Iterated Robust Tabu Search for MAX-SAT

Kevin Smyth[1], Holger H. Hoos[1⋆], and Thomas Stützle[2]

[1] Department of Computer Science, University of British Columbia,
Vancouver, B.C., V6T 1Z4, Canada
{hoos,ksmyth}@cs.ubc.ca
http://www.cs.ubc.ca/labs/beta
[2] Fachbereich Informatik, Technische Universität Darmstadt,
Alexanderstr. 10, D-64289 Darmstadt, Germany
stuetzle@informatik.tu-darmstadt.de

Abstract. MAX-SAT, the optimisation variant of the satisfiability problem in propositional logic, is an important and widely studied combinatorial optimisation problem with applications in AI and other areas of computing science. In this paper, we present a new stochastic local search (SLS) algorithm for MAX-SAT that combines Iterated Local Search and Tabu Search, two well-known SLS methods that have been successfully applied to many other combinatorial optimisation problems. The performance of our new algorithm exceeds that of current state-of-the-art MAX-SAT algorithms on various widely studied classes of unweighted and weighted MAX-SAT instances, particularly for Random-3-SAT instances with high variance clause weight distributions. We also report promising results for various classes of structured MAX-SAT instances.

1 Introduction and Background

The satisfiability problem in propositional logic (SAT) is the task to decide whether a given propositional formula has a model. More formally, given a set of m clauses $\{C_1, \ldots, C_m\}$ involving n Boolean variables $x_1, \ldots, x_n$ the SAT problem is to decide whether an assignment of values to variables exists such that all clauses are simultaneously satisfied. This problem plays a prominent role in various areas of computer science, mathematical logic and artificial intelligence, but also in many applications [7, 13, 1].

MAX-SAT is the optimisation variant of SAT and can be seen as a generalisation of the SAT problem: Given a propositional formula in conjunctive normal form (CNF), the MAX-SAT problem then is to find a variable assignment that maximises the number of satisfied clauses. In *weighted MAX-SAT*, each clause C_i has an associated weight w_i and the goal becomes to maximise the total weight of the satisfied clauses. The decision variants of both SAT and MAX-SAT are $\mathcal{NP}$–complete [5]. Furthermore, it is known that optimal solutions to MAX-SAT are hard to approximate; for MAX-3-SAT (unweighted MAX-SAT with 3 literals per clause), *e.g.*, there exists no polynomial-time approximation algorithm with a (worst-case) approximation ratio

⋆ To whom correspondence should be addressed.

lower than $8/7 \approx 1.1429$. It is worth noting that approximation algorithms for MAX-SAT can be empirically shown to achieve much better solution qualities for many types of MAX-SAT instances; however, their performance is usually substantially inferior to that of state-of-the-art stochastic local search (SLS) algorithms for MAX-SAT (see, *e.g.*, [8]).

Many SLS methods have been applied to MAX-SAT leading to a large number of algorithms for unweighted and weighted MAX-SAT. These include algorithms originally proposed for SAT, which can be applied to unweighted MAX-SAT in a straightforward way by keeping track of the best solution found so far in the search process. It is not clear that SLS algorithms that are known to perform well on SAT can be expected to show equally strong performance on unweighted MAX-SAT and some empirical evidence suggests that this is generally not the case. Therefore, many SLS algorithms were directly developed for unweighted and, in particular, weighted MAX-SAT or extended from existing SLS algorithms for SAT in various ways.

The currently best performing SLS algorithms for unweighted and weighted MAX-SAT fall into three categories: Tabu Search algorithms, Dynamic Local Search algorithms, and Iterated Local Search. Very good performance was reported for Reactive Tabu Search (H-RTS), a tabu search that dynamically adjusts the tabu tenure, on unweighted MAX-SAT instances [2]. High performing Dynamic Local Search algorithms include DLM by Wah and Shang [19], a later extension called DLM-99-SAT [21], and Guided Local Search (GLS) [15]. Computational results suggest that GLS is currently the top performing SLS algorithm for specific classes of weighted MAX-SAT instances, outperforming DLM and WalkSAT extensions to weighted MAX-SAT [11]. Also noteworthy is the recent Iterated Local Search by Yagiura and Ibaraki (ILS-YI) [23] that uses a local search algorithm based on 2- and 3-flip neighbourhoods. Particularly for MAX-SAT-encoded minimum-cost graph colouring and set covering instances, as well as for a big, MAX-SAT-encoded real-world time-tabling instance, the 2-flip variant of ILS-YI performs better than other versions of ILS-YI and a tabu search algorithm implemented by Yagiura and Ibaraki.

In this paper, we propose a new Iterated Local Search (ILS) algorithm and experimentally compare its performance to current state-of-the-art algorithms. The key idea behind ILS is to alternate between local searches and so-called perturbation phases which are designed to take the search away from the local optimum reached by the subsidiary local search procedure. Our new ILS algorithm, Iterated Robust Tabu Search (IRoTS), uses a Robust Tabu Search (RoTS) algorithm for both the subsidiary local search and perturbation phases. RoTS is a particular Tabu Search algorithm, originally applied to the Quadratic Assignment Problem [20], which we adapted to MAX-SAT.

Our empirical evaluation of IRoTS indicates that on a range of well-known benchmark instances for weighted and unweighted MAX-SAT this new algorithm performs significantly better than state-of-the-art MAX-SAT algorithms, such as GLS [15]. This is particularly the case for weighted MAX-SAT instances with highly variable clause weight distributions, as well as for highly overconstrained unweighted MAX-SAT in-

procedure *Iterated Local Search*
  $s_0 = GenerateInitialSolution$
  $s^* = LocalSearch(s_0)$
  repeat
    $s' = Perturbation(s^*)$
    $s^{*\prime} = LocalSearch(s')$
    $s^* = AcceptanceCriterion(s^*, s^{*\prime})$
  until termination condition met
end procedure

Fig. 1. Algorithm outline of ILS.

stance; in both cases, IRoTS reaches quasi-optimal solutions[3] up to an order of magnitude faster than GLS. IRoTS also performs significantly better than many of the state-of-the-art algorithms on various classes of structured MAX-SAT instances, although on these instances it typically does not reach the performance of GLS.

A detailed analysis of the behaviour of IRoTS on individual problem instances shows that its performance over multiple runs as well as over multiple instances from the same test-set is typically less variable than that of GLS, which indicates that our Iterative Robust Tabu Search algorithm escapes more effectively from local optima than the Dynamic Local Search scheme underlying GLS.

The remainder of this paper is structured as follows. In the next section we introduce Robust Tabu Search for weighted MAX-SAT and our new Iterated Local Search algorithm, Iterated Robust Tabu Search. In Section 3, we describe the experimental protocol and benchmark instances used for our empirical analysis, whose results are presented and discussed in Sections 4 and 5. Finally, in Section 6 we draw some conclusions and briefly discuss several directions for future work.

2 ILS for MAX-SAT

In this section we describe our Iterated Local Search (ILS) implementation for MAX-SAT. ILS is a class of algorithms that essentially perform a biased random walk in the space of the local optima encountered by an underlying local search algorithm [14]. This walk is obtained by iteratively perturbing a locally optimal solution s^*, then applying local search to obtain a new locally optimal solution $s^{*\prime}$, and finally using an acceptance criterion to decide from which of the two solutions $s^*, s^{*\prime}$ to continue the search. An algorithm outline of ILS is given in Figure 1.

Our ILS algorithm for weighted and unweighted MAX-SAT is strongly based on an adaptation of Robust Tabu Search (RoTS) [20] to MAX-SAT. RoTS is used in the local search phase and also in the perturbation phase. Since RoTS is actually itself a

[3] Many of the test sets we experiment with in this study are intractable for complete solvers, so we are forced to empirically estimate the optimal solutions for some instances. Details of the protocol used for this estimation are given in Section 3.

high-performing SLS algorithm for MAX-SAT, we first give some details on the RoTS algorithm, before describing how it is used in our ILS algorithm.

In each search step, the RoTS algorithm for MAX-SAT flips a non-tabu variables that achieves a maximal improvement in the total weight of the unsatisfied clauses (the size of this improvement is also called *score*) and declares it tabu for the next *tl* steps. The parameter tl is called the tabu tenure. An exception to this "tabu" rule is made if a more recently flipped variable achieves an improvement over the best solution seen so far (this mechanism is called aspiration). Furthermore, whenever a variable has not been flipped within a certain number of search steps (in our implementation: $10n$), it is forced to be flipped. This implements a form of long-term memory and helps prevent stagnation of the search process. Finally, instead of using a fixed tabu tenure, every n iterations the parameter tl is randomly chosen from an interval $[t_{min}, t_{max}]$ according to a uniform distribution. The tabu status of variables is determined by comparing the number of search steps that have been performed since the most recent flip of a given variable with the current tabu tenure; hence, changes in *tl* immediately affect the tabu status and tenure of all variables.

Our ILS algorithm, called IRoTS in the following, is initialised by setting each variable independently to true or false with equal probability (the same random initialisation is used by most other SLS algorithms for SAT and MAX-SAT). As previously stated, its subsidiary local search and perturbation procedures are both based on the RoTS algorithm described above. Each local search phase executes RoTS steps until no improvement in the incumbent solution has been achieved for *escape_threshold* iterations. This parameter is set by default to $n^2/4$, which was determined to give robust performance over a wide range of test sets. The default tabu tenure for the local search phase was set to $n/10 + 4$, which robustly achieves good performance on many of our test sets. The perturbation phase consists of a fixed number of RoTS search steps (default value: $9n/10$) with tabu tenure values that are substantially higher than the ones used in the local search phase (default tabu tenure for the perturbation is $n/2$). At the beginning of each local search and perturbation phase, all variables are declared non-tabu, irrespectively of their previous tabu status. If applying perturbation and subsequent local search to a candidate solution s results in a candidate solution s' that is better than the best candidate solution accepted since the search was initialised, the search is continued from s'. If s and s' have the same solution quality, one of them is chosen uniformly at random. In all other cases, the worse of the two candidate solutions s and s' is chosen with probability 0.1, and the better one otherwise.

The default parameter settings for IRoTS used in this study were determined in preliminary experiments, in which we also observed that IRoTS typically achieves significant performance improvements over RoTS. These default settings were used in all experiments reported here, with the exception of the experiments on the structured instances. Details on our experiments (including parameter settings) are given in Sections 4 and 5.

Throughout this paper, we compare IRoTS to two variants of Guided Local Search (GLS) as well as the ILS algorithm by Yagiura and Ibaraki (ILS-YI). GLS [15] iteratively modifies clause penalties to help the local search escape from local optima. GLS is based on HSAT [6] as the underlying local search method; in each search step, HSAT

flips a variable with maximal score, breaking ties in favour of the variable which was flipped longest ago. When trapped in a local optimum, GLS modifies a penalty vector consisting of penalty values clp_i for each clause C_i in the given CNF formula. In GLS, clause weights in weighted MAX-SAT instances are only considered when computing the utility value of a clause, defined as $util(a, C_i) = w_i/(1 + clp_i)$ if clause i is unsatisfied under the current variable assignment a and zero otherwise. At each iteration of GLS, only the penalties of clauses with maximum utility are increased. The clause penalties are used when computing the variables' score when evaluating flips. In the GLS2 variant, all penalty values are regularly decayed, while in the basic GLS penalties never decrease. The reference implementations for GLS and GLS2 used for our experiments were kindly provided by Patrick Mills.

Yagiura and Ibaraki proposed and studied a simple ILS algorithm for MAX-SAT, ILS-YI, which initialises the search at a randomly chosen assignment, uses a subsidiary iterative first improvement search procedure, and a perturbation phase that consists of a fixed number of (undirected) random walk steps; the acceptance criterion always selects the better of the two given candidate solutions [22, 23]. The ILS-YI algorithm performs particularly well when using a 2- or 3-flip neighbourhood in the subsidiary local search procedure. The key to an efficient implementation of ILS-YI with this 2- and 3-flip local search procedure lies in an efficient caching scheme for evaluating moves in the respective larger neighbourhoods. An implementation of ILS-YI is publicly available at `http://www-or.amp.i.kyoto-u.ac.jp/members/yagiura/msat-codes/`.

3 Experiment Design

For the computational experiments conducted in this study, we compare the performance of the algorithms described in Section 2 on both random and structured instances. All experiments were run on a 1Ghz Pentium III with 1 GB RAM, 256KB L2 cache, running Linux RedHat 7.3 and using `gcc-3.2`. We first compared the performance of the algorithms on the `wjnh` and the `bor` test sets, which have been previously proposed and studied in the literature [18, 3, 12]. The `wjnh` test set consists of 44 randomly generated instances. The clause lengths vary in size, and the clause weights are randomly and uniformly chosen integers from the interval [1 . . . 1000]. Several of the instances are satisfiable — a clear distinction is made between the satisfiable and unsatisfiable instances when reporting results for this test set. The `bor` test set consists of both weighted and unweighted Random 2- and 3-SAT instances, resulting in four classes of instances $\{weighted, unweighted\} \times \{2SAT, 3SAT\}$. Each of these classes is divided into three test sets consisting of 50, 100, and 150 variable instances for a total of 12 test sets. The number of instances in each test set is relatively small, ranging from 2 to 9, and each instance has a different clause to variable ratio (all instances are significantly overconstrained). The clause weights in the weighted test sets are randomly and uniformly chosen integers from the interval [1 . . . 10].

In order to be able to perform more systematic and detailed empirical evaluations, we generated twelve new test sets of benchmark instances with 100 instances each. Several of these test sets are unweighted, and sampled from the Uniform Random 3-SAT

Name	n	m
rnd50-250u	50	250
rnd100-500u	100	500
rnd100-700u	100	700
rnd100-850u	100	850
rnd100-1000u	100	1000
rnd200-1000u	200	1000

Name	n	m	μ	σ
rnd50-w50	50	250	250	50
rnd50-w250	50	250	250	250
rnd100-w100	100	500	500	100
rnd100-w500	100	500	500	500
rnd200-w200	200	1000	1000	200
rnd200-w1000	200	1000	1000	1000

Table 1. Test-sets of unweighted MAX-SAT instances with varying clause to variable ratios (left), and weighted MAX-SAT instances with different variance σ^2 of clause weight distributions (right) used within this study.

distribution [16] for various number of variables and clauses, corresponding to the over-constrained region of Uniform Random-3-SAT. The remaining test sets were obtained from this set by adding integer clause weights that were randomly generated according to discretised, truncated normal distributions. In all cases, the mean of the clause weight distribution was chosen as $\mu = 5n$, where n is the number of variables, and the distribution was symmetrically truncated such that all clause weights are restricted to lie in the interval $[1 \ldots 2\mu - 1]$. Symmetric truncation guarantees that the actual mean is close to μ. Within this class of distributions, standard deviations σ of n and $5n$ were used for generating our test-sets. The resulting test-sets are summarised in Table 1.

Additionally, we performed experiments on three test sets of structured MAX-SAT instances; these were obtained by encoding Minimum Cost Graph Colouring and Set Covering Problems, as well as and Level Graph Crossing Minimisation Problems into weighted MAX-SAT [23, 4]. We denote these test-sets as `YI-GCP`, `YI-SCP`, and `LGCMP`, respectively.

To evaluate the relative performance of IRoTS, ILS-YI, GLS, and GLS2, we measured the distribution of the CPU time (and number of search steps) required by each algorithm for reaching a certain (typically optimal) solution quality on each given instance. These run-time distributions (RTDs) [10] were measured by running each algorithm 100 times on each problem instance until the specified solution quality was reached or exceeded. We refer to the median of the RTDs thus obtained as the *search cost* (sc) for the given algorithm and instance.

With the exception of the experiments on structured MAX-SAT instances, we generally measured RTDs for reaching provably optimal or best-known solution qualities. Where possible, we used Borcher and Furman's complete solvers `maxsat` and `wmaxsat` [3] to determine the optimal solution quality for each instance. (These are the best-performing complete MAX-SAT solvers we are aware of.) However, since the larger instances become intractable for these solvers, we estimated the quality of the optimal solutions by using an "iterative deepening" scheme for IRoTS. IRoTS is run for *base_cutoff* $= 10^6$ steps. Whenever an assignment is found that is better than any previously found assignment, the cutoff is set to the maximum of *base_cutoff* and ten times the number of search steps taken up to that point. This "iterative deepening" scheme is repeated 10 times per instance, and the best solution quality found over all 10 runs was

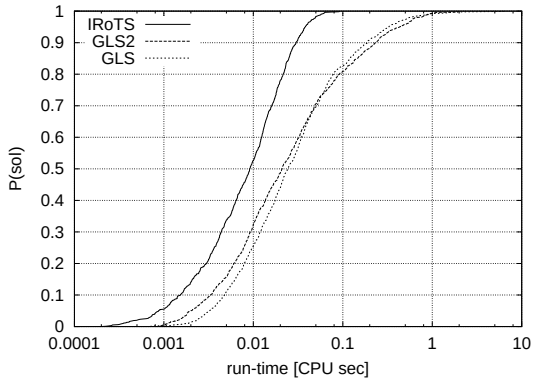

Fig. 2. RTDs for IRoTS, GLS, and GLS2 on the instance from the `rnd100-w100` test set with the median sc (in terms of run-time for GLS).

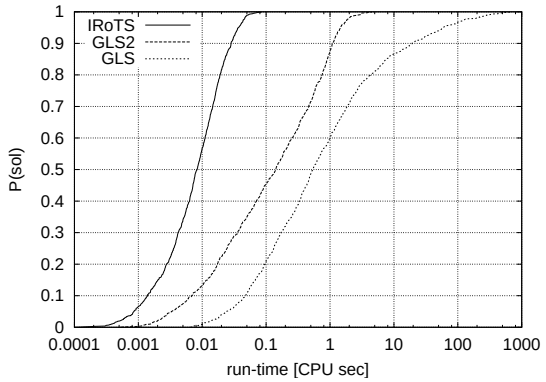

Fig. 3. RTDs for IRoTS, GLS, and GLS2 on the instance from the `rnd100-w100` test set with the highest sc (in terms of run-time for GLS).

reported as optimal. We verified the solution qualities reported as optimal following this protocol against the true optimal solution qualities for the test sets for which it was still possible to run the complete solver, and in all cases IRoTS had found the true optimal solution. We also verified the optimal solutions for a small, randomly selected set of larger instances (this required running the complete solver for multiple CPU days). In the remainder of this paper we treat the solutions returned by this protocol as optimal.

4 Results for Random Instances

In this section, we present our results for the randomly generated instances described in Section 3. Figures 2 and 3 show the run-time distributions of IRoTS, GLS, and GLS2 on the instances from the test set `rnd100-w100` with median and max sc respectively.

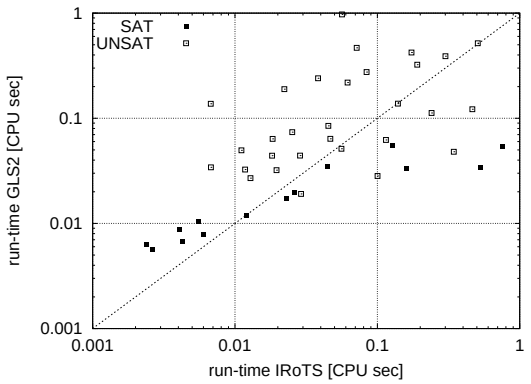

Fig. 4. Performance of IRoTS vs. GLS2 on the `wjnh` test set, measuring median CPU time for finding optimal solutions.

It is clear from the fat right tails of the RTDs for GLS and GLS2 that both of these algorithms suffer from stagnation behaviour on these instances. This effect is most apparent in the hard instance (Figure 3), where we see a very pronounced right tail for GLS. The RTDs for IRoTS are approximately exponential.

This trend was present for all of the random test sets, and results in GLS2 performing better, in general, than GLS on all of the test sets; therefore we focus on GLS2 in our analysis. Another result of this stagnation behaviour is that the ratio of the mean to the median of the RTDs is much higher for GLS and GLS2 than for IRoTS, which is more "well-behaved". Because the median is a more stable summary statistic, we report statistics only of the median in all our results. When considering mean instead of median run-times, the ratios of the run-times of GLS and GLS2 to IRoTS are even higher than the ratios that we report.

The ILS-YI algorithm using the 1-flip neighbourhood (ILS-YI(1)) required multiple orders of magnitude more CPU time than IRoTS on all of the weighted and unweighted Random 3-SAT test sets, and was unable to solve some instances from the `wjnh` test set within 1 CPU hour. The ILS-YI algorithm using the 2-flip neighbourhood (ILS-YI(2)) performed significantly better than ILS-YI(1), though still required greater than an order of magnitude more CPU time than IRoTS in all cases, and performed particularly badly on the `wjnh` test set (there were instances from `wjnh` which required greater than 1 CPU hour for ILS-YI(2) to solve). Because of these poor results, for the sake of brevity we don't report further details for ILS-YI here.

Figure 4 shows a scatterplot of the median CPU time required for IRoTS and GLS2 for solving each of the `wjnh` instances. Some of the instances in this test set are actually satisfiable (in these cases, the clause weights are redundant, since an optimal solution will have a cost of zero), and have been marked in the plots. We see that the satisfiable instances tend to be easier for both IRoTS and GLS2. For 39 of the 44 instances in the test set, GLS requires fewer steps that IRoTS to find an optimal solution; the median sc for IRoTS is 14896, while the median sc for GLS2 is 2977 search steps. However, when we consider the median CPU time required to find an optimal solution we see that

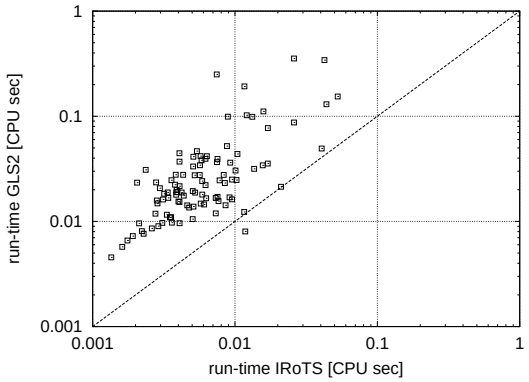

Fig. 5. Performance of IRoTS vs. GLS2 on the `rnd100-w100` test set, measuring median CPU time for finding optimal solutions.

IRoTS requires less CPU time than GLS2 on 28 of the 44 instances, and the median CPU time for IRoTS to find a solution is 0.038 seconds, while the median CPU time for GLS2 is 0.050 seconds. In summary, IRoTS performs on par with GLS, the best known algorithm for the `wjnh` instances. Since GLS was shown to perform better than a variety of earlier proposed algorithms on this test set [18, 19, 21, 11], this fact also applies to IRoTS.

We ran all of the algorithms on the `bor` test set, and present a short summary of these results here. Overall, IRoTS was the best performing algorithm on these test sets. IRoTS typically required between 1.5 and 5 times less CPU time than GLS and GLS2 for finding optimal solutions. For example, on the 150 variable weighted 3-SAT test set, the mean run-time for IRoTS, GLS, and GLS2 are 0.0093, 0.024, and 0.018 CPU seconds, respectively. The instances tend to get easier (as the clause/variable ratio is increased) for IRoTS but harder for GLS and GLS2 (similar results are reported later in this section for the unweighted Random 3-SAT test sets). The ILS-YI algorithms both required over 5 times more CPU time to solve all of the test sets than all of the other algorithms, with the worst results for the larger test sets. RoTS performed as well as or better than GLS and GLS2, and worse than IRoTS in almost all cases.

Because there are so few instances in each of these test sets, the data presented above for the `bor` test set may not be representative of the typical behaviour of these algorithms on random test sets. To address this question, we now consider the performance of the algorithms on the weighted and unweighted Random 3-SAT test sets described in Section 3. Again, we report only results for IRoTS and GLS2, which were the two best performing algorithms on these test sets.

Figure 5 shows a scatter plot of the median CPU time required for IRoTS and GLS2 to optimally solve each instance in the `rnd100-w100` test set. We see that IRoTS requires less CPU time than GLS2 on all but one instance. We also notice that the CPU times are positively correlated, but that there is a significant amount of noise in the correlation especially for the harder instances. Interestingly, the amount of noise is proportional to σ^2 (the variance in the clause weights) — as the variance in the

Test Set	run-length [search steps] IRoTS q_{50}	IRoTS $^{q90}/_{q10}$	GLS2 q_{50}	GLS2 $^{q90}/_{q10}$	$f.b.$	$s.f.$	run-time [CPU sec $\times 10^{-4}$] IRoTS q_{50}	IRoTS $^{q90}/_{q10}$	GLS2 q_{50}	GLS2 $^{q90}/_{q10}$	$f.b.$	$s.f.$
rnd50-250u	**113**	7.9	142	9.6	0.7	1.3	**5**	3.5	18	4.7	1.0	3.6
rnd50-w50	**274**	8.0	448	8.7	0.19	1.7	**9**	3.8	41	6.2	1.0	4.5
rnd50-w250	**574**	10.1	754	19.6	0.37	1.3	**13**	6.1	68	15.6	0.99	5.2
rnd100-500u	639	10.3	**618**	8.1	0.39	0.97	**21**	5.6	88	4.6	1.0	4.2
rnd100-w100	2202	7.5	**2160**	10.6	0.54	0.98	**52**	6.0	207	9.1	0.99	4.0
rnd100-w500	6591	11.1	**6126**	39.5	0.5	0.93	**131**	10.6	570	36.7	0.88	4.4
rnd200-1000u	6630	13.3	**5665**	23.6	0.54	0.85	**240**	11.2	712	21.2	0.99	3.0
rnd200-w200	**45648**	18.7	69523	22.7	0.56	1.5	**1449**	18.2	8244	22.1	0.95	5.7
rnd200-w1000	318836	21.4	**217964**	24.4	0.26	0.68	**10166**	20.9	28277	27.5	0.71	2.8

Table 2. Summary statistics of the search cost distribution for IRoTS and GLS2 on the Random 3-SAT test sets; q_x denotes the xth percentile; $f.b.$ is the fraction of instances in the test set with sc(IRoTS) $< sc$(GLS2); $s.f.$ is the "speedup-factor", *i.e.* the ratio q_{50}(GLS2) / q_{50}(IRoTS).

Test Set	run-length [search steps] IRoTS q_{50}	IRoTS $^{q90}/_{q10}$	GLS2 q_{50}	GLS2 $^{q90}/_{q10}$	$f.b.$	$s.f.$	run-time [CPU sec $\times 10^{-4}$] IRoTS q_{50}	IRoTS $^{q90}/_{q10}$	GLS2 q_{50}	GLS2 $^{q90}/_{q10}$	$f.b.$	$s.f.$
rnd100-500u	639	10.3	**618**	8.1	0.39	0.97	**21**	5.6	88	4.6	1.0	4.2
rnd100-700u	**514**	6.2	1016	9.5	0.95	2.0	**19**	3.5	139	6.8	1.0	7.3
rnd100-850u	**612**	5.9	1937	6.3	1.0	2.3	**24**	3.7	288	5.7	1.0	12.0
rnd100-1000u	**499**	6.3	2193	7.4	0.99	4.4	**22**	3.1	404	6.4	1.0	18.4

Table 3. Summary statistics of the search cost distribution for IRoTS and GLS2 on 100 variable unweighted Random 3-SAT test sets with increasing clauses/variables ratio.

clause weight distribution increases, the correlation between the sc of IRoTS and GLS2 decreases.

Table 2 shows summary statistics of the distribution of search costs for IRoTS and GLS2 on the Random 3-SAT test sets. Clearly, IRoTS performs better than GLS2 on these test sets; IRoTS requires fewer flips for 4 out of the 9 test sets, and less CPU time than GLS2 in all cases. Table 3 shows the relative performance of IRoTS and GLS2 on the unweighted 100 variable Random 3-SAT test sets as the clauses/variables ratio increases. Note that the relative performance of IRoTS vs. GLS2 increases with the clauses/variables ratio. Furthermore, the instances become progressively easier for IRoTS as the level of constrainedness increases, while at the same time they become more and more difficult for GLS2. For the most overconstrained test sets, the median CPU time for finding optimal solutions is more than an order of magnitude less for IRoTS than for GLS2.

It may be noted that many of the Random-3-SAT instances are optimally solved within the first RoTS local search phase of the IRoTS algorithm (which is terminated after $n^2/4$ iterations without improvement), indicating that in these cases, RoTS alone is sufficient for finding optimal solutions. This is, however, not generally true for the

Algorithm	YI-GCP		YI-SCP		LGCMP	
	q_{50}	q_{90}/q_{10}	q_{50}	q_{90}/q_{10}	q_{50}	q_{90}/q_{10}
GLS	**0.36**	1.18	**0.35**	0.95	11.15	1.07
GLS2	2.12	1.18	1.27	0.92	11.56	1.07
IRoTS	4.46	1.75	3.39	1.05	**1.18**	0.92
ILS-YI(1)	–	–	–	–	2.79	0.93
ILS-YI(2)	0.78	1.19	–	–	16.38	1.10
RoTS	–	–	–	–	1.96	0.93

Table 4. Summary statistics of search costs distribution on test sets of structured instances for each of the algorithms, measured in CPU seconds. Entries marked with a '–' indicate that the respective algorithm was unable to find the desired solution quality within 1 CPU hour.

hardest instances, where IRoTS shows substantially improved performance over RoTS; this performance advantage of IRoTS over RoTS is even more pronounced for the structured instances considered in the next section.

5 Results for Structured Instances

For the MAX-SAT encoded graph colouring, set covering, and crossing minimisation instances [4], IRoTS with the default parameters performs rather poorly. This is not surprising, since these instances are quite different from the previously considered random instances in terms of their syntactic properties (such as clause length and weight distributions). To obtain better performance, we changed the *escape_threshold* parameter to 100 steps, and used a different perturbation mechanism, in which instead of performing RoTS with a large tabu tenure, each variable is flipped independently at random with probability 0.05. The latter modification was motivated by the observation that using RoTS in the perturbation phase resulted in stagnation of the search process. We believe that the reason for this lies in the fact that RoTS is based on a greedy heuristic, and even though when using RoTS for perturbation the tabu tenure is set very high, the underlying greedy search mechanism seems to be unable to escape from the deep local optima that appear to be encountered when solving these instances.

Due to the extremely large computation times required by some algorithms for finding optimal (or best known) solutions of the structured instances considered here, we performed our comparative analysis based on RTDs for suboptimal solution qualities. For each instance, we ran the worst-performing of IRoTS, GLS, and GLS2 10 times with a fixed cutoff of 10 seconds. Then we chose the 90^{th} percentile of the solution quality distribution observed for that run-time as a target for all of the algorithms, and report statistics on the time required to find the respective (typically sub-optimal) solution quality.

The results from these experiments are shown in Table 4. The relative performance of IRoTS, GLS, and GLS2 is similar for the graph colouring and set covering instances, with GLS performing best by a large margin. IRoTS was able to solve all of the instances in these two test sets, but required approximately an order of magnitude more

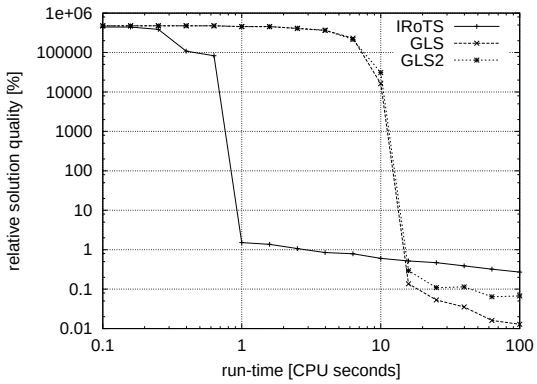

Fig. 6. SQT for a typical 125 node crossing minimisation instance. The y-axis shows the median relative solution quality found at each point in time, calculated as $sq(t)/sq^* - 1$, where $sq(t)$ is the median absolute solution quality reached at time t and sq^* is the best quality solution ever found by any of the algorithms studied in this paper.

CPU time than GLS. The performance of GLS2 and IRoTS was more comparable, but GLS2 was still better by a factor of more than 2. Interestingly, the 2-flip ILS-YI algorithm performed very well on the graph colouring instances, but not on the set covering instances.

IRoTS shows more promise on the MAX-SAT encoded crossing minimisation instances, where it requires less than an order of magnitude less CPU time to find solutions of the given quality than GLS, GLS2, and the 2-flip ILS-YI algorithm. The 1-flip ILS-YI algorithm also performs well on these instances, though IRoTS is still a factor of 2 better than ILS-YI. Interestingly, this is the only test set where the 1-flip ILS-YI algorithm was among the best performing, indicating that Iterated Local Search in general may be well-suited to this type of MAX-SAT instances.

Additional experiments indicate that GLS and GLS2 perform better on the MAX-SAT encoded crossing minimisation instances when searching for higher quality solutions. Figure 6 shows the development of solution quality over time (SQT) for a 125 node crossing minimisation instance. We see that IRoTS finds much higher quality solutions than either GLS variant in short runs (this is reflected in the results reported in Table 4), but that both GLS variants find higher quality solutions if sufficiently long run-times are allowed. The results for random instances (where IRoTS outperforms GLS and GLS2) are qualitatively different, as Figure 7 shows. It is encouraging to note that IRoTS eventually finds solutions within 1% of the best known solution qualities, and that we see no evidence of stagnation for IRoTS in the SQTs.

The results of IRoTS on the structured instances also show a very strong improvement over RoTS at least for the graph colouring and set covering instances. This is further evidence that the underlying RoTS algorithm can be significantly improved by embedding it into the Iterated Local Search framework.

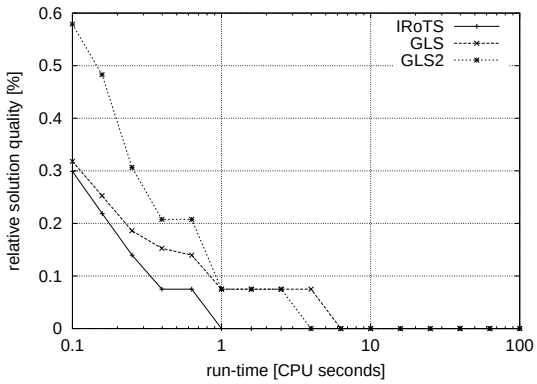

Fig. 7. SQT for an instances from the `rnd200-w1000` test-set with median sc for both IRoTS and GLS2. The y-axis shows the median relative solution quality found at each point in time, calculated as $sq(t)/sq^* - 1$, where $sq(t)$ is the median absolute solution quality reached at time t and sq^* is the best quality solution ever found by any of the algorithms studied in this paper.

6 Conclusions and Future Work

In this work we introduced a new stochastic local search algorithm for MAX-SAT, Iterated Robust Tabu Search (IRoTS). This algorithm combines two SLS methods that have been used very successfully for solving a variety of other hard combinatorial optimisation problems, Iterated Local Search (ILS) and Robust Tabu Search (RoTS). Our empirical analysis of IRoTS on a range of MAX-SAT instances, including weighted and unweighted as well as randomly generated and structured instances, shows that in many cases IRoTS outperforms GLS, one of the best-performing MAX-SAT algorithms currently known, and ILS-YI, an earlier and simpler Iterated Local Search algorithm. However, we also observed cases in which the performance of IRoTS did not reach that of GLS, such as MAX-SAT-encoded Minimum Cost Graph Colouring and Set Covering problems.

Like most other papers on MAX-SAT algorithms in the literature, we have focused mainly on randomly generated MAX-SAT instances and present only limited results on structured instances. We are currently extending this evaluation to additional sets of MAX-SAT-encoded instances from other domains with the goal of obtaining a better understanding of how the behaviour of state-of-the-art MAX-SAT algorithms differs between structured and random instances. Given an increased recent interest in using MAX-SAT algorithms for solving encoded instances of other combinatorial problems, such as MPE finding in Bayes Nets [17], the results from such a study should be highly relevant for the assessment and future development of MAX-SAT algorithms.

Our experimental results indicate that IRoTS performs particularly well on unweighted instances that are highly overconstrained, and therefore have optimal solutions with a large number of unsatisfied clauses, and on weighted instances with high variability clause weight distributions. In future work, we plan to further investigate how the performance of IRoTS and other MAX-SAT algorithms, in particular GLS, de-

pend on features of the given problem instance. An important part of this is the analysis of the underlying search spaces.

We recently developed generalisations of Novelty$^+$, a state-of-the-art SLS algorithm for SAT [9], to weighted MAX-SAT. Preliminary experimental results indicate that in many cases, including the `wjnh` instances as well as some of the structured instance sets considered here, these algorithms outperform GLS and IRoTS. In many of the cases in which IRoTS is particularly successful, these Novelty$^+$ variants don't reach its performance. On the other hand, we have presented limited evidence in this paper that IRoTS does not achieve state-of-the-art performance on SAT instances. This suggests that inherently different SLS strategies are required for efficiently solving SAT and at least certain types of MAX-SAT instances. This hypothesis will be further investigated in future research and might shed new light on the fundamental differences between solving decision and optimisation problems.

Overall, we see this present work as the first in a series of empirical studies that characterise and improve the state-of-the-art in solving MAX-SAT while providing deeper insights into the MAX-SAT problem and the behaviour of high-performance SLS algorithms for combinatorial optimisation problems. The fact that even at this early stage, we improved over state-of-the-art algorithms on a wide range of instances is very encouraging and, in our mind, illustrates the overall potential in this line of work.

Acknowledgments. This work has been supported by NSERC Individual Research Grant #238788 and by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. We thank Patrick Mills and Edward Tsang for providing us with their GLS implementation. Furthermore, we are indebted to Ewald Speckenmeyer and Mattias Gaertner for pointing out the Level Graph Crossing Minimisation Problem and for providing us with instances of this problem.

References

1. P. Asirelli, M. de Santis, and A. Martelli. Integrity constraints in logic databases. *J. of Logic Programming*, 3:221–232, 1985.
2. R. Battiti and M. Protasi. Reactive search, a history-based heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2, 1997.
3. B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 1999.
4. B. Randerath et al. A satisfiability formulation of problems on level graphs. Technical Report 40–2001, Rutgers Center for Operations Research, Rutgers University, Piscataway, NJ, USA, June 2001.
5. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, 1979.
6. I.P. Gent and T. Walsh. Towards an understanding of hill–climbing procedures for SAT. In *Proceedings of AAAI'93*, pages 28–33. MIT Press, 1993.

7. J. Gu and R. Puri. Asynchronous circuit synthesis with boolean satisfiability. *IEEE Transact. of Computer-Aided Design of Integrated Circuits and Systems*, 14(8):961–973, 1995.
8. P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.
9. H.H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proc. AAAI-99*, pages 661–666. MIT Press, 1999.
10. H.H. Hoos and T. Stützle. Evaluating Las Vegas algorithms — pitfalls and remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245. Morgan Kaufmann, San Francisco, 1998.
11. Y. Jiang, H. Kautz, and B. Selman. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *Proceedings of the 1st International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.
12. S. Joy, J. Mitchell, and B. Borchers. A branch and cut algorithm for MAX-SAT and weighted MAX-SAT. In D. Du, J. Gu, and P.M. Pardalos, editors, *Satisfiability problem: Theory and Applications*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 519–536. American Mathematical Society, 1997.
13. A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.
14. H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.
15. P. Mills and E. Tsang. Guided local search for solving SAT and weighted MAX-SAT problems. In I.P. Gent, H. van Maaren, and T. Walsh, editors, *SAT2000 — Highlights of Satisfiability Research in the Year 2000*, pages 89–106. IOS Press, 2000.
16. D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. of AAAI'92*, pages 459–465. MIT Press, 1992.
17. James Park. Using weighted MAX-SAT to approximate MPE. In *Proc. of AAAI-02*, pages 682–687. AAAI Press, 2002.
18. M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In D. Du, J. Gu, and P.M. Pardalos, editors, *Satisfiability problem: Theory and Applications*, volume 35, pages 393–405. AMS, 1997.
19. Y. Shang and B.W. Wah. Discrete lagrangian-based search for solving MAX-SAT problems. In *Proc. of IJCAI'97*, volume 1, pages 378–383. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.
20. É.D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.
21. Z. Wu and B.W. Wah. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proc. of AAAI'99*, pages 673–678. MIT Press, 1999.
22. M. Yagiura and T. Ibaraki. Efficient 2 and 3-flip neighborhoods seach algorithms for the MAX SAT. In W.-L. Hsu and M.-Y. Kao, editors, *Computing and Combinatorics*, volume 1449 of *Lecture Notes in Computer Science*, pages 105–116. Springer Verlag, Berlin, Germany, 1998.
23. M. Yagiura and T. Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: Experimental evaluation. *Journal of Heuristics*, 7(5):423–442, 2001.