Adaptive Memory-Based Local Search for MAX-SAT

Zhipeng Lü[a,b], Jin-Kao Hao[b,*]

Accept to Applied Soft Computing, Feb 2012

[a] *School of Computer Science and Technology, Huazhong University of Science and Technology, 430074 Wuhan, P.R.China*
[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

Abstract

Many real world problems, such as circuit designing and planning, can be encoded into the maximum satisfiability problem (MAX-SAT). To solve MAX-SAT, many effective local search heuristic algorithms have been reported in the literature. This paper aims to study how useful information could be gathered during the search history and used to enhance local search heuristic algorithms. For this purpose, we present an adaptive memory-based local search heuristic (denoted by *AMLS*) for solving MAX-SAT. The *AMLS* algorithm uses several memory structures to define new rules for selecting the next variable to flip at each step and additional adaptive mechanisms and diversification strategies. The effectiveness and efficiency of our *AMLS* algorithm is evaluated on a large range of random and structured MAX-SAT and SAT instances, many of which are derived from real world applications. The computational results show that *AMLS* competes favorably, in terms of several criteria, with four state-of-the-art SAT and MAX-SAT solvers *AdaptNovelty+*, $AdaptG2WSAT_p$, *IRoTS* and *RoTS*.
Keywords: Local Search; Tabu Search; SAT and MAX-SAT; Hybrid Algorithm; Metaheuristics

1. Introduction

As one of the most studied NP-complete problems, the propositional satisfiability problem or SAT has deserved numerous studies in the last few

*Corresponding author.
Email addresses: zhipeng.lv@hust.edu.cn, zhipeng.lui@gmail.com (Zhipeng Lü), hao@info.univ-angers.fr (Jin-Kao Hao)

decades. Besides its theoretical importance, SAT has many practical applications such as circuit designing, planning or graph coloring, since such problems can be conveniently formulated with SAT in a natural way [7].

A SAT instance $\mathcal{F}$ is typically defined by a set of n Boolean variables and a conjunctive normal form (CNF) of a set of m disjunctive clauses of literals, where each literal is a variable or its negation. The SAT problem consists in deciding whether there exists an assignment of truth values to variables such that all clauses in $\mathcal{F}$ can be satisfied.

MAX-SAT is the optimization variant of SAT in which the objective is to find an assignment of truth values to the variables in $\mathcal{F}$ that minimizes the number of *unsatisfied* clauses or equivalently to find an assignment that maximizes the number of satisfied clauses. In weighted MAX-SAT, each clause c_i is associated with a weight w_i and the objective is to minimize the total weight of the unsatisfied clauses. Obliviously, SAT can be considered as a special case of MAX-SAT and the latter is a special case of weighted MAX-SAT where each clause weight equals to one.

Given a CNF formula $\mathcal{F}$ and an initial assignment, local search procedures repeatedly modify locally this assignment, typically by flipping each time one variable, in order to find an assignment having as large as possible weights of satisfied clauses of $\mathcal{F}$. Since the introduction of *GSAT* [23] and *WalkSAT* [22], there have been a large number of local search heuristics proposed to tackle the SAT and MAX-SAT problems. These heuristics mainly differ from each other on the variable selection heuristics used at each local search iteration.

Besides historically important *GSAT* and *WalkSAT*, other representative state-of-the-art local search algorithms in the literature include various enhanced *GSAT* and *WalkSAT* solvers (e.g., *GSAT/Tabu* [18], *WalkSAT/Tabu* [19], *Novelty* and *R_Novelty* [19], *G2WSAT* [14]), adaptive solvers based on dynamic noise tuning (e.g., *AdaptNovelty+* [9], $AdaptG2WSAT_P$ [16] and *TNM* [15]), variable weighting algorithms (e.g., *VW* [21]), clause weighting algorithms (e.g., Pure Additive Weighting Scheme (*PAWS*) [27], Scaling and Probabilistic Smoothing (*SAPS*) [10] and Discrete Lagrangian Method (*DLM*) [31]), genetic algorithm [12], hybrid algorithms (e.g., *GASAT* [13] and *Hybrid* [30]) and other SAT or CSP solvers [4, 6, 11, 17, 24, 29]. Interested readers are referred to [7] for more details.

However, no single local search heuristic can be effective on all types of instances, since each type of instances presents certain characteristics. One way to design an effective heuristic algorithm is to take advantage of various memory information gathered during the search process to guide the algorithm into promising search regions and diversify the search when necessary. Furthermore, it is also essential to adapt the search to switch smoothly between intensification and diversification according to the search history. Therefore, this paper aims to study how useful information collected during the search history could be used to enhance the performance of local search heuristic algorithms.

Following this spirit, we propose a memory-based local search heuristic (denoted by *AMLS*) for MAX-SAT and SAT. In order to achieve a suitable tradeoff between intensification and diversification, our variable selection heuristic is based on various memory structures. In addition, the diversification parameters used in the algorithm are dynamically adjusted according to the search history.

In addition to taking advantage of some well known strategies in the literature, our *AMLS* introduces some original features:

- In the variable selection heuristic, our *AMLS* algorithm globally takes into account information related to tabu mechanism, variable flipping recency, and consecutive falsification and satisfaction of clauses, which is missing in the literature.

- To refine our variable selection rule, we introduce a penalty function which is guided by clause falsification and satisfaction information.

- Our *AMLS* algorithm employs a dynamic tabu tenure strategy. Additionally it uses an aspiration criterion to choose a tabu variable under specific conditions.

- We adopt the Hoos's adaptive mechanism [9] to dynamically adjust both diversification parameters p and wp, while this mechanism is previously used to adjust p only.

- We employ an adaptive random perturbation operator to diversify the search when the search reaches a local optimum solution.

Our experimental results show that, on a broad range of MAX-SAT and SAT instances, many of which are derived from real word application, *AMLS* compares favorably with the state-of-the-art local search algorithms, such as *AdaptNovelty+*, adaptive gradient-based WalkSAT algorithm with promising decreasing variable heuristics ($AdaptG2WSAT_p$), Robust Tabu Search (*RoTS*) and Iterated Robust Tabu Search (*IRoTS*). Furthermore, without any manual parameter tuning, *AMLS* solves effectively these instances in a reasonable time.

The remaining part of the paper is organized as follows. In Section 2, we briefly review some previous related works in the literature. Then, Section 3 presents our adaptive memory based local search algorithm. Section 4 is dedicated to computational results. Discussion and remarks are presented in Section 5 and conclusions are given in Section 6.

2. Related Works

In spite of the differences between our algorithm and existing SAT and MAX-SAT solvers, *AMLS* inherits some elite features of previous algorithms. We now review some of these related works and the way of adopting these features into our *AMLS* algorithm.

For the incomplete SAT solvers developed during the last two decades, the most significant advancement was perhaps to introduce "random walk" component into the local search procedures, leading to the well-known *GSAT* with random walk [23] and *WalkSAT* [22]. At each step, *GSAT* greedily chooses a best variable to flip among *all the variables* that occur in at least one unsatisfied clause, while *WalkSAT* first randomly selects *one unsatisfied clause* and then always picks a variable from the selected clause to flip. Our *AMLS* algorithm attempts to incorporate both the intensification of *GSAT* to select the best variable to flip and the diversification of *WalkSAT* to always pick a variable from one random unsatisfied clause to flip.

One direct improvement on *GSAT* and *WalkSAT* is to extend them into a simple tabu search strategy. *GSAT/Tabu* is obtained from *GSAT* by associating a tabu status with the variables [18]. Similar to *GSAT/Tabu*, there is also an extension to *WalkSAT* which employs a tabu mechanism, called *WalkSAT/Tabu* [19]. For both algorithms, each flipped variable is enforced with a tabu tenure to avoid repeating previous recent moves. Our *AMLS* algorithm then adopts the tabu table to diversify the search when we choose the best variable to flip as *GSAT*.

McAllester, Selman and Kautz (1997) improved the *WalkSAT* algorithm by introducing the following two improvements for selecting the variable to flip: Favoring the best variable that is not recently flipped and selecting the second best variable according to certain probability p. The probability p is called the *noise parameter*. These strategies lead to the famous *Novelty* and *R_Novelty* heuristics. Our *AMLS* algorithm then borrows the idea of strategically selecting the second best variable to flip to further enhance its diversification capability.

For *WalkSAT*-based algorithms, the choice of the noise parameter p has a major impact on the performance of the respective algorithm. However, finding the optimal noise setting is extremely difficult. Hoos proposed an adaptive mechanism to adjust the noise parameter p according to the search history [9]. The main idea is to increase the noise value when the search procedure detects a stagnation behavior. Then, the noise is gradually decreased until the next stagnation situation is detected. This adaptive mechanism leads to various adaptive algorithms, such as *AdaptNovelty+* [9] and *AdaptG2WSAT*$_P$ [16]. Li and Wei (2009) proposed another noise adaptive mechanism relying on the history of the most recent consecutive falsification of a clause. The main principle is that if a clause is most recently falsified by a same variable for a number of consecutive times, then the noise level should be increased [15]. We adapt this adaptive mechanism into our *AMLS* algorithm to enhance its robustness and utilize the information of the recent consecutive falsification of a clause to strategically select the second best variable to flip.

Hoos introduced random walk into the *Novelty* and *R_Novelty* heuristics to prevent the extreme stagnation behavior, leading to the *Novelty+* and *R_Novelty+* heuristics [8]. These variants perform a random walk with a small probability wp, and select the variable to be flipped according to the standard *Novelty* and *R_Novelty* mechanisms with probability $1 - wp$. We adopt this famous "random walk" strategy as one of the main mechanisms to diversify the search in our *AMLS* algorithm.

Battiti and Potasi proposed a Reactive Tabu Search (H-RTS) algorithm for unweighted MAX-SAT, in which the tabu tenure is dynamically adjusted during the search [1]. Also noteworthy is an Iterated Local Search by Yagiura and Ibaraki [32] that is different from many other approaches in that 2-flip and 3-flip neighborhoods are used. The perturbation phase consists of a fixed number of random walk steps. In [25], a new stochastic local search algorithm, called Iterated Robust Tabu Search (IRoTS), was presented for MAX-SAT that combines two well-known metaheuristic approaches: Iterated Local Search and Tabu Search. IRoTS used a different perturbation mechanism, in which a number of RoTS steps are performed with tabu tenure values that are substantially higher than the ones used in the local search phase. For strong diversification purpose, our *AMLS* algorithm employs a perturbation strategy to diversify the search in a drastic way.

3. Adaptive Memory-Based Local Search

3.1. Evaluation Function and Neighborhood Moves

Given a CNF formula $\mathcal{F}$ and an assignment A, the evaluation function $f(A)$, is the total weight of unsatisfied clauses in $\mathcal{F}$ under A. This function is to be minimized with zero as its optimal value, if possible. Our *AMLS* algorithm is based on the widely used *one-flip* move *neighborhood* defined on the set of variables contained in at least one unsatisfied clause (*critical variables*).

More formally, the most obvious way to represent a solution for a MAX-SAT instance with n variables is to use a n-binary vector $A = [a_1, a_2, \ldots, a_n]$. $A|i$ denotes the truth value of the i^{th} variable a_i. $A[i \leftarrow \alpha]$ denotes an assignment A where the i^{th} variable has been set to the value α. Given a clause c, we use $sat(A, c)$ to denote the fact that the assignment A satisfies the clause c. Each clause c is associated with a weight w_c. For unweighted problems, $w_c = 1$ for all the clauses. We use $i \prec c$ to represent that variable a_i or its negation $\neg a_i$ appears in clause c. Therefore, the evaluation function $f(A)$ can be writhen as:

$$f(A) = \sum_{c \in \mathcal{F}} \{w_c | \neg sat(A, c)\} \tag{1}$$

In a local search procedure, applying a move mv to a candidate solution A leads to a new solution denoted by $A \oplus mv$. Let $M(A)$ be the set of all possible moves which can be applied to A, then the neighborhood of A is defined by: $N(A) = \{A \oplus mv | mv \in M(A)\}$. For MAX-SAT, we use the *critical one-flip* neighborhood defined by flipping one *critical variable* at a time. A move that flips the truth assignment of the i^{th} variable a_i from $A|i$ to 1-$A|i$ is denoted by $mv(i)$. Thus,

$$M(A) = \{mv(i) | \exists c \in \mathcal{F}, i \prec c \wedge \neg sat(A, c), i = 1, \ldots, n\} \tag{2}$$

3.2. Main Scheme and Basic Preliminaries

The general architecture of the proposed *AMLS* procedure is described in Algorithm 1. From an initial truth assignment A of variables, *AMLS* optimizes the evaluation function $f(A)$ by minimizing the total weight of unsatisfied clauses.

At each search step, a key decision is the selection of the next variable to be flipped (see line 8 of Algorithm 1). This memory-based variable selection heuristic is explained later in Algorithm 2. *AMLS* uses two diversification parameters (p and wp). The adaptive tunings of both parameters (line 14) and the perturbation operator (line 16) are described in later sections.

As a basis for the following explanation, we first explain how neighboring solutions are evaluated. Indeed, for large problem instances, it is necessary to be able to rapidly determine the effect of a move on $f(A)$. In our implementation, since at each iteration we examine all the *critical* neighborhood

Algorithm 1 *AMLS* procedure for MAX-SAT

Input: CNF formula $\mathcal{F}$, $Maxpert$ and $Maxsteps$
Output: the best truth assignment A^* and $f(A^*)$
$A \leftarrow$ randomly generated truth assignment
$A^* \leftarrow A$
for $try := 1$ to $Maxpert$ **do**
  Parameter Initialization: $p := 0$, $wp := 0$
  for $step := 1$ to $Maxsteps$ **do**
    $mv(y) \leftarrow$ neighborhood move selected from $M(A)$ (see Algorithm 2)
    $A := A \oplus mv(y)$
    if $f(A^*) < f(A)$ **then**
      $A^* \leftarrow A$
    end if
    Set the tabu tenure of variable a_y
    Update the diversification parameters p and wp
  end for
  $A \leftarrow$ Perturbation Operator(A^*) (see Section 3.5)
end for

moves, an incremental evaluation technique widely used in the family of *GSAT* algorithms [7] is employed to cache and update the variable scores. This forms the basis for choosing the variable to be flipped at each search step.

Specifically, let y be the index of a variable to be flipped. The break of y, $break(y)$, denotes the total weight of clauses in $\mathcal{F}$ that are currently satisfied but will be unsatisfied if variable y is flipped. The make of y, $make(y)$, denotes the total weight of clauses in $\mathcal{F}$ that are currently unsatisfied but will be satisfied if variable y is flipped. The score of y with respect to A, $score(y)$, is the difference between $break(y)$ and $make(y)$. Formally,

$$break(y) = \sum_{c \in \mathcal{F}} \{w_c | sat(A, c) \wedge \neg sat(A \oplus mv(y), c)\} \quad (3)$$

$$make(y) = \sum_{c \in \mathcal{F}} \{w_c | \neg sat(A, c) \wedge sat(A \oplus mv(y), c)\} \quad (4)$$

$$score(y) = break(y) - make(y) \quad (5)$$

In our implementation, we employ a fast incremental evaluation technique to calculate the move value of transitioning to each neighboring solution. Specifically, at each search step only the *break* and *make* values affected by the current move (i.e., the variables that appear in the same clauses with the currently flipped variable) are updated.

Algorithm 2 Variable selection heuristic for *AMLS*

1: **Input**: A and $N(A)$
2: **Output**: The selected variable y to be flipped
3: // Intensification Phase: lines 8-13, 22
4: // Diversification Phase: lines 14-21
5: Let $N_{ts}(A)$ denote the set of neighborhood moves that are tabu
6: Let x_{tb} be the best variable in $N_{ts}(A)$ (tabu moves) in terms of the *score* value
7: Let x_{nb} and x_{nsb}, respectively, be the *best* and the *second best* variables in $N(A) \backslash N_{ts}(A)$ (non-tabu variable moves) in terms of the *score* value
8: **if** $score(x_{tb}) < score(x_{nb})$ and $f(A) + score(x_{tb}) < f(A^*)$ **then**
9: $\quad y := x_{tb}$; **return** y
10: **end if**
11: **if** $score(x_{nb}) < 0$ **then**
12: $\quad y := x_{nb}$; **return** y
13: **end if**
14: **if** $rand[0,1] < wp$ **then**
15: $\quad y :=$ random walk move selected from $N(A) \backslash N_{ts}(A)$; **return** y
16: **end if**
17: **if** x_{nb} is the *least recently* flipped variable in $N(A) \backslash N_{ts}(A)$ and $rand[0,1] < p$ **then**
18: $\quad$ **if** $penalty(x_{nsb}) < penalty(x_{nb})$ **then**
19: $\quad\quad y := x_{nsb}$; **return** y
20: $\quad$ **end if**
21: **end if**
22: $y := x_{nb}$; **return** y

In order to enhance the search capability of our algorithm, we introduce a variable selection heuristic which relies on a set of memory components, including a tabu table to avoid selecting the recently flipped variables, a flip recency structure to record the iteration at which a variable is recently flipped and two other memory structures to respectively record the frequency of a variable that recently consecutively falsifies and satisfies a clause. All these memory structures are jointly used by our variable selection heuristic, as described in Algorithm 2.

Tabu Search (TS) typically incorporates a *tabu list* as a "recency-based" memory structure to assure that solutions visited within a certain span of iterations, called the tabu tenure, will not be revisited [5]. Our *AMLS* algorithm uses such a tabu list as one of its three diversification strategies. In our implementation, each time a variable y is flipped, a value is assigned to an associated record $TabuTenure(y)$ (identifying the "tabu tenure" of

y) to prevent y from being flipped again for the next $TabuTenure(y)$ iterations. For our experiments, we set the tabu tenure in two ways according to whether the instance is satisfiable or not. Specifically, if the instance is unsatisfiable, we set:

$$TabuTenure(y) = tl + rand(15) \tag{6}$$

If the instance is satisfiable, we set:

$$TabuTenure(y) = \lfloor tp \cdot |N(A)| \rfloor + rand(15) \tag{7}$$

where tl is a given constant and $rand(15)$ takes a random value from 1 to 15. $|N(A)|$ is the cardinality of the current neighborhood $N(A)$ and tp is a tabu tenure parameter. We empirically set tp to 0.25 in all the experiments in this paper and we observe that $tp \in [0.15, 0.35]$ gives satisfying results on a large number of instances.

One observes that for satisfiable instances at the beginning of the search, the cardinality of the current neighborhood $|N(A)|$ is large enough and the second part of the tabu tenure function becomes dominated by the first part. On the other hand, as the algorithm progresses, the size of the current neighborhood becomes smaller and smaller. In this situation, since the search is around the local optimum regions, we mainly use a random tabu tenure (the second part) to enlarge the diversification of the search. This is quite different from the situation for the unsatisfiable instances.

For convenience, we use $N_{ts}(A)$ to represent the subset of the current neighborhood that are declared tabu. *AMLS* then restricts consideration to variables in $N(A) \backslash N_{ts}(A)$ (i.e., moves that are not currently tabu). However, an *aspiration criterion* is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than both the current best non-tabu move (x_{nb}) and the best solution found so far, as shown in lines 8-10 of Algorithm 2. Note that in the case that two or more tabu moves have the same score value, we break ties in two ways: one is to favor the least recently flipped variable; the other is to select a variable randomly. This also applies to the situation of identifying the *best* and the *second best* (x_{nsb}) non-tabu moves as following. These two tie-breaking options lead to two versions of our *AMLS* algorithm, denoted by *AMLS1* and *AMLS2* respectively.

Under the condition that the aspiration criterion is not satisfied, if the best non-tabu move (variable) in the current neighborhood can improve the current solution (i.e., $score(x_{nb}) < 0$), our algorithm deterministically selects the best non-tabu move as the variable to be flipped, as described in lines 11-13 of Algorithm 2. These two strategies constitute the intensification phase of our *AMLS* algorithm. These intensification strategies also guarantee that new better solutions would not be missed if such solutions exist in the current neighborhood.

Additionally, our *AMLS* algorithm uses two more strategies to diversify the search when *improving* move (i.e., the move that can improve the current solution) does not exist. In this case, our *AMLS* algorithm randomly selects a variable in the current neighborhood to flip with a small probability wp, as shown in lines 14-16 of Algorithm 2. wp is a diversification parameter just as in *Novelty+* algorithm, which lies in $[0, 0.05]$ and is adaptively adjusted during the search. One notices that this strategy is somewhat different from the "random walk" strategy used in the *WalkSAT* family algorithms, since we consider all the non-tabu *critical variables* while the *WalkSAT* family algorithms always consider the variables in a randomly selected clause.

Our last diversification mechanism is to strategically select the second best non-tabu move according to the search history. This selection is only active when the best non-tabu variable is the recently flipped variable in the current non-tabu neighborhood (line 17 in Algorithm 2), evaluated by the memory structure called *recency* which represents when a variable is most recently flipped. In our implementation, each time a variable y is flipped, the current iteration index (the *step* number in Algorithm 1) is assigned to an associated record $recency(y)$.

Then, we compare the *best* and the *second best* non-tabu variables (denoted by x_{nb} and x_{nsb} respectively) according to a penalty value. The basic idea is that if a clause is most recently falsified (or satisfied) by the same variable for a number of consecutive times, this variable receives a high penalty if its flipping falsifies (or satisfies) again the clause. We denote this kind of penalty cost by $penalty(\cdot)$. Then, if $penalty(x_{nsb})$ is smaller (better) than $penalty(x_{nb})$, the second best non-tabu variable x_{nsb} is selected with a probability p, as shown in lines 17-21 of Algorithm 2. Just like wp mentioned above, the parameter p lies in $[0, 1]$ and is also dynamically adjusted during the search.

Now, we give the details for calculating the $penalty(\cdot)$ value. For this purpose, we give some basic definitions. During the search, for a clause c we respectively record the variable $v_f[c]$ that most recently *falsifies* the clause and the variable $v_s[c]$ that most recently *satisfies* c. A variable *falsifies* a clause means that flipping the variable makes the clause from satisfied to unsatisfied, while a variable *satisfies* a clause implies that the clause turns from unsatisfied to satisfied after the variable is flipped. Meanwhile, the value $n_f[c]$ (respectively $n_s[c]$) records the consecutive times that variable

$v_f[c]$ *falsifies* (respectively $v_s[c]$ *satisfies*) clause c.

At the beginning of the search, for each clause c we initialize $v_f[c]$ to be *null*. Once a clause c is falsified by flipping a variable y, we check whether $v_f[c]$ and y are the same variable. If yes, variable y falsifies clause c again, and thus we increase $n_f[c]$ by 1. Otherwise, variable y is a new variable that falsifies clause c and we set $v_f[c] = y$ and $n_f[c] = 1$. The values of $v_s[c]$ and $n_s[c]$ are likewise updated based on the consecutive satisfaction history of clause c.

At each step for a variable y to be flipped, we define the following two sets:

$RS[y] = \{c \in \mathcal{F} | y$ *satisfies* clause c and $y = v_s[c]\}$

$RF[y] = \{c \in \mathcal{F} | y$ *falsifies* clause c and $y = v_f[c]\}$

$RS[y]$ and $RF[y]$ respectively denote the set of clauses which have been recently consecutively *satisfied* and *falsified* by variable y.

Thus, the penalty of variable y is defined as:

$$penalty(y) = \frac{\sum_{c \in RS[y]} 2^{n_s[c]}}{2|RS[y]|} + \frac{\sum_{c \in RF[y]} 2^{n_f[c]}}{2|RF[y]|} \quad (8)$$

This penalty function measures the degree to which the flipping of variable y can repeat the most recent satisfaction and falsification of a clause on average. Note that if the set $RS[y]$ or $RF[y]$ is empty, the corresponding part in Eq.(8) is set to zero, implying that the flipping of variable y will not repeat the most recent satisfaction or falsification of any clause.

This penalty function is different from the information used in the *Novelty* family algorithms, where only the recency information is used to guide the selection of the second best variable. Furthermore, this strategy is also different from the look-ahead strategy used in *AdaptG2WSAT*$_P$ [16], which selects the second best variable according to the promising scores made by a *two-flip* move. In *TNM* [15], the consecutive falsification information of clauses is also used. However, *TNM* only considers the falsification information on the currently selected clause and the satisfaction information of clauses are not taken into account.

3.4. Dynamic Parameters Adjustment

Previous research has demonstrated that it is highly desirable to have a mechanism that automatically adjusts the noise parameter such that a near optimal performance can be achieved. One of the most effective techniques is the adaptive noise mechanism proposed in [9] for the *WalkSAT* family algorithms, leading to various algorithms, such as *AdaptNovelty+* [9] and

AdaptG2WSAT$_P$ [16]. We extend this adaptive mechanism to adjust the two diversification parameters in our algorithm.

According to this mechanism, the noise parameters are first set at a level low enough such that the objective function value can be quickly improved. Once the search process detects a stagnation situation, the noise level is increased to reinforce the diversification until the search process overcomes the stagnation. Meanwhile, the noise is gradually decreased when the search begins to improve the objective value.

In our *AMLS* algorithm, there are two diversification parameters wp and p that can be adjusted. One observes that the larger the values of wp and p are, the higher possibility that the search can be diversified. Specifically, we record at each adaptive step the current iteration number and the objective value of the current solution. Then, if one observes that this objective value has not been improved over the last $m/6$ steps, where m is the number of clauses of the given problem instance, the search is supposed to be stagnating. At this point, the two parameters are *increased* according to: $wp := wp + (0.05 - wp)/5$ and $p := p + (1 - p)/5$. Similarly, both parameter values are kept until another stagnation situation is detected or the objective value is improved, in the latter case, both parameters are *decreased*: $wp := wp - wp/10$ and $p := p - p/10$. These constant values are borrowed from [9] and have shown to be quite effective for all the tested instances in this paper.

3.5. Perturbation Operator

When the best solution cannot be further improved using the local search algorithm presented above, we employ a simple perturbation operator to locally perturb the local optimum solution and then another round of the local search procedure restarts from the perturbed solution. In order to guide efficiently the search to jump out of the local optima and lead the search procedure to a new promising region, we reconstruct the local optimum solution obtained during the current round of local search as follows.

Our perturbation operator consists of a sequential series of perturbation steps. At each perturbation step, we first order the score values of all the neighborhood moves in $N(A)$ and then randomly choose one among the μ best moves in terms of score values. After flipping the chosen variable, all the affected move values are updated accordingly and the chosen variable is declared tabu. Note that here the tabu tenure is much longer than the case in the local search procedure. We set it to be a random value from $Maxsteps/4$ to $Maxsteps/3$. This strategy is aimed to prevent the search from repeating the perturbation moves at the beginning of the next round of local search.

We repeat the above-mentioned perturbation moves for a given number λ of times, which is also called the perturbation strength. Finally, a new round of local search procedure is launched from the perturbed solution with the flipped variables having longer tabu tenure than the usual local search. This idea is similar to that used in *IRoTS* algorithm [25]. It should be noticed that once a variable is flipped, it is strictly restricted to be flipped again during the current perturbation phase.

4. Experimental Results

4.1. Reference Algorithms and Experimental Protocol

In order to evaluate the relative effectiveness and efficiency of our proposed algorithm, we compared our *AMLS* algorithm with 4 effective MAX-SAT and SAT solvers in the literature:

- *AdaptNovelty+*: Stochastic local search algorithm with an adaptive noise mechanism in [9].
- *AdaptG2WSAT$_p$*: Adaptive gradient-based greedy *WalkSAT* algorithm with promising decreasing variable heuristic in [16].
- *IRoTS*: Iterated robust tabu search algorithm in [25].
- *RoTS*: Robust tabu search algorithm in [26].

We should notice that these 4 reference algorithms are among the most successful approaches for solving a wide range of MAX-SAT and SAT benchmark instances in the literature. For these reference algorithms, we carry out the experiments using UBCSAT (version 1.1)—an implementation and experimentation environment for stochastic local search algorithms for SAT and MAX-SAT solvers—which is downloadable at the webpage[1][28]. As claimed by Tompkins and Hoos [28], the implementations of these reference algorithms in UBCSAT is more efficient than (or just as efficient as) the original implementations. Therefore, a comparison of our algorithm with the UBCSAT implementations is meaningful.

Our algorithm is coded in C and compiled using GNU GCC on a Cluster running Linux with Pentium 2.88GHz CPU and 2GB RAM. Table 1 gives

[1] http://www.satlib.org/ubcsat/index.html

the descriptions and settings of the parameters used in our *AMLS* algorithm for the experiments.

Given the stochastic nature of the proposed algorithm as well as the reference algorithms, each problem instance is independently solved 20 and 100 times respectively for the unsatisfiable and satisfiable instances. Notice that to solve each problem instance, our algorithm and the four reference algorithms are given the same amount of computing effort on the same computer.

As indicated in Section 3.3, we break ties in two ways when we select the best (tabu and non-tabu) and the second best variables: by favoring the least recently flipped variable and selecting a variable randomly. Thus, we denote our *AMLS* algorithm using these two ways of breaking ties by *AMLS1* and *AMLS2*, respectively.

In this paper, we use the total number of iterations as the stop condition of all these 6 algorithms, i.e., the value of $Maxpert \times Maxsteps$ in Algorithm 1.

4.2. Test Instances

To evaluate our *AMLS* algorithm for the MAX-SAT problem, we have tested *AMLS* on 79 well-known benchmark instances. Some of them are derived from other specific problems while others are randomly generated. In some circumstances, it is necessary for each clause to have a weight such that the objective is to maximize the total weights of the satisfied clauses. These 79 benchmark instances belong to four families:

- *UnWeighted2/3MaxSat*. This set contains 13 randomly generated *unweighted MAX-2-SAT* and *MAX-3-SAT* instances first described in [2][2]. The stop condition for this set of instances is 10^6 iterations.
- *WeightedMaxSat*. These 10 instances are *weighted* variants of randomly generated instances used in [32][3]. All of them have 1000 variables and 11050 clauses. The stop condition for this set of instances is 10^8 iterations.
- *UnWeightedMaxSat*. This set of instances consists of 27 structured *unweighted MAX-SAT* instances presented at the SAT2002 or SAT2003 competitions which are available at the web site[4]. These instances are

[2]http://infohost.nmt.edu/~borchers/maxsat.html
[3]http://www.hirata.nuee.nagoya-u.ac.jp/~yagiura/msat.html
[4]http://www.info.univ-angers.fr/pub/lardeux/SAT/benchmarks-EN.html

Table 1: Settings of important parameters

Parameters	Section	Description	Value
$Maxpert$	3.2	number of perturbation phases	100
tl	3.3	tabu tenure constant	15
tp	3.3	tabu tenure parameter	0.25
μ	3.5	number of candidate perturbation moves	15
λ	3.5	perturbation strength	r[20, 30]

also tested in [3] and [13]. The stop condition for this set of instances is 10^7 iterations.

- *Satisfiable*. This set of instances consists of 29 *hard satisfiable* both randomly generated and structured instances used in the DIMACS Challenge which have been widely used by many SAT solvers and are available in SATLIB[5]. The stop condition for this set of instances is set according to the size and difficulty of the tested instances.

4.3. Computational Results

We first present in Table 2 the computational results of the two versions of *AMLS* (*AMLS1* and *AMLS2*) on the 13 *UnWeighted2/3MaxSat* instances, compared with the 4 reference algorithms.

In Table 2 the first two columns identify the problem instance and the best known objective values f^*. From column 3, each three columns give the results of one of the tested algorithms according to three criteria: (1) the success rate, *#suc*, to the best known objective values over 20 runs, (2) the average objective value, f_{avr}, over 20 independent runs and (3) the average search step, *#step*, for reaching the best known result f^*. Notice that these criteria have been widely used for efficiency evaluation of heuristic algorithms. If the algorithms lead to different results, the best results are indicated in bold.

Table 2 shows that the two versions of our *AMLS* algorithm can easily reach the best known objective values with a 100% success rate within the given stop condition for all the considered instances, equalling that of *IRoTS* and *RoTS*. One observes that the algorithms *AdaptNovelty+* and *AdaptG2WSAT*$_p$ cannot always find the best known objective values for 4 and 2 instances, respectively. In particular, for the instance p2600/150 *AdaptNovelty+* cannot find the best known objective value 38 under this stop condition and *AdaptG2WSAT*$_p$ can only reach this objective value for

[5]http://www.satlib.org/benchm.html

Table 2: Computational results on the *UnWeighted2/3MaxSat* instances (stop condition: 10^6)

Instance	f^*	*AMLS1*			*AMLS2*			*AdaptNovelty+*		
		#*suc*	f_{avr}	#*step*	#*suc*	f_{avr}	#*step*	#*suc*	f_{avr}	#*step*
p2200/100	5	20	5.0	222	20	5.0	417	20	5.0	478
p2300/100	15	20	15.0	220	20	15.0	152	20	15.0	817
p2400/100	29	**20**	**29.0**	3083	**20**	**29.0**	2006	10	29.5	354426
p2500/100	44	**20**	**44.0**	359	**20**	**44.0**	292	5	44.8	358968
p2600/100	65	**20**	**65.0**	369	**20**	**65.0**	299	1	66.9	387088
p3500/100	4	20	4.0	43051	20	4.0	14771	20	4.0	2646
p3550/100	5	20	5.0	6272	20	5.0	7784	20	5.0	3795
p3600/100	8	20	8.0	4368	20	8.0	4829	20	8.0	8299
p2300/150	4	20	4.0	**208**	20	4.0	210	20	4.0	237
p2450/150	22	20	22.0	**354**	20	22.0	398	20	22.0	35985
p2600/150	38	**20**	**38.0**	5188	**20**	**38.0**	15283	0	39.2	—
p3675/150	2	20	2.0	24153	20	2.0	20558	20	2.0	7909
p3750/150	5	20	5.0	41177	20	5.0	11906	20	5.0	8137

Instance	f^*	*AdaptG2WSAT*$_p$			*IRoTS*			*RoTS*		
		#*suc*	f_{avr}	#*step*	#*suc*	f_{avr}	#*step*	#*suc*	f_{avr}	#*step*
p2200/100	5	20	5.0	**150**	20	5.0	672	20	5.0	920
p2300/100	15	20	15.0	175	20	15.0	**127**	20	15.0	200
p2400/100	29	16	29.2	43585	**20**	**29.0**	**505**	**20**	**29.0**	513
p2500/100	44	**20**	**44.0**	8885	**20**	**44.0**	**125**	**20**	**44.0**	141
p2600/100	65	**20**	**65.0**	21354	**20**	**65.0**	158	**20**	**65.0**	**151**
p3500/100	4	20	4.0	1952	20	4.0	**1939**	20	4.0	2314
p3550/100	5	20	5.0	1561	20	5.0	**2230**	20	5.0	3775
p3600/100	8	20	8.0	3861	20	8.0	**1817**	20	8.0	2053
p2300/150	4	20	4.0	216	20	4.0	226	20	4.0	263
p2450/150	22	20	22.0	3917	20	22.0	374	20	22.0	580
p2600/150	38	2	38.9	42976	**20**	**38.0**	**1807**	**20**	**38.0**	3564
p3675/150	2	20	2.0	6705	20	2.0	**5674**	20	2.0	22831
p3750/150	5	20	5.0	5237	20	5.0	**4737**	20	5.0	9262

Table 3: Computational results on the 10 random *WeightedMaxSat* instances (stop condition: 10^8)

Instance	*AMLS1*			*AMLS2*			*AdaptNovelty+*		
	f_{best}	f_{avr}	t_{avr}	f_{best}	f_{avr}	t_{avr}	f_{best}	f_{avr}	t_{avr}
Randwb01	9309	9922.9	352.6	9046	10023.9	333.7	**8617**	9293.1	138.3
Randwb02	7992	8665.9	385.1	7996	8665.8	340.9	**7982**	8697.0	139.9
Randwb03	**9011**	9660.8	149.2	8838	9578.9	379.8	9030	9327.1	147.8
Randwb04	**7098**	7628.7	122.9	**7098**	7607.4	311.1	7374	7840.8	189.4
Randwb05	9089	10173.2	238.3	9484	10177.6	346.6	**9070**	10309.1	239.5
Randwb06	9558	10296.9	151.6	9645	10524.2	518.9	**9176**	9744.5	342.5
Randwb07	**9387**	10514.2	438.5	9523	10522.6	111.7	9472	9923.5	264.7
Randwb08	**8228**	8969.9	441.2	**8228**	9019.3	353.3	8242	8980.5	289.5
Randwb09	10274	10818.9	269.4	**10032**	10971.4	525.7	10303	11031.0	225.6
Randwb10	10028	10560.0	54.0	10001	10543.2	485.0	**9903**	10264.0	243.4
Instance	*G2WSAT**			*IRoTS*			*RoTS*		
	f_{best}	f_{avr}	t_{avr}	f_{best}	f_{avr}	t_{avr}	f_{best}	f_{avr}	t_{avr}
Randwb01	16069	17085.7	207.3	10626	11440.2	612.4	11486	12263.5	509.9
Randwb02	13929	14907.5	208.4	9505	10186.6	613.4	10251	10989	514.8
Randwb03	13587	15037.7	314.7	10037	11098.9	583.7	10849	11485.5	583.5
Randwb04	12440	13510.6	287.5	7965	9072.4	653.4	9156	10089.9	582.3
Randwb05	15627	17204.5	254.3	11260	11959.1	598.6	11642	12478.1	478.9
Randwb06	17053	17814.4	235.6	11464	12146.3	568.3	11399	12458.2	542.9
Randwb07	15601	17167.8	287.5	10434	11594.3	645.7	11530	12286.5	387.6
Randwb08	14552	16277.3	305.6	10089	11171.8	587.3	10005	11399.9	437.5
Randwb09	16662	18175.9	312.8	11800	12457.4	689.3	11863	12811.7	398.7
Randwb10	16618	17662.9	235.6	11290	12392.2	624.8	12227	12979.8	412.8

*: due to the unavailability of the weighted version of $AdaptG2WSAT_p$ in UBCSAT, we employ its simplified version *G2WSAT* in this experiment.

twice over 20 runs. In terms of the average search step for reaching the best known results, our *AMLS1* algorithm has the best value for 2 instances while only *IRoTS* outperforms *AMLS1*, having the best value for 9 instances. In sum, both versions of our *AMLS* algorithm are effective in finding the best known objective values compared with the 4 reference algorithms for this set of instances.

In Table 3, we show our computational results on the 10 *WeightedMaxSat* instances. Note that this set of instances is in weighted MAX-SAT version. For each algorithm, we show results according to the following three criteria: (1) the best objective value, f_{best}, over 20 independent runs, (2) the average objective value, f_{avr}, over 20 independent runs and (3) the CPU time, t_{avr} (in seconds), for reaching the best objective value f_{best}. In this experiment, since the weighted version of $AdaptG2WSAT_p$ is not available in UBCSAT, we use its previous version *G2WSAT*.

Table 3 shows that both versions of our *AMLS* algorithm are competitive with the reference algorithms for these weighted MAX-SAT instances. Specifically, *AMLS1* and *AMLS2* obtain the best objective values f_{best} for 4

Table 4: Computational results on the *UnWeightedMaxSat* instances (stop condition: 10^7)

Instance	*AMLS1*			*AMLS2*			*AdaptNovelty+*			*AdaptG2WSAT$_p$*			*IRoTS*			*RoTS*		
	f_{best}	f_{avr}	t_{best}	f_{best}	f_{avr}	t_{best}	f_{best}	f_{avr}	t_{best}	f_{best}	f_{avr}	t_{best}	f_{best}	f_{avr}	t_{best}	f_{best}	f_{avr}	t_{best}
difp_19_0	**6**	14.8	21.4	**6**	12.4	11.9	9	9.45	8.73	10	13.2	11.4	21	27.8	66.5	23	28.4	52.5
difp_19_1	6	14.8	18.7	**4**	12.3	21.9	8	9.85	10.5	11	12.9	15.4	19	26.2	47.8	23	29.6	54.8
difp_19_3	**5**	14.6	13.3	**5**	12.6	17.0	8	9.45	5.86	11	13.2	9.68	23	29.4	67.3	24	28.2	63.8
difp_19_99	7	16.0	25.9	**6**	13.1	9.1	8	9.75	7.58	11	12.7	16.3	26	32.7	52.7	27	29.4	42.4
difp_20_0	**7**	16.5	22.5	**7**	12.9	9.6	9	10.1	9.87	11	12.6	18.8	18	28.2	41.0	27	30.1	39.7
difp_20_1	**6**	16.0	18.9	**6**	13.7	19.2	8	10.2	7.89	9	13.0	10.5	26	31.3	73.2	25	28.9	52.9
difp_20_2	**5**	14.1	20.7	**5**	12.4	6.6	8	10.2	6.86	10	12.9	17.8	26	30.1	62.1	26	29.6	60.3
difp_20_3	7	16.0	19.4	**5**	13.1	21.9	9	11.0	5.37	9	12.1	16.9	23	28.3	47.2	19	25.9	58.2
difp_20_99	8	17.3	32.8	**7**	14.4	3.9	**7**	10.3	10.8	11	12.8	15.2	26	32.5	52.8	28	30.9	44.4
difp_21_0	**7**	15.6	25.8	8	14.6	24.9	10	12.5	12.7	14	15.6	14.9	31	37.9	48.2	30	36.4	39.6
difp_21_1	**6**	17.4	15.4	7	14.9	17.9	9	11.8	14.7	12	15.6	8.97	34	38.6	42.7	31	36.7	35.4
difp_21_2	**8**	15.5	26.5	**8**	14.6	11.2	11	12.2	10.8	**8**	15.9	10.2	33	38.8	78.3	28	35.5	49.5
difp_21_3	**8**	11.8	18.9	**8**	10.5	12.4	10	12.1	8.97	14	16.3	16.4	34	39.7	69.3	32	36.7	58.1
difp_21_99	**7**	11.9	25.6	**7**	10.3	16.4	9	11.8	12.4	13	15.4	13.8	33	38.9	60.3	28	35.4	50.6
mat25.shuf	**3**	3.74	3.11	**3**	4.02	11.5	**3**	3.00	4.97	**3**	3.00	6.44	4	4.25	34.6	**3**	3.00	17.8
mat26.shuf	**2**	3.55	12.4	**2**	4.07	14.1	**2**	2.00	5.43	**2**	2.00	3.46	4	5.35	43.6	**2**	2.00	13.6
mat27.shuf	**1**	3.59	10.9	**1**	4.19	11.3	**1**	1.00	5.87	**1**	1.00	3.85	5	6.45	38.5	**1**	1.00	5.87
glassy-a	**6**	6.18	1.9	**6**	6.65	0.18	**6**	6.00	2.87	**6**	6.00	3.09	**6**	6.00	7.80	**6**	6.00	3.87
glassy-b	**6**	6.12	8.4	**6**	6.65	0.17	**6**	6.00	8.65	**6**	6.00	5.87	**6**	6.00	6.53	**6**	6.00	4.39
glassy-c	**5**	5.22	16.6	**5**	5.88	1.21	**5**	5.00	2.01	**5**	5.00	6.26	**5**	5.00	3.87	**5**	5.00	1.28
glassy-d	**7**	7.45	14.2	**7**	8.34	12.0	**7**	7.10	8.96	**7**	7.00	14.5	**7**	7.70	2.98	**7**	7.00	4.97
glassy-e	**6**	6.14	2.85	**6**	6.69	6.84	**6**	6.00	1.08	**6**	6.00	3.98	**6**	6.00	4.78	**6**	6.00	6.87
glassy-f	**8**	8.38	14.1	**8**	9.26	2.54	**8**	8.15	13.7	**8**	8.00	9.85	**8**	8.75	20.5	**8**	8.30	8.97
glassy-g	**7**	7.26	3.96	**7**	8.00	3.92	**7**	7.00	3.28	**7**	7.00	12.7	**7**	7.07	10.5	**7**	7.00	5.97
glassy-h	**9**	9.06	9.5	**9**	9.96	2.18	**9**	9.00	4.01	**9**	9.00	6.87	**9**	9.00	5.98	**9**	9.25	10.6
glassy-i	**7**	7.12	15.6	**7**	7.97	0.74	**7**	7.00	1.89	**7**	7.00	7.12	**7**	7.00	2.98	**7**	7.10	9.60
glassy-j	**6**	6.29	8.42	**6**	6.89	0.71	**6**	6.00	2.86	**6**	6.00	11.2	**6**	6.20	8.89	**6**	6.00	4.23

and 3 instances, respectively. *AdaptNovelty+* performs slightly better than our AMLS algorithm and it can reach the best objective values for 5 instances. However, the three other algorithms *G2WSAT*, *IRoTS* and *RoTS* performs worse than *AMLS* in terms of both the best and average objective values. These results demonstrate that our *AMLS* algorithm is quite competitive for solving this set of weighted MAX-SAT problems.

We present in Table 4 the computational results of our *AMLS* algorithms on the set of *UnWeightedMaxSat* instances. The symbols are the same as in Table 3. Table 4 indicates that our *AMLS* algorithm performs quite well on this set of instances. Specifically, for the 14 $difp$ instances, both versions of *AMLS* can obtain the best objective values f_{best} for 10 and 12 instances, respectively, while *AdaptNovelty+* and $AdaptG2WSAT_p$ can obtain the best objective values f_{best} only for one instance and the remaining two algorithms *IRoTS* and *RoTS* cannot obtain the best objective values for any instance. However, one notice that *AdaptNovelty+* can obtain better average objective values f_{avr} although it is inferior to our *AMLS* algorithm in terms of finding the best objective values. When it comes to the 3 *mat* and 10 *glassy* instances, both versions of *AMLS* can find the best objective values under this stop condition, while *IRoTS* has difficulty in finding the best objective values for 3 *mat* instances. In sum, this experiment further confirm the efficiency of our *AMLS* algorithm for solving the general unweighted MAX-SAT problems.

Finally, we present in Table 5 the computational statistics of our *AMLS* algorithm on the set of 29 DIMACS *Satisfiable* instances. In this experiment, we only compare our *AMLS* algorithm with $AdaptG2WSAT_p$ due to the space limit and the reason that $AdaptG2WSAT_p$ performs much better than other three reference algorithms for almost all these satisfiable instances. In this comparison, we use a tabu tenure as shown in Eq. (7) in our *AMLS* algorithm. In addition, we disable the perturbation operator in our *AMLS* algorithm (i.e., $Maxpert = 1$) and run our local search algorithm until the stop condition (i.e., the total number of iterations) is satisfied. According to the size and difficulty of these considered instances, column 2 gives the total number of iterations for each instance.

Columns 3 to 11 respectively give the computational results for the three algorithms: $AdaptG2WSAT_p$, *AMLS1* and *AMLS2*. For each algorithm, the following three criteria are presented: the success rate over 100 independent runs (*suc*), the average number of flips for the success runs (*#steps*) and the average CPU time (in seconds) for each success run (*time*). The best results for an instance and each criterion is indicated in bold.

When comparing our results with those obtained by $AdaptG2WSAT_p$,

Table 5: Computational results on the DIMACS *Satisfiable* benchmark instances

Instance	Iter	*AdaptG2WSAT*$_p$			*AMLS1*			*AMLS2*		
		#suc	*#steps*	*time*	*#suc*	*#steps*	*time*	*#suc*	*#steps*	*time*
ais8	10^5	96	31889	0.043	**100**	**15458**	0.018	99	20167	0.032
ais10	10^6	91	323812	0.711	**100**	**224846**	0.430	98	318071	0.786
ais12	10^7	**71**	3810914	9.625	65	**3671783**	9.380	47	4174483	11.27
bw_large.a	10^6	100	9635	0.012	100	11579	0.018	100	**6846**	0.008
bw_large.b	10^6	100	**81615**	0.128	96	254110	0.426	100	138123	0.240
bw_large.c	10^7	**100**	**1303511**	2.992	34	3922871	8.73	94	2746692	5.892
bw_large.d	10^7	**100**	**1948941**	5.990	18	4102525	12.14	59	4049814	11.64
f1000	10^6	**96**	419463	0.335	83	**397006**	0.535	62	431452	0.550
f2000	10^7	84	3950685	7.772	**99**	**2920372**	3.782	19	5975934	7.702
f3200	10^8	57	43978432	38.65	**100**	**13416486**	20.25	6	32411411	47.74
flat200-1	10^6	100	50662	0.033	100	**35797**	0.032	100	43960	0.042
flat200-2	10^6	93	343990	0.255	95	321643	0.303	95	**248619**	0.240
flat200-3	10^6	99	114717	0.077	100	124379	0.125	100	**101609**	0.105
flat200-4	10^6	89	361486	0.275	**100**	**192106**	0.185	95	217268	0.210
flat200-5	10^7	69	3883019	5.895	**100**	**1791504**	1.612	90	3626992	3.310
logistics.a	10^6	100	**51440**	0.056	100	89405	0.120	100	162293	0.241
logistics.b	10^6	100	**50710**	0.036	100	133716	0.190	94	292362	0.421
logistics.c	10^7	100	**107137**	0.115	100	705501	1.050	100	1319925	1.972
logistics.d	10^7	100	**121331**	0.137	4	7843512	21.65	100	2916208	5.050
par16-1-c	10^8	**100**	**12899883**	8.359	44	46162634	64.34	6	41464972	47.56
par16-2-c	10^8	**59**	65850012	51.38	37	**46484788**	58.92	5	72062553	114.8
par16-3-c	10^8	**88**	**43677124**	28.24	35	47293115	57.62	4	29702870	46.53
par16-4-c	10^8	**95**	**33141448**	21.78	53	39980734	47.65	5	68242170	104.3
par16-5-c	10^8	**93**	38197528	24.85	62	**27056467**	58.75	4	54002044	88.12
par16-1	10^9	56	747646999	281.6	**82**	**321157528**	470.1	4	523418603	791.6
par16-2	10^9	42	507322537	574.1	**76**	**449727827**	524.6	0	—	—
par16-3	10^9	47	466264326	298.1	**62**	**286739791**	446.6	5	332755276	520.4
par16-4	10^9	36	465727526	318.5	**83**	**356592452**	416.6	6	393824484	589.6
par16-5	10^9	41	481204281	315.2	**96**	**343774441**	412.2	4	312170271	1169

one observes that for the 29 tested instances, *AMLS1* and *AMLS2* reaches the solutions with a higher (respectively lower) success rate than *AdaptG2WSAT*$_p$ for 13 and 6 (respectively 11 and 17) instances, with equaling results for the remaining 5 and 6 instances. Globally, *AMLS1* and *AdaptG2WSAT*$_p$ perform better than *AMLS2* for this set of instances, demonstrating the advantage of breaking ties by favor of the least recently flipped variables and the first two algorithms are comparable with each other.

Roughly speaking, *AMLS1* performs better than *AdaptG2WSAT*$_p$ with respect to almost all the three criteria for the three sets ais, f and $flat$ of instances. However, *AdaptG2WSAT*$_p$ obtains better success rate than both *AMLS1* and *AMLS2* for the bw_large and $logistics$ instances. The 10 *parity* instances named $par16*$ are challenging for many SAT solvers. Interestingly, *AMLS1* obtains solutions with a higher (respectively lower) success rate than *AdaptG2WSAT*$_p$ for 5 (respectively 5) instances (i.e., *AMLS1* performs better on the $par16$ instances while *AdaptG2WSAT*$_p$ works better on the $par16$-c instances). On the other hand, one notice that *AMLS2* performs quite bad for this set of instances. Therefore, it is another interesting topic to investigate the different performance of the two versions of *AMLS*.

To summarize, our *AMLS* algorithm (especially *AMLS1*) is quite competitive for solving this set of challenging satisfiable instances.

5. Discussion and Remark

Our *AMLS* algorithm shares some similar components as the previous local search algorithms. First of all, *AMLS* incorporates both the intensification of *GSAT* to select the best variable to flip [23] and the diversification of *WalkSAT* to always pick a variable from one random unsatisfied clause to flip [22]. Secondly, *AMLS* adopts the adaptive mechanism introduced in [9] to enhance its robustness and utilizes the information of the recent consecutive falsification of a clause of [15] to strategically select the second best variable to flip. In addition, the famous "random walk" strategy is also used as one of the main tools to diversify the search in our *AMLS* algorithm.

However, our *AMLS* algorithm also possesses several distinguished features. First of all, it integrates all the above components into a single solver in a systematic way such that it can achieve a better tradeoff between intensification and diversification. Secondly, we introduce a penalty function guided by clause falsification and satisfaction information to refine our variable selection rule. Thirdly, our *AMLS* algorithm employs a dynamic tabu tenure strategy. Fourthly, *AMLS* extends the Hoos's adaptive mechanism [9] to automatically adjust the two diversification parameters p and wp. Last but not the least, an adaptive random perturbation operator is proposed to diversify the search when the search reaches a local optimum.

Although our *AMLS* algorithm has demonstrated good performance on a large set of public benchmark instances, it has several limitations. First of all, like other local search heuristic algorithms in the literature, *AMLS* has difficulty in solving some highly structured instances. Secondly, the adaptive parameter adjusting mechanism used in our algorithm uses several predefined parameters, whose tuning may require first-hand experience.

There are several directions to improve and extend this work. First, it would be instructive to get a deep understanding of the behavior of the algorithm. Second, it would be valuable to explore other search strategies like variable and clause weighting (e.g., [20, 21, 27]). Furthermore, it would be worthwhile to investigate the look-ahead strategy [16] and other neighborhoods based on higher-order flips [32]. Finally, it would be valuable to know whether integrating *AMLS* within the memetic framework would lead to improved performance.

6. Conclusion

In this paper, we have aimed to study various memory structures and their cooperation to reinforce the search efficiency of local search algorithms for solving the MAX-SAT and SAT problems. For this purpose, we have introduced the adaptive memory-based local search algorithm *AMLS*. Various memory-based strategies are employed to guide the search in order to achieve a suitable tradeoff between intensification and diversification. In addition, an adaptive mechanism is used to adjust the two diversification parameters in the *AMLS* algorithm according to the search history. Experimental comparisons with four leading MAX-SAT solvers (*AdaptNovelty+*, $AdaptG2WSAT_p$, *IRoTS* and *RoTS*) demonstrate the competitiveness of our algorithm in terms of the considered criteria.

The study reported in this work demonstrates the importance of memory as a source of pertinent information for the design of effective intensification and diversification strategies. It also confirms the usefulness of the adaptive mechanism proposed in [9] for dynamical parameter tunings.

Acknowledgment

We are grateful to the referees for their comments and questions which helped us to improve the paper. The work is partially supported by the regional MILES (2007-2009) and RaDaPop projects (2009-2012).

References

[1] R. Battiti, M. Protasi, Reactive Search, a history-based heuristic for MAX-SAT, ACM Journal of Experimental Algorithmics 2 (1997) 130–157.

[2] B. Borchers, J. Furman, A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems, Journal of Combinatorial Optimization 2(4) (1999) 299–306.

[3] D. Boughaci, B. Benhamou, H. Drias, Scatter search and genetic algorithms for MAX-SAT problems, Journal of Mathematical Modelling and Algorithms 7(2) (2008) 101–124.

[4] C. Fleurent, J.A. Ferland, Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability, In Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, American Mathematical Society, 26 (1996) 619–652.

[5] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, 2004.

[6] P. Hansen, B. Jaumard, Algorithms for the maximum satisfiability problem, Computing 44(4) (1990) 279–303.

[7] H. Hoos, T. Stützle, Stochastic Local Search: Foundations and Applications, Morgan Kaufmann, 2004.

[8] H. Hoos, On the run-time behaviour of stochastic local search algorithms for SAT. In Proceedings of AAAI-99 (1999) 661–666.

[9] H. Hoos, An adaptive noise mechanism for WalkSAT, In Proceedings of AAAI-2002 (2002) 655–660.

[10] F. Hutter, D. Tompkins, H. Hoos, Scaling and probabilistic smoothing: Efficient dynamic local search for SAT, In Proceedings of AAAI-2000 (2002) 233–248.

[11] K. Inouea, T. Sohb, S. Ueda, Y. Sasaura, M. Banbara, N. Tamura, A competitive and cooperative approach to propositional satisfiability, Discrete Applied Mathematics 154(16) (2006) 2291–2306.

[12] Y. Kilani, Comparing the performance of the genetic and local search algorithms for solving the satisfiability problems, Applied Soft Computing 10(1) (2010) 198–207.

[13] F. Lardeux, F. Saubion, J.K. Hao, GASAT: a genetic local search algorithm for the satisfibility problem, Evolutionary Computation 14(2) (2006) 223–253.

[14] C.M. Li, W.Q. Huang, Diversification and determinism in local search for satisfiability, Lecture Notes in Computer Science 3569 (2005) 158–172.

[15] C.M. Li, W. Wei, Combining adaptive noise and promising decreasing variables in local search for SAT, In SAT 2009 competitive events booklet: preliminary version (2009) 131–132.

[16] C.M. Li, W. Wei, H. Zhang, Combining adaptive noise and look-ahead in local search for SAT, Lecture Notes in Computer Science 4501 (2007) 121–133.

[17] M. Mastrolilli, L.M. Gambardella, Maximum satisfiability: how good are tabu search and plateau moves in the worst-case? European Journal of Operational Research 166(1) (2005) 63–76.

[18] B. Mazure, L. Saïs, E. Grégoire, Tabu search for SAT, In Proceedings of AAAI-97 (1997) 281–285.

[19] D. McAllester, B. Selman, H. Kautz, Evidence for invariants in local search, In Proceedings of AAAI-97 (1997) 321–326.

[20] D.N. Pham, J. Thornton, C. Gretton, A. Sattar, Advances in local search for satisfiability. Lecture Notes in Computer Science 4830 (2007) 213–222.

[21] S. Prestwich, Random walk with continuously smoothed variable weights, Lecture Notes in Computer Science 3569 (2005) 203–215.

[22] B. Selman, H.A. Kautz, B. Cohen, Noise strategies for improving local search, In Proceedings of AAAI-94 (1994) 337–343.

[23] B. Selman, D. Mitchell, H. Levesque, A new method for solving hard satisfiability problems, In Proceedings of AAAI-92 (1992) 440–446.

[24] S. Seitz, P. Orponen, An efficient local search method for random 3-satisfiability, Electronic Notes in Discrete Mathematics 16 (2003) 71–79.

[25] K. Smyth, H. Hoos, T. Stützle, Iterated robust tabu search for MAX-SAT, Lecture Notes in Artificial Intelligence 2671 (2003) 129–144.

[26] D. Taillard, Robust taboo search for the quadratic assignment problem, Parallel Computing 17(4-5) (1991) 443–455.

[27] J. Thornton, D.N. Pham, S. Bain, V. Ferreira, Additive versus multiplicative clause weighting for SAT, In Proceedings of AAAI-2004 (2004) 191–196.

[28] D. Tompkins, H. Hoos, UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT, Lecture Notes in Computer Science 3542 (2004) 305–319.

[29] M. Tounsi, S. Ouis, An Iterative local-search framework for solving constraint satisfaction problem, Applied Soft Computing 8(4) (2008) 1530–1535.

[30] W. Wei, C.M. Li, H. Zhang, A switching criterion for intensification and diversification in local search for SAT, Journal on Satisfiability, Boolean Modeling and Computation 4 (2008) 219–237.

[31] Z. Wu, B.W. Wah, Global-search strategy in discrete lagrangian methods for solving hard satiafiability problems, In Proceedings of AAAI-2000 (2000) 310–315.

[32] M. Yagiura, T. Ibaraki, Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: Experimental evaluation, Journal of Heuristics 7(5) (2001) 423–442.