© International Journal of Combinatorial Optimization Problems and Informatics,
Vol. 5, No. 3, Sep-Dec 2014, pp. 20-36. ISSN: 2007-1558.

Enhanced Walksat with Finite Learning Automata For MAX-SAT

N. Bouhmala *
Vestfold University College
Norway

Abstract

Researchers in artificial intelligence usually adopt the constraint satisfaction problem and the Satisfiability paradigms as their preferred methods when solving various real worlds decision making problems. Local search algorithms used to tackle different optimization problems that arise in various fields aim at finding a tactical interplay between diversification and intensification to overcome local optimality while the time consumption should remain acceptable. The Walksat algorithm for the Maximum Satisfiability Problem (MAX-SAT) is considered to be the main skeleton underlying almost all local search algorithms for MAX-SAT. This paper introduces an enhanced variant of Walksat using Finite Learning Automata. A benchmark composed of industrial and random instances is used to compare the effectiveness of the proposed algorithm against state-of-the-art algorithms.

Keywords: Walksat, Learning automata, Combinatorial optimization.

1 Introduction

The field of optimization has found numerous applications in science, engineering, economics, finance, and risk management. The domain of combinatorial optimization refers to optimization problems where the search space (i.e the set of all feasible solutions) is discrete. Combinatorial Optimization problems are mathematically modeled by introducing variables reflecting the quantities to be determined, a set of constraints, and a cost function whose numerical values expresses the quality of a given solution so that different solutions can be compared and the one having the highest/lowest function value could be selected. Efficient solvers for large scales optimization problems are eagerly sought as such problems arise frequently in diverse areas ranging from production scheduling, routing problems, data mining and military operations to name just a few. Large scale optimization problems are very challenging problems. Most of the recently developed optimization algorithms lose their efficiency when the dimensionality of the problems increases. They are NP-complete problems characterized by large sets of data, constraints and variables. In addition, the term 'large scale' is not closely related to the number of variables or constraints. Very often, problems are classified 'large scale' even if these numbers are moderate, but involve a certain structure that makes them hard for current optimization algorithms. Most searches that come up in large scale optimization occur over spaces that are far too large to be searched exhaustively. One way to overcome the combinatorial explosion is to give up completeness. Stochastic local search algorithms (SLS) are techniques which use this strategy and gained popularity in both worlds whether it is discrete or continuous due to their conceptual simplicity and good performance. The Walksat algorithm [30] is considered to be the main skeleton underlying almost all SLS algorithms for MAX-SAT. Although this technique has shown excellent search capabilities when applying to small or medium sized problems, it still encounter serious challenges when applying to large scale problems, i.e., problems with several hundreds to thousands of variables. The reasons appear to

*noureddine.bouhmala@hive.no

be two-fold. Firstly, the complexity of a problem usually increases with the increasing number of decision variables, constraints, or objectives (for multi-objective optimization problems). Secondly, the size of the solution space of the problem also increases exponentially with the number of decision variables. Because of these two issues, several optimization search techniques tend to spend most of the time exploring a restricted area of the search space preventing the search to visit more promising areas, and thus leading to solutions of poor quality. Designing efficient optimization search techniques requires a tactical interplay between diversification and intensification. The former refers to the ability to explore many different regions of the search space, whereas the latter refers to the ability to obtain high quality solutions within those regions. In recent years, bio-inspired algorithms have been used by several researchers for solving various optimization problems increasingly. The present paper aims at combining Finite Learning automata with Walksat in order to improve its performance. The paper is organised as follows. Section 2 explains the maximum satisfiablity problem. Section 3 present a survey of approaches used to solve MAX-SAT. Section 4 explains the Walksat algorithm. Section 5 introduces the combination of learning automata with the Walksat algorithm. Section 6 presents the results of the experiments while section 7 concludes the paper with future work.

2 The Maximum Satisfiability Problem

The satisfiability problem (SAT) which is known to be NP-complete [9] plays a central role problem in many applications in the fields of VLSI Computer-Aided design, Computing Theory, Artificial Intelligence and defence. SAT is an increasingly used paradigm that can model a wide spectrum of combinatorial optimization problems. It has become an important field of study in both theoritical and applied computer science. Generally, a SAT problem is defined as follows. A propositional formula $\Phi = \bigwedge_{j=1}^{m} C_j$ with m clauses and n Boolean variables is given. Each Boolean variable, $x_i, i \in \{1, \ldots, n\}$, takes one of the two values, ***True*** or ***False***. A clause , in turn, is a disjunction of literals and a literal is a variable or its negation. Each clause C_j has the form:

$$C_j = \left(\bigvee_{k \in I_j} x_k \right) \vee \left(\bigvee_{l \in \bar{I}_j} \bar{x}_l \right),$$

where $I_j, \bar{I}_j \subseteq \{1, ...n\}$, $I \cap \bar{I}_j = \emptyset$, and $\bar{x}_i$ denotes the negation of x_i. The task is to determine whether there exists an assignment of values to the variables under which Φ evaluates to ***True***. Such an assignment, if it exists, is called a satisfying assignment for Φ, and Φ is called satisfiable. Otherwise, Φ is said to be unsatisfiable. Most SAT solvers use a Conjunctive Normal Form (CNF) representation of the formula Φ. In CNF, the formula is represented as a conjunction of clauses, with each clause being a disjunction of literals. The maximum satisfiability problem is the optimization variant of SAT. More formally, let w_i denote the weight of clause C_i. Then equation 1 is the objective function to be maximized, with $S(C_i)$ is equal to 1 when C_i is true and 0 otherwise.

$$\sum_{k=1}^{m} w_i \cdot S(C_i), \tag{1}$$

There exists two important variations of the Max-SAT problem. The weighted Max-SAT problem is the Max-SAT problem in which each clause is assigned a positive weight. The goal of the problem is to maximize the sum of weights of satisfied clauses. The unweighted Max-SAT problem is the Max-SAT problem in which all the weights are equal to 1 and the goal is to maximize the number of satisfied clauses. In this paper, the focus is restricted to formulas in which all the weights are equal to 1 (i.e.unweighted Max-SAT).

3 Related Work

Various stochastic local search algorithms have been developed for MAX-SAT. They all start with an initial assignment of values to variables randomly or heuristically generated. During each iteration, a new solution is selected from the neighborhood of the current one by performing a move. Choosing a good neighborhood and a method for searching it is usually guided by intuition, because very little theory is available as a guide. All the methods usually differ from each other based on the criteria used to flip the chosen variable. One of the earliest local search for solving MAX-SAT is GSAT [31]. The GSAT algorithm operates by changing a complete assignment of variables into one in which the maximum possible number of clauses are satisfied by changing the value of a single variable. Another widely used variant of GSAT is the WalkSAT based on a two stage selection mechanism originally introduced in [30]. Several state-of-the-art local search algorithms are enhanced versions of GSAT and Walksat algorithms. Examples include GSAT/Tabu [22], WalkSAT/Tabu [23], Novelty+ and R-Novelty+ heuristics [15], G2WSAT [20]. Lacking the theoretical guidelines while being stochastic in nature, the deployment of several SLS involves extensive experiments to find the optimal noise or walk probability settings. The main difference between meta-heuristics relies in the way neighborhood structures are defined and explored. While The aforementioned meta-heuristics, work only with a single neighborhood structure, other meta-heuristics choose to operate on a set of different neighborhood structures giving rise to Variable neighborhood search algorithms [13] [21]. They aim at finding a tactical interplay between diversification and intensification to overcome local optimality using a combination of a local search with systematic changes of neighborhood. To avoid manual parameter tuning, new methods have been designed to automatically adapt parameter settings during the search [14] [19] and results have shown their effectiveness for a wide range of problems. As the the quality of the solution improves when larger neighborhood is use, the work proposed in [36] uses a restricted 2 and 3-flip neighborhoods and better performance has been achieved compared to the 1-flip neighborhood for structure problems. Clause weighting based SLS algorithms [4] [5] have been proposed to solve SAT and Max-SAT problems. The key idea is associate the clauses of the given CNF formula with weights. Although these clause weighting SLS algorithms differ in the manner clause weights should be updated (probabilistic or deterministic) they all choose to increase the weights of all the unsatisfied clauses as soon as a local minimum is encountered. Evolutionary algorithms are heuristic algorithms that have been applied to MAX-SAT problems. The Genetic local search algorithm (GASAT) [18] [16] is considered to be the best known genetic algorithm for MAX-SAT problems. GASAT is a hybrid algorithm that combines a specific crossover and a tabu search procedure. Experiments have shown that GASAT provides very competitive results compared with state-of-art SAT algorithms. Boughaci et al. introduced a new selection strategy based on both fitness and diversity to choose individuals to participate in the reproduction phase of a genetic algorithm [2]. Experiments showed that the resulting genetic algorithm was able to find solutions of a higher quality than the scatter evolutionary algorithm [3]. In [32], a new stochastic local search algorithm, called Iterated Robust Tabu Search (IRoTS), was presented for MAX-SAT that combines an Iterated Local Search and Tabu Search. The work presented in [12] introduced Learning Automata (LA) as a mechanism for enhancing SLS based SAT solvers, thus laying the foundation for novel LA-based SAT solvers. Finally, a new strategy based on an automatic procedure for integrating selected components from various existing solvers have been devised in order to build new efficient algorithms that draw the strengths of multiple algorithms [35] [17]. The work conducted in [37] proposed an adaptive memory based local search algorithm that exploits various strategies in order to guide the search to achieve a suitable trade-off between intensification and diversification. The computational results show that it competes favorably with some state-of-the-art MAX-SAT solvers. Finally, highly efficient solvers have emerged based on a new diversification scheme to prevent cycling [6] [7] [8].

4 Walksat/SKC Algorithm

In this section, the Walksat/SKC algorithm originally introduced in [30] and shown in Algorithm 1 is described. A random initial assignment is computed (line 2). The next step o fthe algorithm involves picking randomly an unsatisfied clause (line 4). If there exists a variable with break count equals to zero (line 5), this variable is flipped, otherwise a random variable (line 8) or the variable with minimal break count (line 10) is selected with a certain probability (noise probability)(line 7). The break count of a variable is defined as the number of clauses that would be unsatisfied by flipping the chosen variable. It turns out that the choice of unsatisfied clauses, combined with the randomness in the selection of variables, can enable Walksat to avoid local minima and to better explore the search space. The flips are repeated until a pre-set value of the maximum number of flips is reached (MAX-FLIPS) and this phase is repeated as needed up to MAX-TRIES times.

```
   input  : Problem in CNF format
   output: Number of satisfied clauses
 1 for i ← 1 to MAX-TRIES do
 2     T ←Random-Assignment ();
 3     for j ← 1 to MAXFLIPS do
 4         C_k ←Random-Unsatisfied-Clause();
 5         if (∃ variable v ∈ C_k with breakcount = 0) then
 6             Chosen-Variable ← v;
 7             else if random(0, 1) ≤ p_noise then
 8                 Chosen-Variable ←Random-Variable(C_k);
 9                 else
10                     chosen-variable ←Random-Lowest-Breakcount(C_k);
11                 end
12             end
```

Algorithm 1: Walksat Algorithm

5 The Algorithm

We base our work on the principles of Learning Automata [25] [33]. Learning Automata have been used to model biological systems [34], and have recently attracted considerable interest because they can learn the optimal actions when operating in (or interacting with) unknown stochastic environments. Furthermore, they combine rapid and accurate convergence with low computational complexity. Learning Automata solutions have been proposed for several other combinatorial optimization problems [28] [10] [11] [24] [26] [27] [29]. The work reported in [12] was the first to combine the traditional random walk with learning automata for the satisfiability problem. Inspired by the success of the above solution scheme, we will in the following propose how the classical GSAT-Random-Walk algorithm can be enhanced with learning capability, using Learning Automata.

5.1 The Automata and its Environment

Generally stated, a finite learning automaton performs a sequence of actions on an *environment*. The environment can be seen as a generic *unknown* medium that responds to each action with some sort of reward or penalty, perhaps *stochastically*. Based on the responses from the environment, the aim of the finite learning automaton is to find the action that minimizes the expected number of penalties received. Figure 1 illustrates the interaction between the finite learning automaton and the environment. Because we treat the environment as unknown, we will here only consider the definition of the finite learning automaton. The finite learning automaton can be defined in

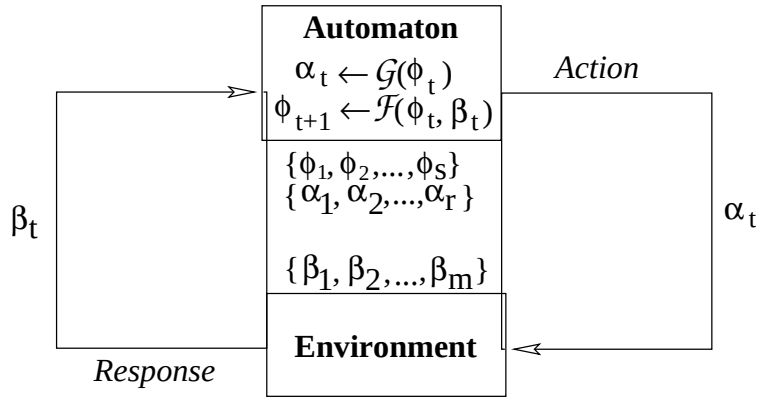

Figure 1: A learning automaton interacting with an environment

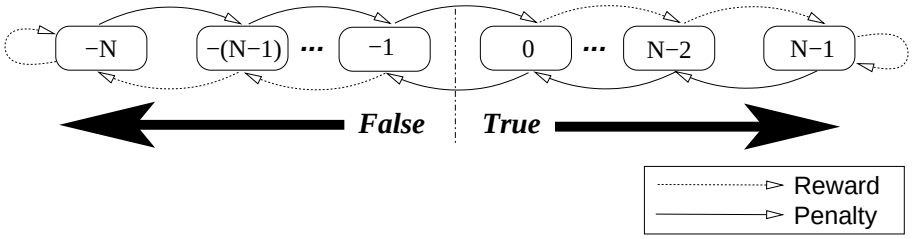

Figure 2: The state transitions and actions of the Learning SAT Automaton

terms of a quintuple [25]:

$$\{\underline{\Phi}, \underline{\alpha}, \underline{\beta}, \mathcal{F}(\cdot,\cdot), \mathcal{G}(\cdot,\cdot)\}.$$

$\underline{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_s\}$ is the set of internal automaton states. $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the set of automaton actions. And, $\underline{\beta} = \{\beta_1, \beta_2, \ldots, \beta_m\}$ is the set of inputs that can be given to the automaton. An output function $\alpha_t = \mathcal{G}[\phi_t]$ determines the next action performed by the automaton given the current automaton state. Finally, a transition function $\phi_{t+1} = \mathcal{F}[\phi_t, \beta_t]$ determines the new automaton state from (1) the current automaton state and (2) the response of the environment to the action performed by the automaton. Based on the above generic framework, the crucial issue is to design automata that can learn the optimal action when interacting with the environment. Several designs have been proposed in the literature, and the reader is referred to [25] [33] for an extensive treatment. In this paper we target the SAT problem, and our goal is to design a team of Learning Automata that seeks the solution of SAT problem instances. We build upon the work of Tsetlin and the linear two-action automaton [34] [25]. For each literal in the SAT problem instance that is to be solved, we construct an automaton with

- States: $\underline{\Phi} = \{-N-1, -N, \ldots, -1, 0, \ldots, N-2, N\}$.
- Actions: $\underline{\alpha} = \{\boldsymbol{True}, \boldsymbol{False}\}$.
- Inputs: $\underline{\beta} = \{reward, penalty\}$.

Figure 2 specifies the $\mathcal{G}$ and $\mathcal{F}$ matrices. The $\mathcal{G}$ matrix can be summarized as follows. If the automaton state is positive, then action ***True*** will be chosen by the automaton. If on the other hand the state is negative, then action ***False*** will be chosen. Note that since we initially do not know which action is optimal, we set the initial state of the Learning SAT Automaton randomly to either *'-1'* or *'0'*.

The state transition matrix $\mathcal{F}$ determines how learning proceeds. As seen in the figure, providing a *reward* input to the automaton *strengthens* the currently chosen action, essentially by making it less likely that the other action will be chosen in the future. Correspondingly, a *penalty* input *weakens* the currently selected action by making it more likely that the other action will be chosen later on. In other words, the automaton attempts to incorporate past responses when deciding on a sequence of actions.

5.2 Combining Walksat & Learning Automata (LA-WSAT)

Overview: In addition to the definition of the LA, we must define the environment that the LA interacts with. Simply put, the environment is a MAX-SAT problem instance as defined in Section 1. Each variable of the MAX-SAT problem instance is assigned a dedicated LA, resulting in a team of LA. The task of each LA is to determine the truth value of its corresponding variable, with the aim of satisfying all of the clauses where that variable appears. In other words, if each automaton reaches its own goal, then the overall MAX-SAT problem at hand has also been solved.

Pseudo-code: With the above perspective in mind, we will now present the details of the LA-WSAT that we propose. Algorithms 2-4 contain the complete pseudo-code for solving MAX-SAT problem instances, using a team of LA. An ordinary WSAT strategy is used to penalize an LA when it "disagrees" with WSAT, i.e., when WSAT and the LA suggest opposite truth values. Additionally, we use an "inverse" WSAT strategy for rewarding an LA when it agrees with WSAT. Note that as a result, the assignment of truth values to variables is indirect, governed by the states of the LA. At the core of the LA-WSAT algorithm is a punishment/rewarding scheme that guides the team of LA towards the optimal assignment. In the spirit of automata based learning, this scheme is incremental,and learning is performed gradually, in small steps.

Remark 1: Like a two-action Tsetlin Automaton, our proposed LA seeks to minimize the expected number of penalties it receives. In other words, it seeks finding the truth assignment that minimizes the number of unsatisfied clauses among the clauses where its variable appears.

Remark 2: Note that because multiple variables, and thereby multiple LA, may be involved in each clause, we are dealing with a game of LA [25]. That is, multiple LA interact with the same environment, and the response of the environment depends on the actions of several LA. In fact, because there may be conflicting goals among the LA involved in the LA-WSAT, the resulting game is competitive. The convergence properties of general competitive games of LA have not yet been successfully analyzed, however, results exists for certain classes of games, such as the Prisoner's Dilemma game [25]. In our case, the LA involved in the LA-WSAT are non-absorbing, i.e., every state can be reached from every other state with positive probability. This means that the probability of reaching the solution of the SAT problem instance at hand is equal to 1 when running the game infinitely. Also note that the solution of the MAX-SAT problem corresponds to a Nash equilibrium of the game.

Remark 3: In order to maximize speed of learning, we initialize each LA randomly to either the state *'-1'* or *'0'*. In this initial configuration, the variables will be flipped relatively quickly because only a single state transition is necessary for a flip. Accordingly, the joint state space of the LA is quickly explored in this configuration. However, as learning proceeds and the LA move towards their boundary states, i.e., states *'-N'* and *'N-1'*, the flipping of variables calms down. Accordingly, the search for a solution to the MAX-SAT problem instance at hand becomes increasingly focused.

6 Experimental Results

6.1 Test Suite & Parameter Settings

The performance of LA-WSAT is evaluated against WSAT using a set of real industrial problems and random problems.. This set is taken from the sixth Max-SAT 2011 organized as an affiliated event of the Fourteenth International Conference on Theory and Applications of Satisfiability

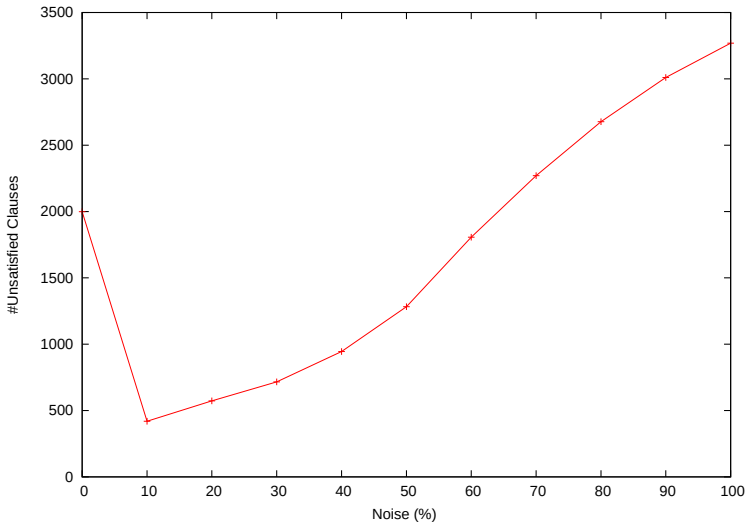

Figure 3:
Noise probability Vs Number of unsatisfied clauses: dividers6-hack.dimacs.filtered: Variables = 35376,clauses = 132699.

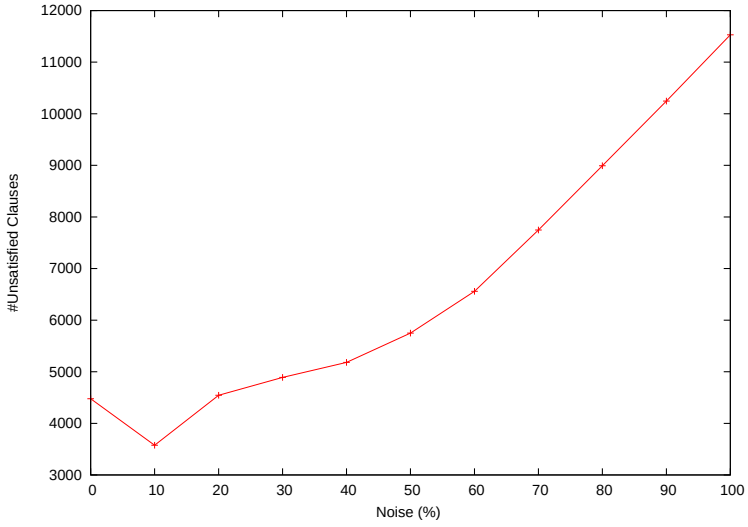

Figure 4:
Noise probability Vs Number of unsatisfied clauses:fpu8-problem.dimacs-24.filtered: Variables = 160232 , clauses = 548848.

```
   input  : Problem in CNF format
   output: Number of Unsatisfied clauses
 1 begin
 2   for i ← 1 to n do
 3     /* The initial state of each automaton is set to either '-1' or '1' */
 4     state[i] = random_element({−1, 0});
 5     /* And the respective literals are assigned corresponding truth values */
 6     if (state[i] == -1) then
 7       x_i = False;
 8     else
 9       x_i = True ;
10     end
11   end
12   while (Not Stop) do
13     C_k ←Random-Unsatisfied-Clause ();
14     if (∃ a literal ∈C_k with breakcount = 0) then
15       chosen-literal ←Random-Chosen-Literal-Candidate(C_k) ;
16       Reward-LA (LA,chosen-literal) ;
17     end
18     else if (random(0, 1) ≤ p_noise) then
19       chosen-literal ←Random-Literal(C_k);
20       Punish-LA (LA,chosen-literal) ;
21     end
22     else
23       chosen-literal ←Random-Lowest-Breakcount(C_k)
24       Reward-LA(LA,chosen-literal);
25     end
26   end
27 end
```

Algorithm 2: Walksat-Learning-Automata Based Algorithm

Testing (SAT-2011). Due to the randomization nature of both algorithms, each problem instance was run 50 times with a cut–off parameter (max-time) set to 30 minutes. The tests were carried out on a DELL machine with 800 MHz CPU and 2 GB of memory. The code was written in C++ and compiled with the GNU C compiler version 4.6. The following parameters have been fixed experimentally and are listed below:

- The number of states N is set to 3.
- Noise probability: The performance of WSAT depends highly on the walking probability setting which in turns depends on the class of problems to be solved. The plots in Figures 3-4 show 2 selected tests that reflect the general trend observed on almost all the industrial instances tested. Peak performance with respect to the lowest number of unsatisfied clauses is achieved when the walking probability was set to 10.

6.2 Results and Discussions

Figures 5-6 show the evolution of the mean of unsatisfied clauses for both algorithm as a function of time on a logarithmic scale. Both algorithms start with almost an identical initial solution. Already during the early stage of the search, the difference in solution quality between the two algorithms start to become more distinctive. In the first phase which corresponds to the early part of the search, both algorithms behave as a hill-climbing method. The mean number of unsatisfied clauses decreases rapidly at first, and then flattens off when entering the so-called a plateau region

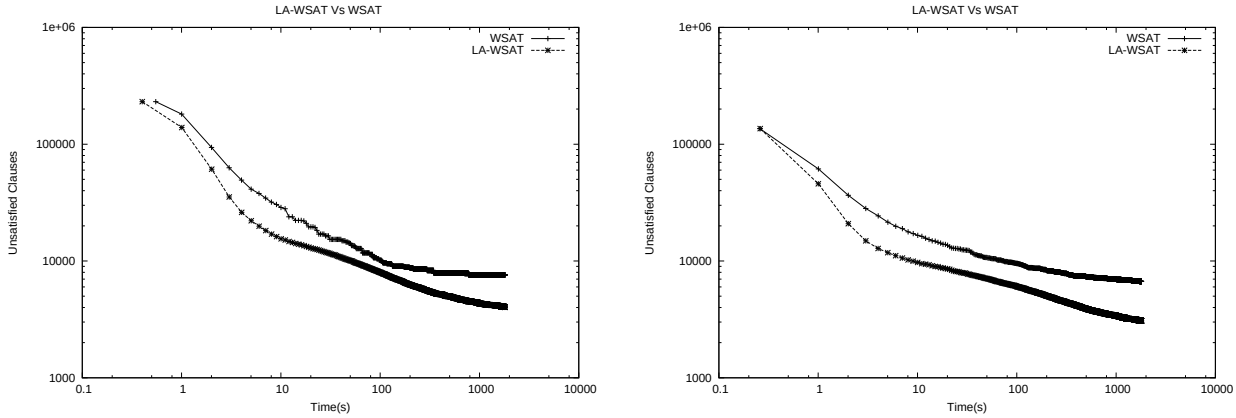

Figure 5:
Log-Log plot: (left) C2-DD-S3-f1-e2-v1-bug-fourvec-gate.dimacs.seq.filtered.cnf: $|V| = 400085$, $|C| = 1121810$, (right) divider-problem.dimacs-2.filtered.cnf:$|V| = 228874$, $|C| = 750750$.

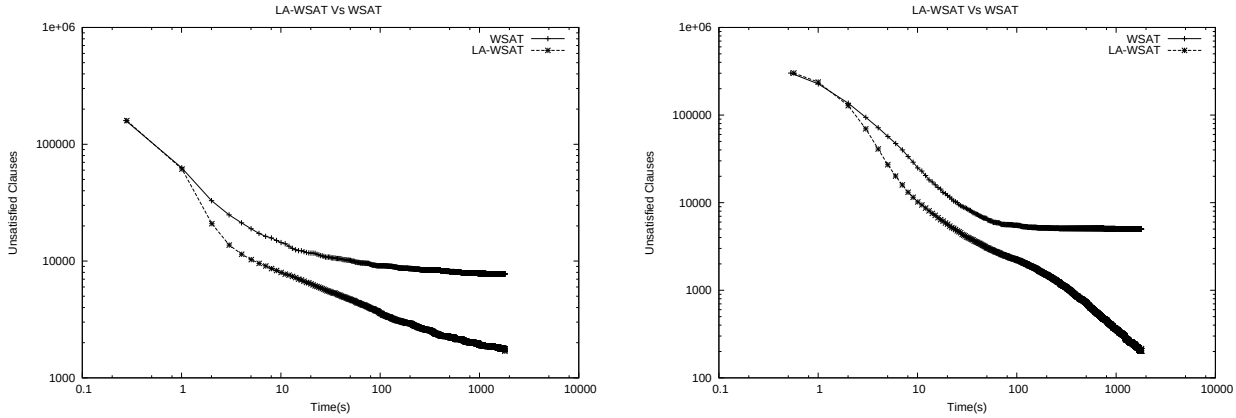

Figure 6:
Log-Log plot: (left) fpu-multivec1-problem-dimacs-14.filtered: $|V| = 257168$, $|C|928310$. (right) sudoku-debug-dimacs: $|V| = 1304121$, $|C| = 1554820$.

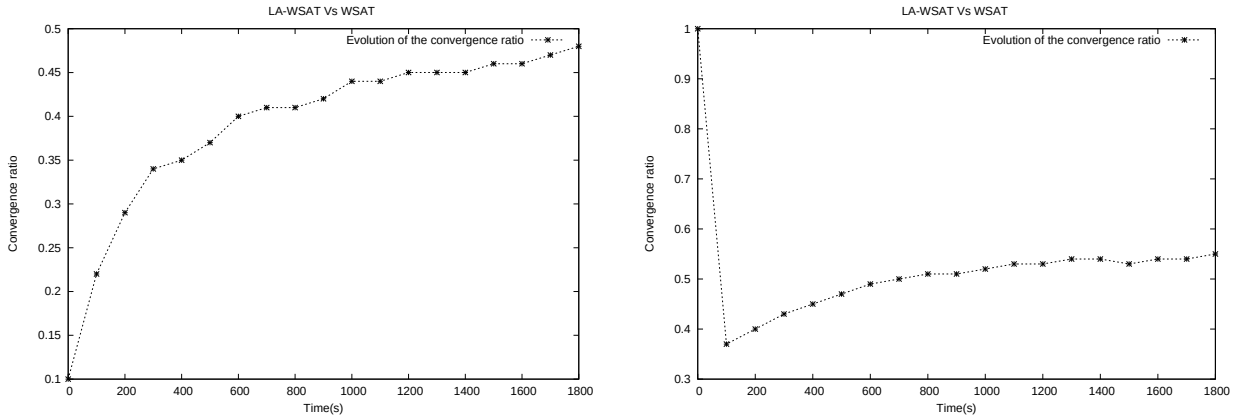

Figure 7:
Convergence Ratio:(left) C2-DD-S3-f1-e2-v1-bug-fourvec-gate.dimacs.seq.filtered.cnf: $|V| = 400085$, $|C| = 1121810$, (right) divider-problem.dimacs-2.filtered.cnf:$|V| = 228874$, $|C| = 750750$.

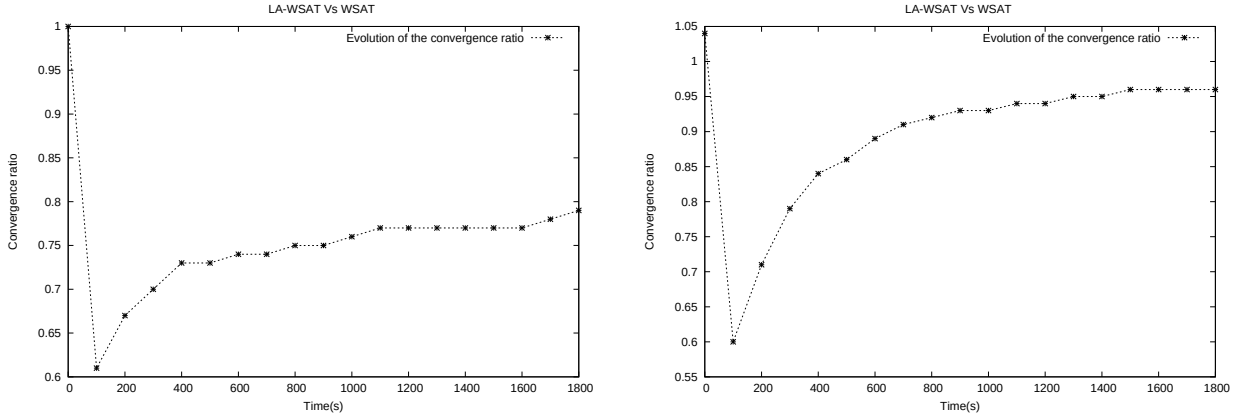

Figure 8:
Convergence Ratio: (left) fpu-multivec1-problem-dimacs-14.filtered: $|V| = 257168$, $|C|928310$.
(right) sudoku-debug-dimacs: $|V| = 1304121$, $|C| = 1554820$.

```
   input  : A literal and its learning automata
   output: Penalize the learning automata
 1 begin
 2 |  if (state[i] < N -1 ) then
 3 |  |  state[i] ← state[i] + 1;
 4 |  end
 5 |  /* Flip literal when automaton changes its action */ ;
 6 |  if (state[i]== 0) then
 7 |  |  Flip (l_i) ;
 8 |  end
 9 |  if (state[i] > -N ) then
10 |  |  state[i] ← state[i] − 1;
11 |  end
12 |  /* Flip literal when automaton changes its action */ ;
13 |  if (state[i]== -1) then
14 |  |  Flip (l_i) ;
15 |  end
16 end
```

Algorithm 3: Punishment Mechanism

```
   input  : A literal and its learning automata
   output: Reward the learning automata
 1 begin
 2 |  if (state[i] ≥ 0 And state[i] < N-1) then
 3 |  |  state[i] ← state[i] + 1;
 4 |  end
 5 |  if (state[i] < 0 And state[i] < N-1) then
 6 |  |  state[i] ← state[i] − 1;
 7 |  end
 8 end
```

Algorithm 4: Reward Mechanism

marking the start of the second phase. The plateau region spans a region in the search space where flips typically leave the number of unsatisfied clauses unchanged, and this phenomenon occurs earlier with WSAT leading to a possibly premature convergence. From these plots, LA-WSAT algorithm dominates WSAT throughout the run offering a better asymptotic convergence compared to WSAT. Figures 7-8 show the convergence speed behavior for the plots described in Figures 5-6 expressed as the ratio between the mean of unsatisfied clauses of the two algorithms as a function of time. A value greater than 1 demonstrates the superiority of WSAT while a value below 1 confirms the opposite. For some instances, WSAT is initially somewhat faster as observed in the right plot of Figure 8. LA-WSAT exhibits a better convergence speed compared to WSAT. The asymptotic performance offered by LA-WSAT is impressive, and dramatically improves on WSAT. The convergence rate observed with LA-WSAT continues to increase throughout the run reaching a value of 50%. On other instances this ratio increases until it reaches a high convergence ratio (getting as high as 93%) before it maintains it until the end of the search. Tables 1-3 compares LA-WSAT against WSAT using industrial instances. The first and second column show the number of variables and clauses respectively. LA-WSAT is capable of delivering solutions of excellent quality compared to WSAT. LA-WSAT dominates WSAT in 39 cases out 45 cases. The difference in the quality ranges within 10% for 7% of the cases, 20% for 5% of the cases, and above 30% for the remaining cases, getting as high as 96%. Table 4-6 compares LA-WSAT with highly efficient solvers such as CCLS [6] and Optimax which is a modified version of glucose SAT solver [1]

Instances	$\|V\|$	$\|C\|$	**WSAT**	**LA-WSAT**
fpu-multivec1-problem.dimacs-14.filtered	257168	928310	6125	1769
fpu-fsm1-problem.dimacs15.filtered	160200	548843	3716	422
fpu8-problem.dimacs24.filtered	160200	548843	3585	412
i2c-problem.dimacs.filtered	521672	1581471	615	161
b15-bug-fourvec-gate-0.dimacs	581064	1712690	7241	1569
c1-DD-s3-f1-e2-v1-bug-fourvec-gate-0.dimacs	391897	989885	955	41
c4-DD-s3-f1-e1-v1-bug-gate-0.dimacs	797728	2011216	3761	1911
c4-DD-s3-f1-e2-v1-bug-fourvec-gate-0.dimacs	448465	1130672	1834	640
c5-DD-s3-f1-e1-v2-bug-gate-0.dimacs	200944	540984	8	8
c6-DD-s3-f1-e1-v1-bug-gate-0.dimacs	298058	795900	3188	1827
divider-problem.dimacs11.filtered	215964	709377	9992	2675
divider-problem.dimacs2.filtered	228874	750705	10688	3073
divider-problem.dimacs3.filtered.cnf	216900	711249	6204	2117
divider-problem.dimacs5.filtered	228874	750705	11194	3271
divider-problem.dimacs8.filtered	246943	810105	7257	3363

Table 1: SAT2013 Industrial benchmarks: LA-WSAT Vs WSAT

Instances	$\|V\|$	$\|C\|$	**WSAT**	**LA-WSAT**
dividers10.dimacs.filtered	45552	162874	694	64
dividers-multivec1.dimacs.filtered	106128	397650	3015	361
i2c-problem.dimacs25	521672	1581471	2686	2498
i2c-master1.dimacs.filtered.cnf	82429	285987	317	95
mim-ctr1-dimacs.filtered	1128648	4422185	10198	1275
rsdecoder1-blackboxKESblock-problem.dimacs	707330	1106376	3757	3280
rsdecoder1-blackbox-CSEEblock-problem.dimacs32.filtered	277950	806460	3479	1868
rsdecoder4.dimacs.filtered	237783	933978	277	928
rsdecoder5.dimacs.filtered	238290	936006	280	708
rsdecoder-debug.dimacs	847501	2223029	19654	7746
rsdecoder-fsm2.dimacs.filtered	238290	936006	247	651
rsdecoder-multivec1.dimacs.filtered	394446	1626312	2143	4441
rsdecoder-multivec1-problem.dimacs38.filtered	1199012	3865513	29810	19318
rsdecoder-problem.dimacs39.filtered	1199602	3868693	28425	18005
rsdecoder-problem.dimacs41.filtered	1186710	3829036	27442	18328

Table 2: SAT2013 Industrial benchmarks: LA-WSAT Vs WSAT

Instances	$\|V\|$	$\|C\|$	**WSAT**	**LA-WSAT**
SM-AS-TOP-buggy1.dimacs.filtered	145900	694438	3791	174
SM-MAIN-MEM-buggy1.dimacs.filtered	870975	3812147	45360	8249
SM-RX-TOP.dimacs.filtered	235456	934091	3645	676
sukdoku-debug.dimacs	1304121	1554820	4971	202
wb1.dimacs.filtered	49525	140091	403	371
wb2.dimacs.filtered	49490	140056	786	773
wb-4m8s1.dimacs.filtered	463080	1759150	2543	686
wb-4m8s3.dimacs.filtered	463080	1759150	2769	911
wb-4m8s4.dimacs.filtered	463080	1759150	2628	1012
wb-commax1.dimacs-45.filtered	277950	1221020	5020	1715
wb-commax3.dimacs-45.filtered	277950	1221020	5313	416
wb-debug.dimacs	399591	621323	370	316
wb-problem-dimacs-45	309491	806440	163	421
wb-problem-dimacs-46	300846	789283	793	943
spi2.dimacs.filtered	124260	515813	919	191

Table 3: SAT2013 Industrial benchmarks: LA-WSAT Vs WSAT

ranked 1st at the 2011 SAT competition. CCLS won four categories of the incomplete algorithms track of MaxSAT Evaluation 2013. The instances used in the benchmark belong to random and crafted categories used at SAT2013 competition. Compared to CCLS, LA-WSAT gave similar results in 50 cases out of 61 cases. However the time of CCLS is several order of magnitude faster that of LA-WSAT. The remaining cases where LA-WSAT was beaten, the difference in quality ranges from 1% and 6%. Another interesting remark to mention is that the time required by LA-WSAT does vary significantly depending on the problem instance while the variations observed with CCLS remain very low. The comparison between Optimax and LA-WSAT shows that that Optimax converges very fast at the expense of delivering solutions of poor quality compared to LA-WSAT. LA-WSAT was capable of delivering solutions of better quality than Optimax in 39 out of 45 cases. The improvement ranges from 5% and 44%. The cases where LA-WSAT and Optimax gave similar results, Optimax was several order of magnitude faster.

7 Conclusions

In this work, a new approach based on combining learning Finite automaton with Walksat for MAX-SAT is introduced. Thus, in order to get a comprehensive picture of the new algorithms performance, a set of large industrial instances is used. The results indicate that learning finite automaton can enhance the convergence behavior of the walksat algorithm. It appears clearly from the results that LA-WSAT can find excellent solutions compared to those of WSAT at a faster convergence rate. The results have shown that in most of the studied cases, LA-WSAT is capable of delivering solution of similar quality compared to CCLS while the time invested is several order of magnitude slower than CCLS. When compared to Optimax, LA-WSAT showed a better performance as it provides solution of better quality. For the time being, further work is mainly conducted on improving the solution quality of LA-WSAT by combining it with the multilevel paradigm. The approach suggests looking at the solution of the problem as a multilevel process operating in a coarse-to-fine strategy. This strategy involves recursive coarsening to create a hierarchy of increasingly smaller and coarser versions of the original problem. The reduction phase works by grouping the variables representing the problem into clusters. This phase is repeated until the size of the smallest problem falls below a specified reduction threshold. A solution for the problem at the coarsest level is generated, and then successively projected back onto each of the intermediate levels in reverse order. The solution at each child level is improved using LA-WSAT before moving to the parent level.

	CCLS		Optimax		LA-WSAT	
Instance	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**
MANNa-27.clq	404	0.29	486	0.10	404	1.08
MANNa-45.clq	418	0.19	518	0.13	418	1.02
MANNa-81.clq	399	0.06	441	0.12	399	1.08
MANN-a9.clq	422	1.16	584	0.10	422	12.03
brock200-1.clq	238	0.95	349	0.10	238	37.03
brock200-2.clq	141	1.11	221	0.13	141	4.01
brock200-3.clq	214	0.96	232	1.75	214	6.53
brock-200-4.clq	209	1.04	299	0.11	209	4.40
brock400-1	255	0.38	340	0.08	256	13.02
brock400-2	252	0.84	310	0.08	252	9.89
brock400-3	238	1.27	278	0.08	239	17.01
brock400-4	249	0.73	374	0.08	250	7.61
brock800-1	205	0.95	273	0.09	205	2.18
brock800-2	207	0.97	270	0.09	207	7.05
brock800-3	203	0.47	315	0.07	203	1.08
brock800-4	200	0.32	310	0.13	200	4.27
cfat200-1.clq	4	0.01	4	0.01	4	1.07
cfat200-2.clq	26	0.71	26	3.93	26	1.2
cfat200-5.clq	116	0.47	172	0.08	116	1.8
c-fat500-1.clq	2	0.01	2	0.01	2	10.04

Table 4: Comparing LA-WSAT with CCLS and Optimax

	CCLS		Optimax		LA-WSAT	
Instance	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**
hamming10-2	400	0.09	532	0.08	400	1.08
hamming10-4	319	0.42	341	0.09	320	5.05
hamming6-2	832	1.18	1100	0.13	844	30.04
hamming6-4	192	1.00	312	0.13	192	1.06
hamming8-2	441	0.12	551	0.13	441	1.09
hamming8-4	176	1.17	254	0.14	176	4.70
johnson16-2-4.clq	215	0.68	344	0.08	215	1.07
johnson32-2-4.clq	329	1.51	492	0,09	329	16.09
johnson8-2-4.clq	75	0.34	100	0.08	75	1.09
johnson8-4-4.clq	770	1.59	1069	0.09	779	90.04
keller4.clq	199	0.74	344	1.01	199	2.02
keller5.clq	250	0.49	317	1.01	250	12.87
p-hat1000-1.clq	52	0.54	52	8.59	52	1.10
p-hat1000-2.clq	142	0.79	181	0.13	142	56.04
p-hat1000-3.clq	238	1.04	362	0.09	238	8.33
p-hat300-1.clq	49	0.50	49	5.38	49	2.19
p-hat300-2.clq	135	0.80	184	0.08	135	70.06
p-hat300-3.clq	269	1.41	327	0.12	269	69.06
phat500-1.clq	75	0.50	108	0,08	75	4.02
phat500-2.clq	176	0.59	244	0.09	176	65.01

Table 5: Comparing LA-WSAT with CCLS and Optimax

	CCLS		Optimax		LA-WSAT	
Instance	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**
phat500-3.clq	284	1.29	434	0.11	285	61.01
phat700-1.clq	63	0.63	63	21.42	63	2.17
phat700-2.clq	154	1.19	224	0.10	156	74.14
phat700-3.clq	267	1.36	395	0.12	268	19.04
san1000.clq	139	0.70	146	17.30	139	7.04
san200-0.7-1.clq	237	0.55	313	0.11	237	66.01
san200-0.7-2.clq	236	0.30	288	0.10	237	1.07
san200-0.9-1.clq	313	0.85	403	0.08	315	9.04
san200-0.9-2.clq	316	1.00	440	0.12	316	60.02
san200-0.9-3.clq	320	1.19	442	0.09	320	11.02
san400-0.5-1.clq	146	1.15	220	0.11	146	2.18
san400-0.7-1.clq	236	0.81	333	0.13	236	32.02
san400-0.7-2.clq	236	0.93	295	0.13	236	7.03
san400-0.9-1.clq	304	0.94	460	0.08	304	46.01
sanr200-0.7.clq	227	0.62	280	0.09	227	4.42
sanr200-0.9.clq	300	1.02	418	0.13	300	13.01
sanr400-0.5.clq	148	0.60	176	0.13	148	1.09
sanr400-0.7.clq	223	0.60	318	0.09	223	37.35
t4pm3-6666.spn	38	1.55	40	69.11	38	1.08
t5pm3-7777.spn	78	1.46	120	0.10	78	5.51
t6pm3-8888.spn	136	3.30	222	0.09	144	93.11

Table 6: Comparing LA-WSAT with CCLS and Optimax

References

[1] G. Audemard, L. Simon, in Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09), july 2009.

[2] D. Boughaci, B. Benhamou, and H. Drias. . Scatter Search and Genetic Algorithms for MAX-SAT Problems. J.Math.Model.Algorithms, pages 101-124, 2008.

[3] D. Boughaci and H. Drias. Efficient and experimental meta-heuristics for MAX- SAT problems. In Lecture Notes in Computer Sciences, WEA 2005,3503/2005:501-512, 2005.

[4] B. Cha and K. Iwama. Performance Tests of Local Search Algorithms Using New Types of Random CNF Formula. Proceedings of IJCAI95, pages 304-309. Morgan Kaufmann Publishers, 1995

[5] J. Frank. Learning Short-term Clause Weights for GSAT. Proceedings of IJCAI97, pages 384-389, Morgan Kaufmann Publishers, 1997.

[6] S. Cai, C. Luo, K. Su. CCASat: Solver description. In: Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions. pages 1314 2012.

[7] S. Cai, K. Su. Configuration checking with aspiration in local search for SAT. In: Proc. of AAAI-12. pages . 434440, 2012.

[8] S. Cai,, K. Su., A. Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif. Intell. 175(9-10), pages 16721696 (2011).

[9] S.A. Cook. The complexity of theorem-proving procedures. Proceedings of the Third ACM Symposium on Theory of Computing, pages: 151-158, 1971.

[10] W. Gale, S. Das, and C.T. Yu. Improvements to an Algorithm for Equipartitioning. IEEE Transactions on Computers, 39 (5), pages: 706-710, IEEE, 1990.

[11] O.C. Granmo, B.J. Oommen, S.A. Myrer, and M.G. Olsen. Learning Automata-Based Solutions to the Nonlinear Fractional Knapsack Problem With Applications to Optimal Resource Allocation. IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-37(B), pages: 166-175, 2007.

[12] O.C. Granmo, N. Bouhmala. Solving the satisfiability problem using finite learning automata. International Journal of Computer Science and Applications, 4(3) pages:15-29, 2007.

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

[14] H. Hoos, An adaptive noise mechanism for WalkSAT, In Proceedings of AAAI-2002, 655660, 2002.

[15] H. Hoos, On the run-time behavior of stochastic local search algorithms for SAT. In Proceedings of AAAI-99, 661666, 1999.

[16] H. Jin-Kao, H., Lardeux, F., and Saubion, F.(2003). Evolutionary computing for the satisfiability problem. In Applications of Evolutionary Computing, volume 2611 of LNCS, pages 258-267, University of Essex, 2003, England.

[17] A.R. KhudaBukhsh, L. Xu, H.H. Hoos, K. Leyton-Brown. SATenstein: Automatically Building Local Search SAT Solvers From Components. Proceedings of the 25th Interna- tional Joint Conference on Artificial Intelligence (IJCAI-09), 2009.

[18] F. Lardeux, F. Saubion, and H. Jin-Kao. GASAT: A Genetic Local Search Algo- rithm for the Satisfiability Problem. Evolutionary Computation, 14(2), MIT Press, 2006.

[19] 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. 121133.

[20] C.M. Li and W.Q. Huang. Diversification and determinism in local search for satisfiability. Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT-05), volume 3569 of Lecture Notes in Computer Science, pages 158-172, 2005.

[21] N. Mladenović, P. Hansen. Variable Neighborhood Search. Computer and Operations Research, pages: 24:1097-1100, 1997.

[22] B. Mazure, L. *Saïs*, and E. *Grégoire*. Tabu search for SAT. Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), pages 281-285, 1997.

[23] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), pages 321-326, 1997.

[24] S. Misra and B.J. Oommen. Dynamic Algorithms for the Shortest Path Routing Problem: Learning Automata-Based Solutions. IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-35(B), pages: 1179-1192, 2005.

[25] K.S. Narendra and M.A.L. Thathachar. Learning Automata: An Introduction. Prentice Hall, 1989.

[26] B.J. Oommen and T.D. Roberts. A Discretized Learning Automata Solutions to the Capacity Assignment Problem for Prioritized Networks. IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-32(B),pages: 821-831, 2002.

[27] B.J. Oommen and E.V. St.Croix. Graph partitioning using learning automata. IEEE Transactions on Computers, 45, 2, pages:195-208, IEEE, 1996.

[28] B.J. Oommen and D.C.Y. Ma. Deterministic Learning Automata Solutions to the Equipartitioning Problem. IEEE Transactions on Computers, 37,1, pages: 2-13, IEEE, 1988.

[29] B.J. Oommen and E.R. Hansen. List organizing strategies using stochastic move-to-front and stochastic move-to-rear operations. SIAM Journal on Computing, 16, SIAM, pages:705-716, 1987.

[30] B. Selman, H.A. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. Proceedings of AAAI'94, pages: 337-343. MIT Press, 1994.

[31] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. Proceedings of AAA92, pages: 440-446, MIT Press, 1992.

[32] K. Smyth, H. Hoos, T. *Stützle*, Iterated robust tabu search for MAX-SAT, Lecture Notes in Artificial Intelligence 2671, 129144, 2003.

[33] M.A.L. Thathachar and P.S. Sastry. Network of Learning Automata: Techniques for On line Stochastic Optimization. Kluer Academic Publishers, 2004.

[34] M.L. Tsetlin. Automaton Theory and Modeling of Biological Systems. Academic Press, 1973.

[35] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown. SATzilla: Portfolio-based Algorithm Se- lection for SAT. Journal of Artificial Intelligence Research (JAIR), 32, pp. 565-606, 2008.

[36] M. Yagiura, T. Ibaraki. Efficient 2 and 3-Flip Neighborhood Search Algorithms for the MAX SAT: Experimental Evaluation. Journal of Heuristics, 7: 423-442, 2001.

[37] L. Zhipeng, H. Jin-Kao. Adaptive Mmory-Based Local Search for MAX-SAT. Accept to Applied Soft Computing 2012.