s}$ (in the SAT and MAX-SAT problems, this is only used when the search is in a local minimum).

- A penalty p_{f_i}, initially set to 0, used to penalise occurrences of the feature, in local minima of the augmented objective function, h.

In the SAT problem, we define features as violated clauses (to be elaborated below). A more complex feature, might be an edge between cities in the travelling salesman problem, where the cost of the feature is the length of the edge, and the indicator function is whether a candidate solution contains the edge or not.

5.2. Selective penalty modifications

When the local search algorithm returns a local minimum, $\mathbf{s}$, which does not satisfy the termination criteria (e.g. all clauses satisfied), GLS penalises all the features present in that solution which have maximum utility, $Util(\mathbf{s}, f_i)$ (as defined in Equation 3, below), by incrementing it's penalty.

$$Util(\mathbf{s}, f_i) = \frac{c_{f_i}(\mathbf{s})}{1 + p_{f_i}} \tag{3}$$

The idea is to penalise features with higher costs first, with the utility of doing so decreasing, the more times they are penalised. This is more beneficial for problems with lots of different features of different costs (e.g. the MAX-SAT problem), although even on problems with features with equal cost, this may still have some benefits in reducing the number of penalties imposed.

5.3. Augmented cost function

GLS uses an augmented cost function (see Equation 4, below), to allow it to guide the local search algorithm out of the local minimum, by penalising features present in that local minimum. The idea is to make

$$I_{f_i}(\mathbf{s}) = \begin{cases} 1 & \text{if clause } C_i \text{ violated by solution } \mathbf{s} \\ 0 & \text{otherwise} \end{cases}$$
$$c_{f_i}(\mathbf{s}) = \begin{cases} 1 & \text{if standard SAT problem} \\ w_{C_i} & \text{if weighted MAX-SAT problem} \end{cases}$$

Figure 2. Clauses as features in the SAT and MAX-SAT problems

the local minimum more costly than the surrounding search space, where these features are not present, thus guiding the local search algorithm out of the local minimum.

$$h(\mathbf{s}) = g(\mathbf{s}) + \lambda \sum_{i=0}^{n} I_{f_i}(\mathbf{s}) \cdot p_{f_i} \tag{4}$$

The parameter λ may be used to alter the intensification of the search for solutions. A higher value for λ will result in a more diverse search, where plateaus and basins in the search space are searched more coarsely; a low value will result in a more intensive search for the solution, where the plateaus and basins in the search landscape are searched with more care. In this paper λ is always set to 1, unless otherwise stated.

6. Guided Local Search for the SAT and MAX-SAT problems

To use Guided Local Search for a particular application, a set of features must be defined, and the local search algorithm must be called within the GLS procedure.

6.1. Features in the SAT and MAX-SAT problem

In the SAT problem unsatisfied clauses seem to make a good feature to penalise, as they are solution features, which we wish to eliminate from the final solution, and moves made by the local search algorithm (flipping a variable), will usually result in a different set of violated clauses.

This is similar to using constraints as features as in the Radio Link Frequency Assignment Problem [44]. We define the indicator and cost functions associated with clause features in Figure 2. The feature is present if the clause is unsatisfied, and not present if it is satisfied. The cost of any unsatisfied clause is 1 for the standard SAT problem (since we have no preference for any particular unsatisfied clauses to

```
Function GLSSAT1(s, g, λ, smax)
{
        ∀f_i, p_{f_i} = 0
        s = random assignment
        do
        {
                h = g augmented as in equation 4
                s = LocalSATSearch(s,h, smax)
                For each feature, f_i, with maximum utility, Util(s, f_i)
                        p_{f_i} = p_{f_i} + 1
        }
        while (h(s) > 0 or termination condition)
        return s
}
```

Figure 3. Pseudo code for GLSSAT1

be eliminated from the solutions, as we want to eliminate all of them) and w_{C_i} for the weighted MAX-SAT problem (as we want to eliminate the most costly violated clauses first). In a sense the standard SAT problem is a weighted MAX-SAT problem where the weight w_{C_i} of each clause is always 1 (note the weight w_{C_i} does not appear in the augmented objective function for the MAX-SAT problem (the same objective function is used as in the SAT problem); instead we leave GLS's selective penalisation to do the job of removing costly clauses from the solution).

6.2. Modifying the local search to use GLS

As GLS usually uses a simple augmented evaluation function, as defined in (4), we need to augment the objective function for the local search algorithm in Section 4, to take into account the penalty terms. This means passing the local search algorithm the augmented objective function h, instead of original objective function g, so that it searches for flips, which minimize h instead of minimising g. Pseudo code for the overall algorithm including GLS is given in Figure 3.

We shall call this new local search algorithm GLSSAT1. GLSSAT1 can solve most of the instances in the DIMACS archive with little difficulty. However, we have found that we are not able to solve the par8-*, par16-*-c, par16-*, par32-*-c, par32-*, f2000, g250-29, g125-17, hanoi4 or hanoi5 using GLSSAT1.

We believe that GLSSAT1 has difficulty with these harder problems, as it learns incorrect information about the search space, penalising the

```
Function GLSSAT2(s, g, λ, smax)
{
        ∀f_i, p_{f_i} = 0
        s = random assignment
        do
        {
                h = g augmented as in equation 4
                s = LocalSATSearch(s,h,smax)
                For each feature, f_i, with maximum utility, Util(s, f_i)
                        p_{f_i} = p_{f_i} + 1
                If (number of calls to LocalSATSearch mod 200 = 0)
                        For each p_{f_i}
                                p_{f_i} = p_{f_i} * (4/5)
        }       while (h(s) > 0 or termination condition)
        return s
}
```

Figure 4. Pseudo code for GLSSAT2

wrong features, so that the search landscape becomes more and more rugged (the reader might like to refer to [7] for a study of the search landscape of the SAT problem) as time goes on, making it slower and slower to traverse, crippling the local search algorithm and forcing GLS to do all the work, and making it difficult to find a path to the solution. Frank ([9]) uses a weight decay scheme to try to counter this sort of problem, by multiplying all the clause weights by a small constant close to 1.0, after every flip with some success on random problems. Shang and Wah [38] address the same kind of problem with their DLM A3 algorithm, by using a scheme whereby the langrangian multipliers are scaled down every 10000 iterations. They report that this stops the langrangian multipliers becoming too large, too quickly. We tried a very similar method, whereby the penalties are multiplied by 4/5 every 200 iterations of GLS's outer loop (rather than every 10000 iterations), which seems to have a very similar effect on GLS as Shang and Wah's scheme does on DLM, allowing us to solve some of the harder problems in the DIMACS archive. We call this version of the GLSSAT algorithm, GLSSAT2 (see figure 4).

Table I. Summary of results on the easier DIMACS SAT instances

Problem group	Percent successful		Average CPU time		Average flips	
	GLSSAT1	WalkSAT	GLSSAT1	WalkSAT	GLSSAT1	WalkSAT
ssa*	100.0	100.0	0.655	0.137	71025	51552
ii*	100.0	100.0	0.376	0.074	2331	2521
as*	100.0	100.0	0.295	0.127	1660	8818
tm*	100.0	100.0	0.244	0.024	724	602
aim*	99.8	51.5	0.259	0.144	20513	55263
jnh*	100.0	100.0	0.121	0.035	2989	5229
par8-*-c	100.0	100.0	0.609	0.085	55853	27080

7. Empirical results

In this section, we give an empirical evaluation of GLSSAT1 and GLSSAT2 on instances of the SAT problem from the DIMACS archive, and a set of weighted MAX-SAT problems.

7.1. The easier DIMACS benchmark problems

We ran GLSSAT1 on the easier problems in the DIMACS benchmark archive with the λ parameter set to 1, and the *smax* (the maximum number of consecutive sideways moves) set to 20. GLSSAT1 was run 10 times (this is to show that it is not just chance that GLSSAT1 finds a solution, rather than to study the effect of restarts) on each problem, allowing a maximum of 3,000,000 flips to solve each problem, starting from random starting points each time. We compared the results of GLSSAT1 against WalkSAT [33] (with Maxflips set to 3,000,000 and noise set to 0.5) to give the reader another local search algorithm to compare with. We give both the average number of flips to find a solution (for an implementation independent comparison) and also the average CPU time and success rate of each algorithm to find a solution. All experiments on the easier DIMACS problems, were performed on a Pentium II 300 Mhz PC running Windows NT 4.0, with 256MB of ram. GLSSAT1 was implemented in C++, and compiled using Microsoft Visual C++ 6.0.

Full results are listed in Appendix A.

7.2. The harder DIMACS problems

For the harder DIMACS problems, we used GLSSAT2 and allowed it 10,000,000 flips for the g*, f* and par8-* problems, and 150,000,000 for the par16-*-c problems to find a solution averaged over 10 runs from random starting points. The maximum number of consecutive sideways moves (*smax*) allowed was set to 20. For the graph coloring problems, we found that if we ran GLSSAT2 with *lambda* set to 1 we could not solve g125-17 and g250-29. By lowering *lambda* to 0.05 and thus making the search more intensive, so that the plateaus and basins in the search space are searched with more care, we found that we could find solutions to these problems. In a similar way to the graph coloring problems we found that we could not solve f1000 and f2000 using GLSSAT2 with the *lambda* parameter set to 1, so we tried lowering it to 0.05 and found again that we could find solutions for these problems. For the parity problems, we set lambda to 1.0. We also ran the standard WalkSAT on these problems to give the reader a reference point, with the maxflips paramter set to 10,000,000 on the g*, f* and par8-* problems and 150,000,000 flips for the par16-*-c problems. The noise parameter was tuned in 0.05 increments, between 0 and 1 (for the par8 problems, noise = 0.1, for the par16 we could not find a suitable noise setting which yielded solutions, for the g problems noise = 0.1 and for the f problems, noise = 0.55) for each problem group to maximise the number of successful runs. We allowed 10 tries to find a solution for each problem. All the experiments on the harder benchmark problems were run under Linux on identical Pentium III 450Mhz PCs with 128MB of ram, compiled using g++/gcc.

7.3. Weighted MAX-SAT benchmark problems

For these problems, we used smax = 2 and lambda = 1. We ran the algorithm 20 times with a maximum of 10000 flips allowed. We compare our results against the results of DLM, MaxWalkSAT [18] (with noise set to 0.3, which we found to give the best performance, out of 0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9 and 1.0) and GRASP over 20 runs, with a maximum of 10000 flips allowed for each, giving the best, worst and average deviation from the optimal solution for GLSSAT1 and the number of optimal solutions found for each problem. All the weighted MAX-SAT experiments were performed on a Pentium II 300 Mhz PC running Windows NT 4.0, with 256MB of ram. GLSSAT1 was implemented in C++, and compiled using Microsoft Visual C++ 6.0.

Table II. Results on harder DIMACS SAT instances

	Success rate / 10		Average CPU time		Average flips	
Problem	GLSSAT2	Walksat	GLSSAT2	Walksat	GLSSAT2	Walksat
par8-1	10	6	8.14	11.58	856626	4996298
par8-2	10	6	8.23	9.85	856772	4197080
par8-3	10	3	7.71	9.50	799245	4003199
par8-4	10	1	4.38	23.08	480596	9878630
par8-5	10	2	12.24	4.83	1219965	2030019
par16-1-c	10	0	281.81	-	25259873	-
par16-2-c	6	0	413.17	-	38414206	-
par16-3-c	10	0	522.28	-	48584078	-
par16-4-c	10	0	386.39	-	35490303	-
par16-5-c	9	0	441.03	-	41180575	-
g125-17	9	10	308.20	61.16	3500340	2511688
g125-18	10	10	1.23	1.07	9629	43033
g250-15	10	10	1.59	0.52	2524	4666
g250-29	10	10	905.24	119.81	3797486	1935538
f600	10	10	8.61	1.07	542008	217203
f1000	9	10	50.89	3.11	2877619	570843
f2000	6	10	113.97	8.53	4458712	1431990

8. Analysis

Overall, GLSSAT performs robustly in solving the SAT problem and solves problems in a comparable number of steps to WalkSAT, whilst on the weighted MAX-SAT problem, GLSSAT performs extremely well (performing better than any of the other local search algorithms, including WalkSAT, in terms of both solution quality and robustness).

However, we could not find solutions to the following problems using GLSSAT1 or GLSSAT2 in the DIMACS archive within 30 minutes: Hanoi4, Hanoi5, par16-*, par32-*, par32-*-c, f2000. Warner and van Maaren [47], report on a new approach to solving the parity problems, whereby they use linear programming to extract conjunctions of equivalencies from the original SAT formulation, so that they can solve a reduced problem, with a modified Davis-Putnam solver in a reasonable time.

In terms of average CPU time, GLSSAT1 was generally outperformed by WalkSAT on most of the easier SAT problems in the DIMACS archive, although it outperformed WalkSAT on the aim problem instances in both terms of average number of flips and success rate at

Table III. Weighted MAX-SAT Results (continued on next page)

	Over 20 runs/tries with a maximum of 10,000 flips for each algorithm						
		GLSSAT1			DLM	MaxWalkSAT	GRASP
Problem		Deviation from optimal solution					
name	No. Optimal	Worst	Average	Best	Best	Best	Best
Jnh1	20	0.00	0.00	0.00	0.00	0.00	188.00
Jnh4	8	50.00	19.50	0.00	41.00	41.00	215.00
Jnh5	15	131.00	29.35	0.00	0.00	114.00	254.00
Jnh6	18	11.00	1.10	0.00	0.00	0.00	11.00
Jnh7	20	0.00	0.00	0.00	0.00	0.00	0.00
Jnh8	19	121.00	6.05	0.00	0.00	0.00	578.00
Jnh9	7	192.00	33.80	0.00	7.00	0.00	514.00
Jnh10	20	0.00	0.00	0.00	0.00	0.00	275.00
Jnh11	5	90.00	11.95	0.00	0.00	13.00	111.00
Jnh12	20	0.00	0.00	0.00	0.00	0.00	188.00
Jnh13	19	18.00	0.90	0.00	0.00	0.00	283.00
Jnh14	18	87.00	7.40	0.00	0.00	0.00	314.00
Jnh15	16	94.00	12.50	0.00	0.00	0.00	359.00
Jnh16	8	5.00	3.00	0.00	0.00	0.00	68.00
Jnh17	20	0.00	0.00	0.00	0.00	0.00	118.00
Jnh18	9	98.00	20.65	0.00	0.00	0.00	423.00
Jnh19	15	107.00	21.15	0.00	0.00	0.00	436.00
Jnh201	20	0.00	0.00	0.00	0.00	0.00	0.00
Jnh202	19	79.00	3.95	0.00	0.00	0.00	187.00
Jnh203	18	64.00	6.40	0.00	0.00	0.00	310.00
Jnh205	20	0.00	0.00	0.00	0.00	0.00	14.00
Jnh207	17	9.00	1.35	0.00	0.00	0.00	137.00
Jnh208	18	146.00	13.80	0.00	0.00	0.00	172.00
Jnh209	20	0.00	0.00	0.00	0.00	0.00	207.00
Jnh210	20	0.00	0.00	0.00	0.00	0.00	0.00
Jnh211	19	154.00	7.70	0.00	0.00	0.00	240.00
Jnh212	15	11.00	2.75	0.00	0.00	0.00	195.00
Jnh214	19	11.00	0.55	0.00	0.00	0.00	462.00
Jnh215	17	96.00	9.40	0.00	0.00	0.00	292.00
Jnh216	13	118.00	27.50	0.00	0.00	0.00	197.00
Jnh217	20	0.00	0.00	0.00	0.00	0.00	6.00
Jnh218	20	0.00	0.00	0.00	0.00	0.00	139.00
Jnh219	19	163.00	8.15	0.00	0.00	103.00	436.00
Jnh220	19	43.00	2.15	0.00	0.00	0.00	185.00
Jnh301	18	12.00	1.20	0.00	0.00	0.00	184.00
Jnh302	20	0.00	0.00	0.00	338.00	129.00	211.00

Table III. Weighted MAX-SAT Results (continued from previous page)

	Over 20 runs/tries with a maximum of 10,000 flips for each algorithm						
		GLSSAT1			DLM	MaxWalkSAT	GRASP
Problem		Deviation from optimal solution					
name	No. Optimal	Worst	Average	Best	Best	Best	Best
Jnh303	16	152.00	28.60	0.00	143.00	80.00	259.00
Jnh304	19	273.00	13.65	0.00	0.00	31.00	319.00
Jnh305	9	258.00	91.90	0.00	194.00	196.00	609.00
Jnh306	20	0.00	0.00	0.00	0.00	0.00	180.00
Jnh307	12	63.00	15.70	0.00	0.00	0.00	155.00
Jnh308	13	156.00	34.40	0.00	0.00	156.00	502.00
Jnh309	19	49.00	2.45	0.00	0.00	0.00	229.00
Jnh310	16	38.00	7.60	0.00	0.00	0.00	109.00
Average	17	65.89	10.15	0.00	16.43	19.61	233.43

finding a solution. GLSSAT2 solves the parity learning problems, quite well, whereas WalkSAT has difficulty in solving the uncompressed par8-* instances and par16-*-c instances (in fact it didn't find any solutions to the par16-*-c instances). On the f and g problems, the tuned walksat performs much better than GLSSAT2, suggesting that walksat may be a more suitable algorithm for these problem instances.

On the weighted MAX-SAT problem, GLS's selective penalisation mechanism comes into it's own. Of the best of 20 runs of GLSSAT1 on the weighted MAX-SAT problem, GLSSAT1 finds the optimal solution for every problem, in fact in 85% (on average 17 out of 20) runs, GLSSAT1 finds the optimal solution. Furthermore, the average deviation, *averaged over 20* runs of GLSSAT1 is less than the average deviation of the *best of 20* runs for DLM and MaxWalkSAT. This shows that GLSSAT1 produces better quality solutions than both DLM and MaxWalkSAT. In the worst of 20 runs, GLSSAT1 outperforms GRASP for every problem. We believe that one of the reasons for GLSSAT1's superiority on these weighted MAX-SAT problems, is because we do not include the weights for each clause in the objective function (this is backed up by the fact that when we ran GLSSAT1 with clause weights in the objective function, it's performance was worse). Instead GLS uses selective penalisation based on the cost of features (i.e. weights of clauses) and the number of times they have already been penalised to select which features to try to remove. This means that clauses with large weights are penalised first, and thus discouraged earlier, so that

GLS guides the search in a direction which favours unsatisfied clauses with low costs.

9. Conclusion

Guided Local Search gives good results on most of the SAT instances in the DIMACS archive, comparable with those of WalkSAT in terms of the number of flips to find a solution and on some problems more reliable in finding solutions than WalkSAT (an algorithm considered to be state of the art). However, where GLSSAT comes into it's own is on the weighted MAX-SAT problem instances, where it comfortably outperforms DLM, MaxWalkSAT and GRASP, in both solution quality and robustness. We believe this is due to GLS's selective penalisation mechanism which is specifically designed for optimisation problems, whereas DLM and WalkSAT are not really designed for this purpose. This means the GLS does not need to include the weights for clauses in the objective function and thus the local search algorithm has less difficulty exploring the search space than DLM does, whilst GLS's selective penalisation mechanism takes the responsibility for removing high cost clauses from the solution.

In conclusion, we believe GLS has a place in the SAT problem, and that it is particularly effective for solving weighted MAX-SAT problems. GLS is a general meta-heuristic, with very few parameters and has been shown to be successful in solving a number of real life problems, as mentioned earlier. In general, we speculate that GLS will add value to many more local search algorithms, particular those dealing with soft constraints with varying costs.

Acknowledgements

Our thanks to John Ford, Dick Williams, Nathan Barnes, James Borrett, Christos Voudouris, Ian Gent, Toby Walsh and the anonymous referees for their useful comments and suggestions. Also, we would like thank Henry Kautz and Bart Selman for making a version of MaxWalkSAT available to us.

References

1. Borchers, B., Mitchell, J.E. and Joy, S.: 1997, 'A Branch-And-Cut Algorithm for MAX-SAT and weighted MAX-SAT', In *Satisfiability Problem: Theory*

and Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, volume 35, pages 519-536, American Mathematical Society, 1997.

2. Cha, B. and Iwama, K.: 1995, 'Performance Test of Local Search Algorithms Using New Types of Random CNF Formulas', In *Proceedings of IJCAI'95*, pages 304-310. Morgan Kaufmann Publishers, 1995.
3. Cha, B. and Iwama, K.: 1996, 'Adding New Clauses for Faster Local Search', In *Proceedings of AAAI'96*, pages 332-337. MIT Press, 1996.
4. Davenport, A.: 1997, 'Extensions and Evaluation of GENET in Constraint Satisfaction', *Ph.D. thesis*, Department of Computer Science, University of Essex, 1997.
5. Davenport A., Tsang E.P.K., Zhu K. and Wang, C.J.: 1994 'GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement', *In Proceedings AAAI 1994, p.325-330.*
6. Davis, M. and Putnam, H.: 1960, 'A computing procedure for quantification theory', *Journal of the ACM* **Vol. no. 7**, pp. 201-215
7. Frank, J., Cheeseman, P. and Stutz, J.: 1997, 'When Gravity Fails: Local Search Topology', *Journal of Artificial Intelligence Research*, 7:249-281, 1997.
8. Frank, J.: 1996, 'Weighting for Godot: Learning Heuristics for GSAT', *Proceedings of AAAI'96*, pages 338-343, MIT Press, 1996.
9. Frank, J.: 1997, 'Learning Short-term Clause Weights for GSAT', *Proceedings of IJCAI'97*, pages 384-389, Morgan Kaufmann Publishers, 1997.
10. Garey, M.R. and Johnson, D.S.: 1979, 'Computers and intractibility', W.H.Freeman and Company.
11. Gent, I. and Walsh, T.: 1995, 'Unsatisfied Variables in Local Search', *In 'Hybrid Problems, Hybrid Solutions', ed. J. Hallam, IOS Press, Amsterdam, 1995, pp 73-85. (Proceedings of AISB-95)*
12. Gent, I. and Walsh, T.: 1993, 'Towards an Understanding of Hill-climbing Procedures for SAT', it In Proceedings of AAAI-93.
13. Ginsberg, M. and McAllester, D.: 1994, 'GSAT and Dynamic Backtracking', *In Fourth Conference on Principles of Knowledge Representation and Reasoning (KR-94), May, 1994, pp 226-237*
14. Gu, J.: 1993 Local search for Satisfiability (SAT) Problem. *In IEEE Transactions on Systems, Man and Cybernetics, Vol. 23, No. 4, July/August 1993.*
15. Gu, J.: 1994 Global Optimization for Satisfiability (SAT) Problem. *In IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 3, June 1994, pages 361-381.*
16. Hampson, S. and Kibler, D.: 1993 'Large plateaus and plateau search in Boolean Satisfiability problems: When to give up searching and start again', *In DIMACS Series, Cliques, Coloring and SAT, Vol. 26, pp437-456.*
17. Hoos, H.H. and Stutzle, T.: 1998 'Evaluating Las Vegas Algorithms - Pitfalls and Remedies', *In Proceedings of UAI-98*, pages 238-245, Morgan Kaufmann.
18. Jiang Y., Kautz H., and Selman B.: 1995 Solving Problems with Hard and Soft Constraints Using a Stochastic Algorithm for MAX-SAT. *1st International Joint Workshop on Artificial Intelligence and Operations Research.*
19. Kautz, H., McAllester, D. and Selman, B.: 1997 Exploiting Variable Dependency in Local Search. DRAFT. *Abstract appears in Abstracts of the Poster Sessions of IJCAI-97, Nagoya, Japan, 1997.*

20. Kilby, P., Prosser, P. and Shaw, P.: 1997 Guided Local Search for the Vehicle routing Problem, *In Proceedings of the 2nd International Conference on Metaheuristics, July 1997.*
21. Lau, T.L.: 1999 Guided Genetic Algorithm, PhD Thesis, Department of Computer Science, University of Essex 1999.
22. Lau, T.L. and Tsang, E. P. K.: 1998a Solving the generalized assignment problem with the guided genetic algorithm. *In Proceedings of 10th IEEE International Conference on Tools for Artificial Intelligence, 1998.*
23. Lau, T.L. and Tsang, E. P. K.: 1998b Guided Genetic algorithm and its application to the large processor configuration problem. *In Proceedings of 10th IEEE International Conference on Tools for Artificial Intelligence, 1998.*
24. Li, C.M. and Ambulagon: 1997 'Look-Ahead Versus Look-Back for Satisfiability Problems. *In Proceedings of CP'97*, pages 341-355. Spring Verlag, 1997.
25. McAllester, D., Selman, B. and Kautz, H.: 1997 Evidence for Invariants in Local Search. *In Proceedings AAAI-97, Providence, RI, 1997.*
26. Minton, S., Johnson M.D., Philips A.B. and Laird P.: 1992 Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *In Artificial Intelligence 58 (1992), pp161-205.*
27. Morris, P.: 1993 The Breakout method for escaping from local minima. *In Proceedings of AAAI-93, pp40-45. AAAI Press/MIT Press.*
28. Resende, M.G.C. and Feo, T.A.: 1996 A GRASP for Satisfiability *In "Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge," David S. Johnson and Michael A. Trick, Eds., DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 26, pp. 499-520, American Mathematical Society, 1996.*
29. Resende, M.G.C., Pitsoulis, L.S. and Pardalos, P.M.: 1997 Approximate Solution of Weighted MAX-SAT Problems using GRASP. *In DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 393-405.*
30. Richards, T. and Richards, B.: 1998 Non-systematic search and learning: an empirical study. *In Lecture Notes in Computer Science 1520, Maher, M. and Puget J-F. (eds.). (Proceedings of 4th International Conference on Principles and Practice of Constraint Programming), Springer Verlag, pp370-384.*
31. Selman, B. and Kautz, H.: 1993 An Empirical Study of Greedy Local Search for Satisfiability Testing. *In Proceedings AAAI-93, 1993.*
32. Selman, B. and Kautz, H.: 1993 Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. *In Proceedings IJCAI-93, 1993.*
33. Selman, B., Kautz, H. and Cohen, B.: 1994 Noise Strategies for Improving Local Search. *In Proceedings AAAI-94, 1994.*
34. Selman, B., Kautz H. and McAllester, D.: 1997 Ten Challenges in Propositional Reasoning and Search. *In Proceedings of the Fifteenth International Conference on Artificial Intelligence (IJCAI-97), NAGOYA, Aichi, Japan, 1997.*
35. Selman, B., Levesque, H. and Mitchell, D.: 1992 A New Method for Solving Hard Satisfiability Problems. *In Proceedings AAAI-92, 1992.*
36. Shang, Y.: 1997 Global Search Methods for Solving Nonlinear Optimization Problems, *Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, August 1997.*
37. Shang, Y. and Wah, B. W.: 1998 Improving the Performance of Discrete Lagrange-Multiplier Search for Solving Hard SAT Problems. *In Proceedings 10th International Conference on Tools with Artificial Intelligence, IEEE, (accepted to appear) Nov. 1998.*

38. Shang, Y. and Wah, B.W.: 1998 A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems. *In Journal of Global Optimization, Kluwer Academic Publishers, vol. 12, no. 1, Jan. 1998, pp. 61-99.*
39. Tsang, E.: 1993 Foundations of Constraint Satisfaction. *Academic Press, 1993.*
40. Tsang E. and Voudouris C.: 1997 Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. *In Operations Research Letters 20 (1997), 119-127.*
41. Voudouris, C.: 1997 Guided Local Search for Combinatorial Optimisation Problems, *Ph.D. thesis. Department of Computer Science, University of Essex, 1997.*
42. Voudouris, C.: 1998 Guided Local Search - an illustrative example in function optimisation. *In BT Technology Journal, Vol. 16, No. 3, July 1998, pp46-50.*
43. Voudouris, C. and Tsang, E.P.K.: 1998a Guided local search and its application to the traveling salesman problem. *In European Journal of Operational Resaerch, Vol.113, Issue 2, November 1998, pp469-499.*
44. Voudouris, C. and Tsang, E.P.K.: 1998b Solving the Radio Link Frequency Assignment Problem using Guided Local Search. *In Proceedings, NATO Symposium on Radio Length Frequency Assignment, Sharing and Conservation Systems (Aerospace), Aalborg, Denmark, October 1998.*
45. Wah, B.W. and Shang, Y.: 1997 Discrete Lagrangian-Based Search for Solving MAX-SAT Problems. *In 15th International Joint Conference on Artificial Intelligence, 1997, pp378-383.*
46. Wang, C.J. and Tsang, E.P.K.:1991 Solving constraint satisfaction problems using neural-networks. *Proceedings, IEE Second International Conference on Artificial Neural Networks, 1991, p.295-299*
47. Warners, J.P. and van Maaren, H.: 1998 A Two Phase Algorithm for Solving a Class of Hard Satisfiability Problems. *Report SEN-R9802 (CWI). To appear in Operations Research Letters 23 (1999), pp. 81-88.*
48. Warners, J.P. and van Maaren, H.: 1997 Satisfiability problems with balanced polynomial representation. *Report of the Faculty of Technical Mathematics and Informatics 97-47, Deft University of Technology, 1997.*
49. Wu, Z.: 1998 The Discrete Lagrangian Theory and its Application to Solve Nonlinear Discrete Constrained Optimization Problems, *M.Sc. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1998.*

10. Appendix A: Full results on DIMACS benchmark problems

Table IV. Results on Technology mapping problems

Problem	Success rate / 10		Mean CPU time		Mean flips	
name	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
tm1-yes	10	10	0.47	0.05	1287	1074
tm2-yes	10	10	0.02	0.00	161	129

Table V. Results on Circuit diagnosis problems

Problem	Success rate / 10		Mean CPU time		Mean flips	
name	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
ssa7552-038	10	10	0.73	0.32	70052	118122
ssa7552-158	10	10	0.88	0.09	102904	32339
ssa7552-159	10	10	0.65	0.06	74412	24185
ssa7552-160	10	10	0.36	0.08	36732	31563

Table VI. Results on the Asynchronous circuit synthesis problems

Problem	Success rate / 10		Mean CPU time		Mean flips	
name	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
as2-yes	10	10	0.01	0.00	147	250
as3-yes	10	10	0.01	0.00	190	240
as4-yes	10	10	0.12	0.03	1078	2271
as5-yes	10	10	2.75	1.46	13014	102867
as6-yes	10	10	0.06	0.02	660	1419
as7-yes	10	10	0.53	0.09	2061	3705
as8-yes	10	10	0.04	0.00	696	350
as10-yes	10	10	0.10	0.01	969	643
as11-yes	10	10	0.08	0.01	1268	625
as12-yes	10	10	0.01	0.00	164	232
as13-yes	10	10	0.04	0.01	488	798
as14-yes	10	10	0.01	0.00	142	82
as15-yes	10	10	0.09	0.02	706	1157

Table VII. Results on Circuit synthesis problems

Problem	Success rate / 10		Mean CPU time		Mean flips	
name	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
ii8a1	10	10	0.00	0.00	92	42
ii8a2	10	10	0.01	0.00	171	109
ii8a3	10	10	0.01	0.00	219	203
ii8a4	10	10	0.03	0.00	499	364
ii8b1	10	10	0.02	0.00	226	132
ii8b2	10	10	0.06	0.01	933	706
ii8b3	10	10	0.10	0.02	1197	1645
ii8b4	10	10	0.15	0.03	1634	2641
ii8c1	10	10	0.03	0.00	304	213
ii8e2	10	10	0.08	0.01	692	402
ii8c2	10	10	0.09	0.01	735	422
ii8d1	10	10	0.04	0.00	505	256
ii8d2	10	10	0.08	0.01	651	436
ii8e1	10	10	0.03	0.00	320	240
ii16c1	10	10	0.28	0.02	1403	781
ii16c2	10	10	0.34	0.61	3357	23108
ii16d1	10	10	0.28	0.03	1482	1452
ii16d2	10	10	0.24	0.52	1974	20238
ii16e1	10	10	0.29	0.02	1118	1005
ii16e2	10	10	0.16	0.03	871	1479
ii16a2	10	10	0.37	0.16	2358	6410
ii16b1	10	10	0.41	0.05	2053	1711
ii16b2	10	10	0.28	0.32	2217	11069
ii16a1	10	10	0.31	0.02	1777	897

Table VII. Results on Circuit synthesis problems (continued)

Problem name	Success rate / 10		Mean CPU time		Mean flips	
	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
ii32a1	10	10	1.05	0.03	5543	1246
ii32b1	10	10	0.15	0.00	3931	252
ii32b2	10	10	0.34	0.01	4865	627
ii32b3	10	10	0.48	0.02	2560	974
ii32b4	10	10	0.33	0.02	1582	952
ii32c1	10	10	0.05	0.00	904	136
ii32c2	10	10	0.11	0.00	1216	299
ii32c3	10	10	0.80	0.01	8385	394
ii32c4	10	10	2.23	0.78	4315	11021
ii32d1	10	10	0.22	0.00	4832	480
ii32d2	10	10	0.63	0.02	7039	1519
ii32d3	10	10	3.41	0.11	13187	3199
ii32e2	10	10	0.21	0.01	2319	389
ii32e3	10	10	0.23	0.01	1483	685
ii32e4	10	10	0.57	0.03	2590	1046
ii32e5	10	10	0.86	0.13	2901	4057
ii32e1	10	10	0.05	0.00	1124	104

Table VIII. Results on Artificial 3-SAT problems

Problem name	Success rate / 10		Mean CPU time		Mean flips	
	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
aim-50-2_0-yes1-3	10	10	0.01	0.17	1181	81560
aim-50-1_6-yes1-1	10	1	0.02	0.03	1918	12359
aim-50-1_6-yes1-2	10	0	0.02	-	1636	-
aim-50-1_6-yes1-3	10	7	0.02	2.18	1709	1110424
aim-50-1_6-yes1-4	10	0	0.01	-	1444	-
aim-50-2_0-yes1-1	10	0	0.01	-	1268	-
aim-50-2_0-yes1-2	10	9	0.01	0.04	889	19688
aim-50-2_0-yes1-4	10	10	0.01	0.37	817	186927
aim-50-3_4-yes1-1	10	10	0.03	0.01	2146	3632
aim-50-3_4-yes1-2	10	10	0.01	0.00	517	1392
aim-50-3_4-yes1-3	10	10	0.01	0.00	342	882
aim-50-3_4-yes1-4	10	9	0.00	0.00	137	749
aim-50-6_0-yes1-1	10	10	0.01	0.00	86	778
aim-50-6_0-yes1-2	10	10	0.00	0.00	86	325
aim-50-6_0-yes1-3	10	10	0.01	0.00	162	543
aim-50-6_0-yes1-4	10	10	0.01	0.00	138	513
aim-100-1_6-yes1-2	10	0	0.08	-	8581	-
aim-100-1_6-yes1-3	10	0	0.10	-	11225	-
aim-100-1_6-yes1-4	10	0	0.09	-	10246	-
aim-100-2_0-yes1-1	10	0	0.09	-	8436	-
aim-100-2_0-yes1-2	10	0	0.10	-	9270	-
aim-100-2_0-yes1-4	10	0	0.08	-	7780	-
aim-100-3_4-yes1-1	10	10	0.13	0.04	9064	11587
aim-100-3_4-yes1-2	10	10	0.02	0.02	1459	6734
aim-100-3_4-yes1-3	10	0	0.03	-	1750	-
aim-100-3_4-yes1-4	10	10	0.02	0.01	1471	2624
aim-100-6_0-yes1-1	10	10	0.01	0.00	197	782
aim-100-6_0-yes1-2	10	10	0.01	0.01	330	1243
aim-100-6_0-yes1-3	10	10	0.01	0.01	294	1866
aim-100-2_0-yes1-3	10	0	0.09	-	7957	-
aim-100-6_0-yes1-4	10	6	0.01	0.01	309	824
aim-100-1_6-yes1-1	10	0	0.09	-	9206	-

Table VIII. Results on Artificial 3-SAT problems (continued)

Problem name	Success rate / 10		Mean CPU time		Mean flips	
	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
aim-200-6_0-yes1-4	10	10	0.03	0.02	715	2747
aim-200-1_6-yes1-1	10	0	0.52	-	50884	-
aim-200-1_6-yes1-2	10	0	0.58	-	57444	-
aim-200-1_6-yes1-3	10	0	0.54	-	53104	-
aim-200-1_6-yes1-4	10	0	0.56	-	54488	-
aim-200-2_0-yes1-1	10	0	0.95	-	86069	-
aim-200-2_0-yes1-2	10	0	0.67	-	60978	-
aim-200-2_0-yes1-3	10	0	0.69	-	62791	-
aim-200-2_0-yes1-4	10	0	0.80	-	72276	-
aim-200-3_4-yes1-1	10	10	3.27	0.20	218214	55052
aim-200-3_4-yes1-2	10	10	0.13	0.13	7639	35660
aim-200-3_4-yes1-3	9	5	2.20	3.27	138974	1008856
aim-200-3_4-yes1-4	10	9	0.29	0.31	17014	91361
aim-200-6_0-yes1-1	10	5	0.03	0.02	786	2932
aim-200-6_0-yes1-2	10	10	0.04	0.04	768	6042
aim-200-6_0-yes1-3	10	6	0.02	0.03	458	4518

Table IX. Results on the Random SAT problems

Problem	Success rate / 10		Mean CPU time		Mean flips	
name	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
jnh1	10	10	0.04	0.01	741	1934
jnh7	10	10	0.02	0.00	403	438
jnh12	10	10	0.05	0.04	951	5097
jnh17	10	10	0.04	0.01	762	810
jnh201	10	10	0.02	0.00	233	390
jnh204	10	10	0.04	0.02	916	2180
jnh205	10	10	0.04	0.01	904	1076
jnh207	10	10	0.19	0.09	4954	13996
jnh209	10	10	0.06	0.04	1285	5919
jnh210	10	10	0.01	0.01	205	791
jnh212	10	10	0.82	0.10	21652	14763
jnh213	10	10	0.02	0.02	421	3104
jnh217	10	10	0.02	0.01	426	605
jnh218	10	10	0.02	0.01	407	903
jnh220	10	10	0.33	0.16	8625	24381
jnh301	10	10	0.22	0.05	4943	7276

Table X. Results on easier Parity learning problems

Problem	Success rate / 10		Mean CPU time		Mean flips	
name	GLSSAT	WalkSAT	GLSSAT	WalkSAT	GLSSAT	WalkSAT
par8-1-c	10	10	0.07	0.03	6390	10320
par8-2-c	10	10	0.05	0.03	4358	9049
par8-3-c	10	10	2.32	0.04	213274	12956
par8-4-c	10	10	0.18	0.21	15925	67803
par8-5-c	10	10	0.43	0.12	39316	35272