A review of the literature on local search algorithms for MAX-SAT

Thomas Stützle, Holger Hoos
Intellectics Group, Darmstadt University of Technology

Andrea Roli
DEIS, Università degli Studi di Bologna

1 Introduction

The satisfiability problem in propositional logic (SAT) is the task to decide for a given propositional formula whether it has a model. This problem plays a prominent role in various areas of computer science, mathematical logic and artificial intelligence, but also in applications such as asynchronous circuit synthesis [29], inductive inference [41], integrity of databases [3], hardware verification and many others. SAT was also the first problem to be proven $\mathcal{NP}$-complete [11] and as such is amongst the central problems in theoretical computer science.

Every satisfiability problem can be expressed in a standard form, called a conjunctive normal form (CNF), that is, as a conjunction of clauses, each clause being a disjunction of literals and each literal being a variable or its negation.

More formally, let $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ be the set of m clauses that involve n Boolean variables $x_1, x_2, \ldots, x_n$ that can take the value 0 and 1 (or *true* and *false*). Each clause C_i is a disjunction of n_i literals, that is

$$C_i = \bigvee_{j=1}^{n_i} l_{ij} \tag{1}$$

where each literal is either a variable x_j (that is, a positive literal) or its negation $\neg x_j$ (that is, a negative literal). Without loss of generality we assume that at most one of x_j and $\neg x_j$ is included in each clause. A clause is satisfied, that is, it evaluates to 1 (true), if at least one of the positive literals in the clause is assigned the value 1 (true) or a negative literal is assigned the value 0 (false). The SAT problem then is to decide whether an assignment of values to variables exist such that all clauses are simultaneously satisfied, that is, whether the propositional formula

$$\Phi = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n_i} l_{ij} \tag{2}$$

is satisfiable. In fact, instead of just deciding whether such an assignment exists, one rather wants to find such an assignment, in case it exists: this is the model-finding variant of SAT, a well known $\mathcal{NP}$-hard problem.

MAX-SAT is the optimization variant of SAT. Given a set of clauses, MAX-SAT is the problem to find a variable assignment that maximizes the number of satisfied clauses. In *weighted MAX-SAT* additionally a weight w_i is assigned to each clause C_i and the goal becomes to maximize the weight of the satisfied clauses. (Alternatively, the objective function could be defined as to minimize the weight of the unsatisfied clauses.) More formally, let w_i be the weight of clause C_i. Then the objective function is defined as

$$f(x) = \sum_{i=1}^{m} w_i \cdot I(C_i), \tag{3}$$

where $I(C_i)$ is one, if and only if the clause C_i is satisfied and otherwise zero. In case the weights are not specified, that is, the instance is unweighted, we set $w_i = 1, \forall i = 1, \ldots, m$. In case of possible confusions we will call this latter variant *unweighted MAX-SAT*; when we refer to MAX-SAT in general, we refer to the general form of the problem including clause weights.

Regarding local search approaches for SAT and MAX-SAT we discuss from Section 3 on, a first remark is that both problems are typically treated identically: when trying to find models to SAT formulae, one actually treats these formulae as MAX-SAT instances and tries to find a variable assignment satisfying all clauses. If such an assignment is found, a model of the formula is obtained. This close relationship between the local search approaches for the two problems is also the reason, why we treat them together, here. Nevertheless, there are some particular differences in the hardness of some special cases of SAT and MAX-SAT. For example, it is well known that 2-SAT, the SAT problem restricted to clauses of length two, is a polynomially solvable special case of SAT. Nevertheless, the MAX-2-SAT problem is known to be $\mathcal{NP}$-hard.

In this article we give an overview of local search approaches for solving MAX-SAT. Although at some occasions we also will refer to algorithms originally intended to solve the SAT problem, we will not give a detailed overview of these algorithms and rather refer to the article by Hoos and Stützle [37] for a detailed comparison of state-of-the-art stochastic local search algorithms for SAT and to the extensive overview article by Gu, Purdom, Franco, and Wah [28] for a general overview of SAT algorithms.

The remainer of this article is structured as follows. First, we give an overview of currently available benchmark instances for MAX-SAT in Section 2. Then we give details on two basic local search architectures for SAT and MAX-SAT problems, the so called GSAT and WSAT architectures in Section 3, and extensions thereof, followed by a concise description of oblivious local search and large neighborhoods for solving MAX-SAT. Next, in Section 4 we present the currently available metaheuristic approaches to MAX-SAT and finally we give in Section 5 some pointers to exact algorithms and other approximation algorithms. We conclude in Section 6.

2 Benchmark instances

There are only few MAX-SAT instances publically available.[1] One of the most wide-spread benchmark sets is one of small, weighted MAX-SAT instances proposed by Resende, Pitsoulis, and Pardalos [53]. These instances are derived from SAT instances submitted by Hooker to the DIMACS Challenge on Graphs, Coloring and Satisfiability which are known to be easily solvable. The instances have 100 variables and between 800 and 900 clauses. The clauses are generated according to the constant probability model, in which each variable is included into a clause with a probability p and then negated with a probability of 0.5. Resende at al. converted these instances to weighted MAX-SAT instances by adding to each clause an integer weight that is uniformly distributed in the interval $[1, 1000]$. Despite their small size, these instances are used as the only benchmark set for a number of algorithms [46, 53, 49, 60, 66]. Nevertheless, one should be so honest to say that these instances do not pose anymore a reasonable challenge to state-of-the-art algorithms for MAX-SAT.

Other MAX-SAT instances, which were used in several researches, comprise mainly unweighted, uniform k-SAT instances, where each clause contains exactly k variables. Often, researchers considered 3-SAT instances in this case, although it is known that also MAX-2-SAT is $\mathcal{NP}$-hard. These instances typically are generated according to the *fixed clause length model* [47]: Given are the number of variables n and the number of clauses l; each clause is produced by choosing k variables at random according to a uniform distribution (that is, each variable is chosen independently of the others with a probability of $1/n$) and then negating each of these variables with probability 0.5. In the clause generation process, tautological clauses (that is, clauses containing a literal and its negation) and clauses with multiple occurrences of the same literal are rejected; clauses are produced until l clauses (not considering rejected clauses) are generated. Thus, for each value of n and l, a random distribution of Random-k-SAT formulae is given; test sets generated as described above correspond to samples from these distributions.

It is well known that these random test sets underly a phase transition phenomenon. For example, for Random-3-SAT there is a phase transition region at ca. 4.26 clauses per variable [47, 43]. For higher ratios, Random-3-SAT instances are typically unsatisfiable (and therefore also interesting as MAX-SAT instances), while for lower ratios it is very easy to find solutions which satisfy all clauses. Therefore, MAX-3-SAT instances are typically obtained at high clause–variable ratios.

An example for such a set of instances is the one used by Battiti and Protasi [4]. This set consists of 10 instances of each size, where the number of variables ranges from 100 to 1000 and the clause–variable ratio ranges between 5 and 10. These benchmark instances are available for download from the page `http://rtm.science.unitn.it/intertools/sat/benchmark.html`.

Some more MAX-2-SAT and MAX-3-SAT instances were randomly generated by Borchers et al. for a series of papers investigating the performance of exact algorithms either based on

[1]We could consider all SAT instances to be MAX-SAT instances, but this interpretation was originally not intended and therefore we do not discuss these instances here in detail. We rather refer to the SATLIB website accessible at `http://www.satlib.org`, where most of these instances are available for download; at the SATLIB side there always is included also a short description of these instances.

extensions of the Davis-Putnam-Loveland algorithm or based on Branch&Cut. However, the instances they used were of rather small size, the largest having 150 variables and 750 clauses, and therefore also are not a real challenge for efficient local search methods. These instances are available at `http://www.nmt.edu/~borchers/maxsat.html`.

In addition to these instances also all SAT instances from SATLIB can be used as benchmarks for algorithms solving MAX-SAT, because in this case for solvable instances, that is for instances for which a variable assignment can be found that satisfies all clauses, an optimal solution for the MAX-SAT problem simply corresponds to a solution satisfying all clauses.

3 Local search architectures for SAT and MAX-SAT

In this section we present the GSAT and the WalkSAT families of algorithms, which are the two main local search architectures underlying almost all local search algorithms for SAT (and also MAX-SAT) and review some of the most widely used members of these families. Before going into details of these architectures, we mention that the algorithms developed in the line of these two architectures were originally intended only to solve SAT instances, but they directly can be applied to MAX-SAT problems, too. For the presentation of the two local search architectures, we will mainly use the jargon when attacking MAX-SAT instead of the more SAT oriented jargon that typically is used when presenting these algorithms. The first of these architectures, the GSAT architecture, was introduced in 1992 applying it to SAT. Nevertheless, there existed already at least one earlier local search approach, the Steepest Ascent–Mildest Descent approach by Hansen and Jaumard [30], that uses a similar underlying local search engine but was applied "only" to MAX-SAT.

3.1 Basics

In local search algorithms for MAX-SAT and SAT, the search space typically comprises the set of all possible truth value assignments for the variables occurring in the given problem instance and the solutions are the assignments corresponding to optimal solutions. The search space thus defined is of exponential size in the number of propositional variables.

Most local search based MAX-SAT algorithms use a 1-flip neighborhood relation for which two truth value assignments are neighbors if they differ in the truth value of exactly one variable. Thus, the local search steps modify the truth value assigned to one propositional variable; such a move is called a *variable flip*. Nevertheless, few extensions exist that consider flipping two or three variables at a time; such a local search algorithm is introduced in Section 3.4.

Typically, local search algorithms for MAX-SAT use an objective function which is defined as the sum of the weights of the clauses that are satisfied under the given variable assignment.[2] The general idea for solving MAX-SAT by local search is to perform a random walk in the search space which is biased such that the weight of the satisfied clauses is maximized (obviously, this

[2] In the case a SAT instance is satisfiable, the global optima of the instance correspond to satisfying variable assignments.

is equivalent to minimizing the weight of the unsatisfied clauses). The main difference between different local search algorithms for SAT and MAX-SAT is in the step function, that is, in the strategy used to select the variable to be flipped next.

Local search algorithms are typically incomplete. This is noticed by the fact that, in general, they cannot guarantee to find an optimal solution for the instance attacked in finite time. In the SAT case this actually means that for satisfiable problem instances they cannot be guaranteed to find a solution. The reason for this is the non-systematic nature of the search. Furthermore, local search algorithms can get trapped in local minima and plateau regions of the search space [19, 18], leading to premature stagnation of the search. One of the simplest mechanisms for avoiding premature stagnation of the search is random restart, which reinitializes the search if after a fixed number of steps (cutoff time) no solution has been found. In fact, random restart is used in many local search algorithms for SAT and also in some for MAX-SAT.

A general outline of a local search algorithm for MAX-SAT is given in Figure 1. The generic procedure initializes the search at some truth value assignment and then iteratively flips some variable's truth value, where the selection of the variable depends on the formula Φ and the current assignment. It terminates after a maximum of *maxSteps* flips.

In the special case of local search algorithms for SAT, often an additional restart mechanism is used that restarts the local search from a new, random initial assignment if after *maxSteps* no solution is found. In this case the algorithm terminates after a given number *maxTries* of unsuccessful restarts.

```
procedure LocalSearch for MAX-SAT
    input MAX-SAT formula $\Phi$ in CNF, maxSteps
    output best solution found
        $s := initAssign(\Phi)$;
        $s_{best} := s$;
        for $j := 1$ to maxSteps do
            if $f(s) < f(s_{best})$ then $s_{best} := s$;
            if $s$ satisfies $\Phi$ then return $s$;
            else
                $x := chooseVariable(\Phi, s)$;
                $s := s$ with truth value of $x$ flipped;
            end if
        end for
    return $s_{best}$;
end LocalSearch for MAX-SAT
```

Figure 1: Outline of a general local search procedure for MAX-SAT.

For many concrete algorithms, and in particular, for all algorithms investigated here, *initAssign* randomly chooses the initial assignment from the set of all possible assignments according to a uniform distribution. Hence, the main difference between local search algorithms for MAX-SAT is typically the implementation of the step function as given by the procedure *chooseVariable*.

3.2 The GSAT Architecture

The GSAT algorithm was introduced in 1992 by Selman, Levesque, and Mitchell [59]. It is based on a rather simple idea: GSAT tries to maximize the number of satisfied clauses by a greedy ascent in the space of variable assignments. Variable selection in GSAT and most of its variants is based on the *score* of a variable x under the current assignment s; this is defined as the difference between the weight of the clauses *un*satisfied by s and the assignment obtained by flipping x in s.[3]

3.2.1 Basic GSAT

The basic GSAT algorithm uses the following instantiation of the procedure *chooseVariable*(s, Φ). In each local search step, one of the variables with maximal score is flipped. If there are several variables with maximal score, one of them is randomly selected according to a uniform distribution. A straightforward implementation of GSAT may be rather inefficient, since in each step the score of all variables would have to be calculated from scratch. The key to efficiently implementing GSAT is to evaluate the complete set of scores only once at the beginning of each try, and then after each flip to update only the scores of those variable which were possibly affected by the flipped variable.

One problem with basic GSAT is that it can easily get stuck in local minima of the objective functions. As these might contain a large number of search space positions, between which GSAT can arbitrarily wander, local minima cannot be efficiently detected in general and the built-in random restart mechanism is the only way for escaping from these.

3.2.2 GSAT with Random Walk (GWSAT)

An important extension of the basic GSAT algorithm is GSAT with random walk (GWSAT) [58] that introduces a second type of local search step, the so-called *random walk step*. In a random walk step, first a currently unsatisfied clause C' is randomly selected. Then, one of the variables appearing in C' is flipped, thus effectively forcing C' to become satisfied. The basic idea of GWSAT is to decide at each local search step with a fixed probability *p* (called *walk probability* or noise setting) whether to do a standard GSAT step or a random walk step. Obviously, for arbitrary $p > 0$ this algorithm allows arbitrary sequences of random walk steps; as detailed in [36], this implies that from arbitrary assignments, a model (if existent) can be reached with a positive, bounded probability, making this algorithm probabilistically approximately complete (PAC).

3.2.3 GSAT with tabu search (GSAT/TABU)

Another well-known mechanism for preventing the search from getting stuck in local optima is tabu search [22, 24, 30]. The general idea is to forbid reversing the effect of a particular move

[3] To avoid possible confusion with other descriptions of GSAT, in the following few subsections we treat MAX-SAT in its minimization form, which is the form actually attacked by GSAT.

for a number *tl* of iterations; the parameter *tl* is called the *tabu tenure*. This mechanism can be easily added to basic GSAT [30, 44, 62]; in the resulting algorithm, after a variable x has been flipped it cannot be flipped back within the next *tl* steps. GSAT/TABU can be efficiently implemented by storing for each variable x the iteration number i_x when it was last flipped. When initializing the search, all the i_x are set to $-tl$ and every time a variable x is flipped, i_x is set to the number j of the current iteration. Obviously, a variable x can only be flipped if $j > i_x + tl$.

It should be noted that GSAT/TABU corresponds basically to the Steepest Ascent–Mildest Descent algorithm proposed by Hansen and Jaumard [30], but for SAT it was rediscovered only much later.

3.2.4 HSAT

HSAT [20] is another GSAT variant where the local search steps make use of history information. When in a search step there are several variables with identical score, HSAT selects the least recently flipped variable, that is, the variable which was flipped longest ago. Only shortly after search initialization, when there are still variables which have not been flipped, a random selection like in GSAT is done. Although when compared to plain GSAT, HSAT was found to show superior performance [20], it is clear that it is even more likely to get stuck in local minima from which it cannot escape, as the additional history-based tie-breaking rule effectively restricts the search trajectories when compared to GSAT. Therefore, it appears to be attractive to extend HSAT with the same random walk mechanism as used in GWSAT; the resulting variant is called HWSAT [21], and like GWSAT it has the PAC property.

3.3 The WalkSAT Architecture

The WalkSAT architecture is based on ideas first published by Selman, Kautz, and Cohen in 1994 [58] and it was later formally defined as an algorithmic framework by McAllester, Selman, and Kautz in 1997 [45]. It is based on a 2-stage variable selection process focused on the variables occurring in currently unsatisfied clauses. For each local search step, in a first stage a currently unsatisfied clause c' is randomly selected. In a second step, one of the variables appearing in c' is then flipped to obtain the new assignment. Thus, while the GSAT architecture is characterized by a static neighborhood relation between assignments with Hamming distance one, WalkSAT algorithms are effectively based on a dynamically determined subset of the GSAT neighborhood relation.

3.3.1 WalkSAT

WalkSAT, originally introduced in [58], differs in one important aspect from the other local search variants discussed here: The scoring function $score_b(x)$ used by WalkSAT counts only the weight of the clauses that are broken — that is, which are currently satisfied, but will become unsatisfied by flipping a given variable. Using this scoring function, the following variable selection scheme is applied: If there is a variable with $score_b(x) = 0$ in the clause c' selected in stage 1, that is, if c' can be satisfied without breaking another clause, this variable is flipped ("zero-damage" flip). If no such variable exists, with a certain probability *p* (noise setting) the variable with maximal $score_b$ value is selected; in the remaining cases, one of the variables from c' is randomly selected (random walk flip).

Conceptually as well as historically, WalkSAT is closely related to GWSAT. However, there are a number of significant differences between both algorithms, which in combination account for the generally superior performance of WalkSAT when applied to SAT. While both algorithms use the same kind of random walk steps, WalkSAT applies them only under the condition that there is no variable with $score_b(x) = 0$. In GWSAT, however, random walk steps are done in an unconditional probabilistic way. From this point of view, WalkSAT is greedier, since random walk steps, which usually increase the weight of the unsatisfied clauses, are only done when every variable occurring in the selected clause would break some clauses when flipped. Yet, in a greedy step, WalkSAT chooses due to the two-stage variable selection scheme from a significantly reduced set of neighbors and can, therefore be considered to be less greedy than GWSAT. Finally, because of the different scoring function, in some sense, GWSAT shows a greedier behavior than WalkSAT: For a GSAT step, it would prefer a variable which breaks some clause but compensates for this by fixing some other clauses, while in the same situation, WalkSAT would select a variable with a smaller total score.

3.3.2 WalkSAT/TABU

Analogously to GSAT/TABU, there is also a WalkSAT variant which uses a Tabu Search mechanism. This algorithm is called WalkSAT/TABU [45]. It uses the same two stage selection mechanism and the same scoring function $score_b$ as WalkSAT and additionally enforces a tabu tenure of *tl* steps for each flipped variable. Here, if no zero damage flip can be made, from all variables which are not tabu, the one with the highest $score_b$ value is picked; when there are several variables with the same maximal score, one of them is randomly selected according to a uniform distribution. As a result of the two-level variable selection scheme, it may however happen that all variables appearing in the selected clause cannot be flipped because they are tabu. In this case, no variable is flipped (a so-called *null-flip*).

3.3.3 Novelty

Novelty, introduced in [45], is one of the most recent local search algorithms for SAT. Conceptually, it combines the algorithms based on the WalkSAT architecture with a history-based variable selection mechanism in the spirit of HSAT. Novelty, too, is based on the intuition, that repeatedly flipping back and forth the same variable should be avoided. Additionally, like for tabu search variants, the number of local search steps which have been performed since it was last flipped (also called the variable's *age*) is taken into consideration. An important difference of Novelty to WalkSAT and WalkSAT/TABU is that it uses the same scoring function as GSAT.

In Novelty, after an unsatisfied clause has been chosen, the variable to be flipped is selected as follows. If the variable with the highest score does not have minimal age among the variables within the same clause, it is always selected. Otherwise, it is only selected with a probability of *1-p*; in the remaining cases, the variable with the next lower score is selected. In Kautz' and Selman's implementation, if there are several variables with identical score, always the one appearing first in the clause is chosen.

3.3.4 R-Novelty

R-Novelty, also introduced in [45], is a variant of Novelty which is based on the intuition that, when deciding between the best and second best variable (using the same score function as for Novelty), the actual difference of the respective scores should be taken into account. Note that the R-Novelty heuristic is quite complex – as reported in [45], it was discovered by systematically testing a large number of WalkSAT variants.

3.4 Complex Neighborhoods for SAT

Most known local search algorithms for SAT and MAX-SAT rely on the 1-flip neighborhood. One exception is the 2 and 3-flip neighborhood local search algorithm for MAX-SAT proposed by Yagiura and Ibaraki [67, 68]. Because the computational time to examine the effects of 2 and 3-flips is high, they use special data structures to speed-up as much as possible the neighborhood evaluation. In addition, they propose restrictions to both neighborhoods that allow to prune non-improving moves from the neighborhood. Computational results from integrating the resulting local search algorithms into metaheuristics like iterated local search and tabu search, showed promising results for weighted MAX-SAT as well as encoded problems such as set covering and time tabling.

A different extension is the Multi-flip approach by Strohmaier [63] for SAT, where several independent flips, that is, only variables are flipped that do not occur in a same clause, are executed in parallel. The advantage of that approach is that the effect of independent flips is the sum of the single flips. The independent set of flips is determined via a neural network type architecture.

In a similar spirit, Roli [54, 55] proposes to perform flips in parallel without taking into account possible interactions among the variables. More exactly, he divides the variables into k subsets of equal cardinality and for each of the k sets flips the variable having the highest score in the set; the evaluation of a variable flip is done as if all the other variables did not change.

3.5 Non-oblivious local search

To guide the local search typically Equation 3 is used as the evaluation function to rate solutions. Yet, it was shown [1, 42], that by different, so called *non-oblivious* objective functions better worst-case approximation ratios for unweighted MAX-SAT can be guaranteed by a descent local search algorithm. Non-oblivious objective functions for unweighted MAX-SAT weight the objective function according to how many literals are true in a clause. Let S_i be the set of clauses in which exactly i literals are true and $|S_i|$ is the number of clauses in set S_i. Then, for example, the theoretically derived non-oblivious objective function is $3/2 \cdot |S_1| + 2 \cdot |S_2|$.

Battiti and Protasi [5] performed some experiments with *non-oblivious* objective functions for unweighted MAX-SAT. The found that when considering only a pure iterated descent algorithm, also improved average behavior[4] was obtained. Further improvements could be obtained by running an *oblivious* local search (that uses objective function 3 for guidance) after the descent with the non-oblivious objective function. Yet, when using the non-oblivious function in local search algorithms that may escape from local optima, better results were obtained with the standard oblivious objective function. Anyway, with a hybrid approach that used the non-oblivious function in the initial descent phase and from then on a sophisticated tabu search algorithm, they at least obtained better behavior in this initial search phase.

3.6 Other local search algorithms for SAT

Apart from the presented GSAT and WalkSAT algorithms and their variants, numerous other local search algorithms for SAT have been proposed. Here we give a short overview of these algorithms, because these algorithms could be applied to MAX-SAT as well. Nevertheless, we include only such approaches that were tested on native SAT instances and not on MAX-SAT. Metaheuristics for MAX-SAT are discussed in the next section.

One particularly interesting technique that provides the basis for a number of local search algorithms studied in literature is “clause weighting”. The underlying idea is to associate weights to the clauses of the given CNF formula weights that are modified during the local search process [10, 16, 17, 46, 48, 56, 57]. The scoring function, which is used for choosing a variable to be flipped, then tries to maximize the weight of the satisfied clauses. Typically, the weights of the clauses that are unsatisfied at a local minimum are increased by some constant.

In [57] weights are only modified after each GSAT try by increasing the weight of unsatisfied clauses by one. In [48] the breakout method is proposed which adjusts the clause weights during a single try. In particular, each time the algorithm encounters a local minimum, the weights of the currently violated clauses are increased. In [10, 16, 17] different strategies of how to change the weights during a single try are investigated. Computational results are presented in [10] for a class of randomly generated instances which have only one single solution [2]. For the hardest of these instances considerable improvements of the weighting scheme when compared to GSAT and GWSAT have been observed. Yet, the instances used in that comparison are not intrinsically hard because they can be solved by polynomial simplifications (unit propagation and unary / binary failed clauses). Experimental results presented in [16, 17] show that the weighting strategies significantly improve over GSAT and give somewhat better results than HSAT on Random-3-Sat instances. More concretely, using the weighting schemes proposed, the average number of flips needed to find a solution is reduced by ca. 50% when compared to HSAT. More recently, better performance could be obtained with the guided local search approach by Tsang and Mills [46], which also was applied to MAX-SAT and therefore is discussed in the next section, and the weighting scheme implemented by Schurmanns and Southey [56]. Yet, it is not clear how efficient the proposed schemes can be implemented since the weights have to be modified rather frequently and the single search steps are certainly more expensive—in terms of CPU time—than for GSAT.

[4]Note that the theoretical results hold only for the worst-case behavior and therefore this is a non-trivial result.

Many other local search variants which are very popular in the Operations Research community have been applied to SAT. These include methods based on Simulated Annealing [6, 61, 58], Evolutionary Algorithms [27, 14], Ant Colony Optimization [13], and Greedy Randomized Adaptive Search Procedures (GRASP) [52]. Yet, from the data published on the performance of these algorithms, there is no evidence that they might generally perform significantly better for SAT than the best performing algorithms of the GSAT or WSAT architecture. For example, in [6] it has been shown that GSAT performs better than Simulated Annealing on hard Random-3-SAT instances.

4 Metaheuristic approaches to MAX-SAT

In this section we present the available metaheuristic approaches to tackle MAX-SAT. The different approaches are ordered according to the type of metaheuristic applied. Note that in this section, we only refer to approaches that are intended to solve the MAX-SAT problem and not SAT. Additional metaheuristic approaches to SAT were referenced in the previous section. If not mentioned otherwise, the following approaches all use a 1-flip local search neighborhood.

4.1 Approaches based on GSAT or WSAT implementations

Few applications of the GSAT or WSAT variants introduced before were reported for MAX-SAT. Some few results with GWSAT for unweighted MAX-3-SAT instances were given in [58] and some results for SAT-encoded Steiner Tree problems with a WalkSAT variant for weighted MAX-SAT were reported [38].

4.2 Approaches based on Tabu Search

MAX-SAT was one of the first application problems to which (simple) tabu search algorithms were applied [22, 23, 24, 30]. In fact, in one of the papers in which tabu search algorithms were first proposed, in the paper on the steepest ascent, mildest descent (SAMD) approach by Hansen and Jaumard [30], the target application was MAX-SAT. In that paper the SAMD algorithm was shown to be superior to several approximation algorithms by Johnson [39] and simulated annealing on a set of randomly generated, unweighted MAX-SAT instances. In fact, the SAMD algorithm is quite close to the tabu search extension of GSAT.

A second, rather elaborate history-based algorithm is the reactive search by Battiti and Protasi [4]. The main idea underlying the approach is to have a two phase approach consisting of a simple, GSAT-type local search and a tabu search phase, the tabu search algorithm being close to GSAT/TABU. The tabu search phase is run for $2 \cdot (tl + 1)$ iterations, where tl is the tabu tenure. After the tabu search stops, a GSAT local search is executed until the local search is trapped in a local optimum[5] and then the Hamming distance to the starting point is measured.

[5]Let us note that this algorithm can also be considered as an iterated local search algorithm, where the perturbation corresponds to the tabu search phase in this algorithm and the acceptance criterion accepts every new solution.

Based on the resulting distance, the tabu tenure is adjusted (hence, the name reactive search) and again the tabu search phase is initiated. we refer to the original paper for details how this is done.

Apart from this reactive scheme to dynamically adapt the tabu tenure, three more details are interesting in that approach. First, in the initial descent phase, the algorithm uses the non-oblivious objective function, which was presented in Section 3.5, to guide the local search.[6] Second, a particular tie breaking criterion is used: if several variable flips give the same best improvement, variables satisfying more new clauses (that are not satisfied under the current assignment) are preferred. If more than one move satisfies this criterion, a random choice is done. Third, the algorithm restarts from a randomly generated solution after $10n$ steps are executed in the local search.

The reactive search algorithm was tested on randomly generated MAX-3-SAT instances as explained in Section 2. The algorithm was shown to perform very well and seems to be among the currently best performing algorithms for unweighted MAX-SAT instances.

4.3 Greedy Randomized Adaptive Search Procedures

Typically, local search algorithms for SAT and MAX-SAT start from random initial solutions. In the Greedy Randomized Adaptive Search Procedures (GRASP) approach to MAX-SAT by Resende Resende, Pitsoulis, and Pardalos [53, 49] a randomized greedy construction heuristic is used to generate good initial solutions for a subsequent local search phase. The GRASP approach was tested on a small set of weighted MAX-SAT instances with 100 variables (see details on these instances in Section 2) yielding optimal solutions for several instances.[7] Yet, the GRASP approach nowadays is outperformed by many other algorithms for MAX-SAT which are presented next.

4.4 Weighting schemes

Several of the local search algorithms for MAX-SAT make use of clause weighting schemes to improve the search performance. These algorithms actually try to escape from local optima by dynamically changing the evaluation function for rating the truth value assignments. Several such approaches, as described in Section 3.6, have earlier been applied to SAT.

There are currently two prominent approaches the discrete Lagrangian method by Shang and Wah [60] and recent improvements thereof by Wu and Wah [66], which is, as the name suggests, an adaptation of the usual Lagrangian method for continuous optimization. As such it has a more solid mathematical foundation than previous, more ad-hoc weighting schemes mainly applied to SAT. The method consists in iteratively adapting the clause weights after a

[6]Unpublished experiments by some the authors of this report showed that, for the benchmark instances applied by Battiti and Protasi, this seems rather to be an unnecessary complication of the algorithm than a key element to achieve high performance.

[7]Note that when treated as SAT, some of the instances tested are actually satisfiable and the optimal solution has a value corresponding to the sum of the weights of all clauses. For the other instances, optimal solutions were determined by solving them with CPLEX, an integer programming software.

local search was trapped in a local optimum of the evaluation function, for details we refer again to the original papers.

More recently, guided local search (GLS) was applied to weighted MAX-SAT problems by Mills and Tsang [46]. This method differs from the previous one mainly in the way the weight update is done and in the local search algorithm. While the Lagrangian methods use adaptations of tabu search, the GLS approach uses HSAT as the underlying search engine. Whenever the local search is deemed to be trapped in a local optimum, the clause weights are adapted.

Both weighting schemes showed excellent performance on the benchmark instances introduced by Resende, Pitsoulis, and Pardalos [53], outperforming the GRASP approach. Of the two approaches, the GLS appears to have a slightly better performance. Unfortunately, no results of these algorithms for larger instances or unweighted MAX-SAT instances are available.

4.5 Variable Neighborhood Search

Variable neighborhood search (VNS) was recently applied to the (weighted) MAX-SAT problem [31, 32]. Some initial results with a basic VNS algorithms showed comparable performance to a (simple) tabu search algorithm when applied to the MAX-SAT instances by Resende at al. [53] and both algorithms significantly outperformed the GRASP approach described before. To further improve the performance of the VNS algorithm, Hansen et al. introduced an extension of VNS, called *skewed VNS*, and tested it on set of randomly generated instances with real weights. A final comparison among the basic VNS, the tabu search and the skewed VNS showed that the latter gave a significantly better performance and the performance gap between the algorithms increased with instance size.

4.6 Genetic Algorithms

Few applications of genetic algorithms to MAX-SAT exist [50, 51, 7]. It should be noted that it is rather straightforward to apply genetic algorithms to MAX-SAT because the solutions can naturally be represented as binary strings and all the standard crossover and mutation operators working on binary strings can be applied in a straightforward way. Yet, judging from the computational results presented so far, pure genetic algorithms that do not incorporate local search show a relatively poor performance. Nevertheless, some insights into the behavior of genetic algorithms for MAX-SAT have been identified [50, 51].

5 Other algorithmic approaches to MAX-SAT

5.1 “Classical” approximation algorithms

For some approximation algorithms bounds on their worst case behavior can be given. For example, there could be obtained for the non-oblivious local search presented in Section 3.5 non-trivial bounds on the quality of the solutions it returns in the worst case, better than that for oblivious local search. Traditionally, many such approximation algorithms with provable bounds on their worst case behavior are based on constructive algorithms. One of the first examples are the approximation algorithms by Johnson [39] for which it could be proved that it is a $\frac{1}{2}$-approximation algorithm. (We say an algorithm for a maximization problem is an ϵ-approximation algorithm if it produces in polynomial time a solution s such that $f(s) \geq \epsilon \cdot f(s_{opt})$, where s_{opt} is an optimal solution.)

Following these initial results, several approximation algorithms yielding even better worst case behavior improved. These include the algorithms by Yannakakis [69] and Goemans and Williamson [25], which both achieve $\epsilon = \frac{3}{4}$. Slightly improved approximation ratios could be obtained in an algorithm based on semidefinite programming which is again due to Goemans and Williamson [26].

For the special cases of MAX-2-SAT and MAX-3-SAT tighter approximation guarantees can be obtained. For MAX-2-SAT a 0.931-approximation algorithm can be obtained [15, 26], while for MAX-3-SAT an 0.801-approximation algorithm is known [64]. Nevertheless, unless $P = NP$, no approximation algorithm for MAX-SAT will be able to guarantee ϵ-approximations with ϵ arbitrarily close to one. In fact, the strongest negative results on the approximability of MAX-SAT were given by Håstad [33], who proved that MAX-SAT cannot be approximated in polynomial time with a performance ratio greater than $7/8$. When restricted to MAX-2-SAT [33] showed that no performance ratio greater than 0.955 can be obtained.

5.2 Exact algorithms

Most exact algorithms for MAX-SAT are based either on Branch & Bound type extensions of backtracking algorithms derived from the Davis-Logeman-Loveland procedure (DLL) [12] or on integer programming approaches.

Currently the best available exact algorithms for SAT are based on the DLL procedure, a backtracking procedure enhanced by techniques like unit propagation etc. It is rather straightforward to extend this procedure to solve MAX-SAT problems [9, 65]. To do so a lower bound lb on the weight of satisfied clauses is maintained and updated each time a new best solution is found. Additionally one keeps track of the weight uw of the clauses that are unsatisfiable given the current partial assignment. Let W be the sum all all clause weights, that is $W = \sum_{i=1}^{m} w_i$. If we have $W - uw \leq lb$, the current partial solution cannot be extended to a better solution than the incumbent one. In this case one can discard this partial solution and invoke backtracking.

For the application of standard integer programming techniques, the first step is to obtain an integer linear programming (ILP) formulation of MAX-SAT. This can be done in a straightforward way by defining z_i as a binary variable with $z_i = 1$ if clause C_i is satisfied and $z_i = 0$ otherwise. Then, the ILP formulation is

$$\max \sum_{j=1}^{m} w_j \cdot z_j$$

subject to the constraints

$$\sum_{l_{ij} \ is \ positive} x_i + \sum_{l_{ij} \ is \ negative} (1 - x_i) \geq z_j, \ j = 1, \ldots, m$$

$$x_i \in \{0, 1\}, i = 1, \ldots, n$$
$$z_j \in \{0, 1\}, j = 1, \ldots, m$$

z_j occurs only on the right-hand side of one constraint and in the objective function. Therefore, z_j will only be one, if the constraint, that is, the clause is satisfied.

There exist several integer programming approaches to the SAT problem [8, 34, 35] and also for MAX-SAT [40] of which Branch and Cut (B&C) approaches are the most successful.

A comparison of an algorithm based on DPL and a B&C algorithm by Joy, Mitchell and Borchers showed that the DPL-based approach performed significantly better than the B&C algorithm on MAX-3-SAT, while the B&C algorithm was found to be superior on MAX-2-SAT problems and encoded Steiner tree problems. Hence, none of the currently exisiting exact algorithms is dominating over the whole set of benchmark problems.

Finally, it should be mentioned that the weighted MAX-SAT instances by Resende et al. [53] as well as the weighted MAX-SAT instances, which were used to evaluate VNS [31], with up to 500 variables were solved to optimality with CPLEX, a well known integer programming software package. Nevertheless, judging from the computational results reported in the literature, exact algorithms seem to limited (with the notable exception being SAT-encoded Steiner tree problems) to instances with a few hundred variables.

6 Conclusions

In this article we have given an overview of existing algorithmic approaches to tackle the MAX-SAT problem. While this report mainly was describing the state-of-the-art local search routines, we shortly scetched the state-of-the-art for other approaches like approximation algorithms with known worst-case performance guarantees and exact algorithms.

If high quality solutions for large MAX-SAT instances are required, the current state-of-the-art for the MAX-SAT problems suggests that metaheuristic approaches are the currently most efficient solution techniques. Typically they outperform classical approximation algorithms [5] and the application of exact algorithms is limited to rather small instances with at most a few hundred of variables.

The computational results reported in the literature so far do not give a comprehensive picture on which algorithm should be preferred, because in most researches the algorithm designers applied their algorithms only to a specific benchmark set. To anyway give some hint on the relative performance of the algorithms as far as possible, let us state that for the available weighted MAX-SAT instances the currently best performing algorithms seem to be those using clause weighting schemes, while for unweighted MAX-SAT (with high clause variable ratios) so far the best results appear to have been obtained by the reactive search algorithm by Battiti and Protasi [4].

The work in the Metaheuristics Network on MAX-SAT can further progress the state-of-the-art in MAX-SAT solving. First, it might well be that the best implementations developed by the network members will prove to perform better than existing approaches. Second, by the systematic experimental testing, we can establish a more coherent set of benchmark instances. Third, further analysis of the MAX-SAT problem may improve our understanding of how problem characteristics influence metaheuristic performance. Hence, MAX-SAT offers a number of research challenges and we can expect significant results out of the work in the Metaheuristics Network.

Acknowledgments This work was supported 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.

References

[1] P. Alimonti. New local search approximation techniques for maximum generalized satisfiability problems. In *Proceedings of the 2nd Italian Conference on Algorithms and Complexity*, pages 40–53, 1994.

[2] Y. Asahiro, K. Iwama, and E. Miyano. Random generation of test instances with controlled attributes. In D.S.Johnson and M.A.Trick, editors, *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 377–394. American Mathematical Society, 1996.

[3] P. Asirelli, M. de Santis, and A. Martelli. Integrity constraints in logic databases. *Journal of Logic Programming*, 3:221–232, 1985.

[4] R. Battiti and M. Protasi. Reactive search, a history-based heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2, 1997.

[5] R. Battiti and M. Protasi. Solving MAX-SAT with non-oblivious functions and history-based heuristics. 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 649–667. American Mathematical Society, 1997.

[6] A. Beringer, G. Aschemann, H.H. Hoos, M. Metzger, and A. Weiß. GSAT versus simulated annealing. In A.G. Cohn, editor, *Proceedings of the European Conference on Artificial Intelligence*, pages 130–134. John Wiley & Sons, Chichester, UK, 1994.

[7] A. Bertoni, P. Campadelli, M. Carpentieri, and G. Grossi. A genetic model: Analysis and application to MAXSAT. *Evolutionary Computation*, 8(3):291–309, 2000.

[8] C.E. Blair, R.G. Jeroslow, and J.K. Lowe. Some results and experiments in programming techniques for propositional logic. *Computers & Operations Research*, 13(5):633–645, 1986.

[9] 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.

[10] B. Cha and K. Iwama. Performance tests of local search algorithms using new types of random CNF formula. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 304–309. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1995.

[11] Stephen A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–156. Shaker Heights, Ohio, 1971.

[12] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[13] L. Dittmann. Anwendung des Ameisensystem auf das SAT Problem. Master's thesis, Intellectics Group, Darmstadt University of Technology, 1997.

[14] A.E. Eiben and J.K. van der Hauw. Solving 3-SAT with adaptive genetic algorithms. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 81–86. IEEE Press, Piscataway, NJ, USA, 1997.

[15] U. Feige and M. Goemans. Approximating the value of two proper proof systems, with applications to MAX-2SAT and MAX-DICUT. In *Proceedings of the 3rd Israel Symposium on Theory and Computing Systems*, pages 182–189, 1995.

[16] J. Frank. Weighting for Godot: Learning heuristics for GSAT. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 338–343. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1996.

[17] J. Frank. Learning short-term clause weights for GSAT. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 384–389. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.

[18] J. Frank, P. Cheeseman, and J. Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[19] I.P. Gent and T. Walsh. An empirical analysis of search in GSAT. *Journal of Artificial Intelligence Research*, 1:47–59, 1993.

[20] I.P. Gent and T. Walsh. Towards an understanding of hill–climbing procedures for SAT. In *Proceedings of AAAI'93*, pages 28–33. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1993.

[21] I.P. Gent and T. Walsh. Unsatisfied variables in local search. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions*, pages 73–85. IOS Press, Amsterdam, the Netherlands, 1995.

[22] F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[23] F. Glover. Tabu search – part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[24] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, London, 1997.

[25] M. Goemans and D. Williamson. A new $3/4$ approximation algorithm for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666, 1994.

[26] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–11145, 1995.

[27] J. Gottlieb and N. Voss. Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 813–822. Springer Verlag, Berlin, Germany, 1998.

[28] J. Gu, P.W. Purdom, J. Franco, and B.W. Wah. Algorithms for the satisfiability problem: A survey. 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*. American Mathematical Society, 1997.

[29] J. Gu and R. Puri. Asynchronous circuit synthesis with boolean satisfiability. *IEEE Transactions on Computer–Aided Design of Integrated Circuits*, 14(8):961–973, 1995.

[30] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[31] P. Hansen, B. Jaumard, N. Mladenović, and A.D. Parreira. Variable neighbourhood search for maximum weighted satisfiability problem. Technical Report G-2000-62, Les Cahiers du GERAD, Group for Research in Decision Analysis, 2000.

[32] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, Boston, 1999.

[33] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 1–10, 1997.

[34] J.N. Hooker. Resolution vs. cutting plane solution of inference problems: Some computational experience. *Operations Research Letters*, 7(1):1–7, 1988.

[35] J.N. Hooker and C. Fedjki. Branch and cut solution of inference problems in propositional logic. *Annals of Mathematics and AI*, 1:123–139, 1990.

[36] H.H. Hoos. *Stochastic Local Search — Methods, Models, Applications*. PhD thesis, TU Darmstadt, FB Informatik, 1998.

[37] H.H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24:421–481, 2000.

[38] 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.

[39] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Science*, 9:256–278, 1974.

[40] 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.

[41] 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.

[42] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 819–836, Los Angeles, CA, 1994. IEEE Computer Society.

[43] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, 1994.

[44] B. Mazure, L. Sais, and É. Grégoire. Tabu Search for SAT. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 281–285. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1997.

[45] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 321–326. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1997.

[46] 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, Amsterdam, The Netherlands, 2000.

[47] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 459–465. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1992.

[48] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 40–45. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1993.

[49] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP for MAX-SAT problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

[50] S. Rana. *Examining the Role of Local Optima and Schema Processing in Genetic Search*. PhD thesis, Department of Computer Science, Colorado State University, Fort Collins, Colorado, USA, 1999.

[51] S. Rana and D. Whitley. Genetic algorithm behavior in the maxsat domain. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 785–794. Springer Verlag, Berlin, Germany, 1998.

[52] M.G.C. Resende and T.A. Feo. A GRASP for satisfiability. In D.S. Johnson and M.A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 499–520. American Mathematical Society, 1996.

[53] 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 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 393–405. American Mathematical Society, 1997.

[54] A. Roli. Criticality and parallelism in GSAT. In *Working Notes for SAT2001: Workshop on the Satisfiability Testing*, 2001.

[55] A. Roli and C. Blum. Critical parallelization of local search for MAX-SAT. In *Procedings of AI*IA, 7th Congress of the Italian Association of Artificial Itnelligence*, 2001.

[56] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 297–302. AAAI Press / The MIT Press, Menlo Park, CA, USA, 2001.

[57] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 290–295. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.

[58] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 337–343. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1994.

[59] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 440–446. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1992.

[60] Y. Shang and B.W. Wah. Discrete lagrangian-based search for solving MAX-SAT problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, volume 1, pages 378–383. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1997.

[61] W.M. Spears. Simulated annealing for hard satisfiability problems. Technical report, Naval Research Laboratory, Washington D.C., 1993.

[62] O. Steinmann, A. Strohmaier, and T. Stützle. Tabu search vs. random walk. In *Advances in Artificial Intelligence (KI97)*, volume 1303 of *LNAI*, pages 337–348. Springer Verlag, 1997.

[63] A. Strohmaier. Multi-flip networks for SAT. In *Proceedings of KI-98,* Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany, 1998.

[64] L. Trevisan, G.B. Sorkin, M. Sudan, and D.P. Williamson. Gadgets, approximation, and linear programming. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science,* pages 617–626, 1996.

[65] R.J. Wallace and E.C. Freuder. Comparative studies of constraint satisfaction and davis-putnam algorithms for maximum satisfiability problems. In D.S. Johnson and M.A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 587–615. American Mathematical Society, 1996.

[66] Z. Wu and B.W. Wah. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 673–678. AAAI Press / The MIT Press, Menlo Park, CA, USA, 1999.

[67] 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.

[68] M. Yagiura and T. Ibaraki. Analyses on the 2 and 3-flip neighborhoods for the MAX SAT. *Journal of Combinatorial Optimization,* 3:95–114, 1999.

[69] M. Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of the Third ACM–SIAM Symposium on Discrete Algorithms*, pages 1–9, 1992.