ENHANCING LOCAL-SEARCH BASED SAT SOLVERS WITH LEARNING CAPABILITY

Ole-Christoffer Granmo
University of Agder, Grimstad, Norway

Noureddine Bouhmala
Vestfold University College, Vestfold, Norway

Keywords: Satisfiability problem, GSAT, Learning automata, Combinatorial optimization, Stochastic learning.

Abstract: The Satisfiability (SAT) problem is a widely studied combinatorial optimization problem with numerous applications, including time tabling, frequency assignment, and register allocation. Among the simplest and most effective algorithms for solving SAT problems are stochastic local-search based algorithms that mix greedy hill-climbing (exploitation) with random non-greedy steps (exploration). This paper demonstrates how the greedy and random components of the well-known GSAT Random Walk (GSATRW) algorithm can be enhanced with Learning Automata (LA) based stochastic learning. The LA enhancements are designed so that the actions that the LA chose initially mimic the behavior of GSATRW. However, as the LA explicitly interact with the SAT problem at hand, they learn the effect of the actions that are chosen, which allows the LA to gradually and dynamically shift from random exploration to goal-directed exploitation. Randomized and structured problems from various domains, including SAT-encoded Logistics Problems, and Block World Planning Problems, demonstrate that our LA enhancements significantly improve the performance of GSATRW, thus laying the foundation for novel LA-based SAT solvers.

1 INTRODUCTION

The conflict between exploration and exploitation is a well-known problem in machine learning and other areas of artificial intelligence. Learning Automata (LA) (Tsetlin, 1973; Thathachar and Sastry, 2004) capture the essence of this conflict, and have thus occupied researchers for over forty years. Initially, LA were used to model biological systems, however, in the last decades they have also attracted considerable interest because they can learn the optimal action when operating in unknown stochastic environments. Also, they combine rapid and accurate convergence with low computational complexity.

Recent applications of LA include allocation of polling resources in web monitoring (Granmo et al., 2007), allocation of limited sampling resources in binomial estimation (Granmo et al., 2007), and optimization of throughput in MPLS traffic engineering (Oommen et al., 2007).

LA solutions have furthermore found application within combinatorial optimization. In (Oommen and Ma, 1988; Gale et al., 1990) a so-called Object Migration Automaton is used for solving the classical equipartitioning problem. An order of magnitude faster convergence is reported compared to the best known algorithms at that time. A similar approach has also been discovered for the Graph Partitioning Problem(Oommen and Croix, 1996). Furthermore, the list organization problem has successfully been addressed by LA schemes, which have been found to converge to the optimal arrangement with probability arbitrary close to unity (Oommen and Hansen, 1987). Finally, in (Granmo and Bouhmala, 2007) we significantly improved the performance of traditional Random Walk (RW) for solving Satisfiability (SAT) problems, by embedding RW into the LA framework.

Inspired by the success of the above solution schemes, we will in this paper propose a novel scheme for solving SAT problems based on enhancing the classical GSAT Random Walk (GSATRW) algorithm with LA based learning capability.

Granmo O. and Bouhmala N. (2010).
ENHANCING LOCAL-SEARCH BASED SAT SOLVERS WITH LEARNING CAPABILITY.
In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence - Artificial Intelligence*, pages 515-521
DOI: 10.5220/0002707505150521

1.1 The Satisfiability (SAT) Problem

The SAT problem was among the first problems shown to be NP complete and involves determining whether an expression in propositional logic is true in *some* model (Cook, 1971). Thus, solving SAT problems efficiently is crucial for inference in propositional logic. Further, other NP complete problems, such as constraint satisfaction and graph coloring, can be encoded as SAT problems. Indeed, a large number of problems that occur in knowledge-representation, learning, VLSI-design, and other areas of artificial intelligence, are essentially SAT problems. Accordingly, improving the efficiency and accuracy of SAT solvers may benefit all of these areas.

Most SAT solvers use the Conjunctive Normal Form (CNF) representation for propositional logic expressions. In CNF, an expression is represented as a conjunction of *clauses*, with each clause being a disjunction of *literals*, and a literal being a *Boolean variable* or its negation. For example, the expression $P \vee \overline{Q}$ consists of one *single* clause, containing the two literals P and $\overline{Q}$. The literal P is simply a Boolean variable, while $\overline{Q}$ denotes the negation of the Boolean variable Q.

More formally, a SAT problem can be specified as follows. A propositional expression $\Phi = \bigwedge_{j=1}^{m} C_j$ 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***. Each clause $C_j, j \in \{1,\ldots,m\}$, in turn, is a disjunction of Boolean variables and has the form:

$$C_j = \left(\bigvee_{k \in I_j} x_k\right) \vee \left(\bigvee_{l \in \overline{I_j}} \overline{x_l}\right),$$

where $I_j, \overline{I_j} \subseteq \{1,.....n\}$, $I \cap \overline{I_j} = \emptyset$, and $\overline{x_i}$ denotes the negation of x_i.

The task is to determine whether there exists an assignment of truth 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. E.g., according to propositional logic, the expression $P \vee \overline{Q}$ becomes ***True*** for any truth assignment where P is ***True*** or Q is ***False***. Accordingly, the expression $P \vee \overline{Q}$ is satisfiable.

Note that since we have two choices for each of the n Boolean variables involved in a propositional expression Φ, the size of the search space S becomes $|S| = 2^n$. That is, the size of the search space grows exponentially with the number of variables.

1.2 Paper Organization

Among the simplest and most effective algorithms for solving SAT problems are stochastic local-search based algorithms that mix greedy hill-climbing (exploitation) with random non-greedy steps (exploration). This paper proposes how the greedy and random components of such local-search algorithms can be enhanced with LA-based stochastic learning.

In Section 2, we propose how the well-known GSATRW (Selman et al., 1994) scheme can be enhanced with learning capability using the basic concepts of LA. Then, in Section 3, we report the results obtained from testing the resulting new LA-based approach on an extensive test suit of problem instances. Finally, in Section 4 we present a summary of our work and provide ideas for further research.

2 SOLVING SAT PROBLEMS USING LEARNING AUTOMATA

This section demonstrates how the greedy and random components of SLS based algorithms can be enhanced with LA-based stochastic learning. We start by defining the basic building block of our scheme — the *Learning SAT Automaton* — before we propose how several such LA can form a *game* designed to solve SAT problems.

2.1 A Learning SAT Automaton

Generally stated, a 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 learning automaton is to find the action that minimizes the expected number of penalties received. Figure 1 illustrates the interaction between the learning automaton and the environment. Because we treat the environment as unknown, we will here only consider the definition of the learning automaton.

A learning automaton can be defined in terms of a quintuple (Narendra and Thathachar, 1989):

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

$\underline{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_s\}$ 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.

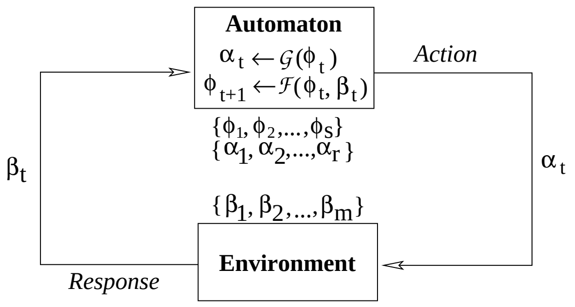
Automaton
$\alpha_t \leftarrow \mathcal{G}(\phi_t)$
$\phi_{t+1} \leftarrow \mathcal{F}(\phi_t, \beta_t)$
Action
$\{\phi_1, \phi_2, \ldots, \phi_s\}$
$\{\alpha_1, \alpha_2, \ldots, \alpha_r\}$
β_t
α_t
$\{\beta_1, \beta_2, \ldots, \beta_m\}$
Environment
Response

Figure 1: A learning automaton interacting with an environment.

Finally, a transition function $\phi_{t+1} = \mathcal{F}[\phi_t, \beta_t]$ determines the new automaton state from:

1. The current automaton state.
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 (Narendra and Thathachar, 1989; Thathachar and Sastry, 2004) for an extensive treatment.

We here target SAT problems, and our goal is to design a team of Learning Automata that seeks their solution. To achieve this goal, we build upon the work of Tsetlin and the linear two-action automaton (Tsetlin, 1973; Narendra and Thathachar, 1989) as described in the following.

First of all, for each literal in the SAT problem 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 $\mathcal{G}$ ma-

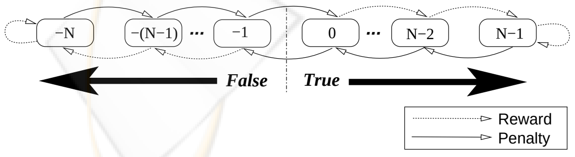

Figure 2: The state transitions and action selection of the Learning SAT Automaton.

trix 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 graph representation of $\mathcal{F}$ found 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.

2.2 Learning Automata GSATRW

In addition to the definition of the LA, we must define the environment that the LA interacts with. Simply put, the environment is a SAT problem as defined in Section 1. Each variable of the SAT problem 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 SAT problem at hand has also been solved.

With the above perspective in mind, we now present the details of the LA-GSATRW algorithm that we propose. Figure 3 contains the complete pseudocode for solving SAT problems, using a team of LA. As seen from the figure, LA-GSATRW alternates between selecting unsatisfied and satisfied clauses. When selecting an unsatisfied clause, an ordinary GSATRW strategy is used to penalize the LA when they "disagree" with GSATRW, i.e., when GSATRW and the LA suggest opposite truth values. Conversely, when selecting a satisfied clause, we use an "inverse" GSATRW strategy for rewarding the LA, this time when they agree with the "inverse" GSATRW. Note that as a result, the assignment of truth values to variables is indirect, governed by the states of the LA. To summarize, at the core of the LA-GSATRW algorithm is a punishment/rewarding scheme that guides the team of LA towards the optimal assignment, in the spirit of the underlying GSATRW strategy.

2.3 Comments to LA-GSATRW

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 ap-

```
Procedure learning_automata_gsat_random_walk()
Input : A set of clauses $\mathcal{C}$; Walk probability $p$ ;
Output : A satisfying truth assignment of the clauses, if found;
Begin
    /* Initialization */
    For i := 1 To n Do
        /* The initial state of each automaton is set to either '-1' or '1' */
        state[i] = random_element($\{-1,0\}$);
        /* And the respective literals are assigned corresponding truth values */
        If state[i] == -1 Then $x_i$ = False Else $x_i$ = True;
    /* Main loop */
    While Not stop($\mathcal{C}$) Do
            If $rnd(0,1) \leq p$ Then
               /* Draw unsatisfied clause randomly */
               $C_j$ = random_unsatisfied_clause($\mathcal{C}$);
               /* Draw clause literal randomly */
               i = random_element($I_j \cup \bar{I}_j$);
            Else
               /* Randomly select one of the literals whose flipping minimizes
               the number of unsatisfied clauses */
               i = random_element(Best_Literal_Candidates($\mathcal{C}$));
            /* The corresponding automaton is penalized for choosing the "wrong" action */
            If $i \in I_j$ And state[i] $< N-1$ Then
               state[i]++;
               /* Flip literal when automaton changes its action */
               If state[i] == 0 Then
                  flip($x_i$);
            Else If $i \in \bar{I}_j$ And state[i] $> -N$ Then
               state[i]--;
               /* Flip literal when automaton changes its action */
               If state[i] == -1 Then
                  flip($x_i$);

            If $rnd(0,1) \leq p$ Then
               /* Draw satisfied clause randomly */
               $C_j$ = random_satisfied_clause($\mathcal{C}$);
               /* Draw clause literal randomly */
               i = random_element($I_j \cup \bar{I}_j$);
            Else
               /* Randomly select one of the literals whose flipping maximizes
               the number of unsatisfied clauses */
               i = random_element(Worst_Literal_Candidates($\mathcal{C}$));
            /* Reward corresponding automaton if it */
            /* contributes to the satisfaction of the clause */
            If $i \in I_j$ And state[i] $\geq 0$ And state[i] $< N-1$ Then
               state[i]++;
            Else If $i \in \bar{I}_j$ And state[i] $< 0$ And state[i] $> -N$ Then
               state[i]--;
    EndWhile
End
```

Figure 3: Learning Automata GSAT Random Walk Algorithm.

pears.

Note that because multiple variables, and thereby multiple LA, may be involved in each clause, we are dealing with a game of LA (Narendra and Thathachar, 1989). 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-GSATRW, 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 (Narendra and Thathachar, 1989).

In our case, the LA involved in LA-GSATRW 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 at hand is equal to 1 when running the game infinitely. Also note that the solution of the SAT problem corresponds to a Nash equilibrium of the game.

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. Indeed, in this initial configuration the algorithm mimics its respective non-learning counterpart. 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 SAT problem at hand becomes increasingly focused.

3 EMPIRICAL RESULTS

We here compare LA-GSATRW with its non-learning counterpart — the GSAT with Random Walk (GSATRW) scheme. A main purpose of this comparison is to study the effect of the introduced stochastic learning. The benchmark problems we used to achieve this contain both randomized and structured problems from various domains, including SAT-encoded Bounded Model Checking Problems, Graph Coloring Problems, Logistics Problems, and Block World Planning Problems. Due to the random nature of the algorithms, when comparing LA-GSATRW with GSATRW, we run the algorithms 100 times, with each run being stopped after 10^7 flips.

3.1 Run-Length-Distributions (RLD)

As an indicator of the behavior of the algorithm on a single instance, we choose the median cost when trying to solve a given instance in 100 trials, and using an extremely high cutoff parameter setting of $Maxsteps = 10^7$ in order to obtain a maximal number of successful tries. To get an idea of the variability of the search cost, we analyzed the cumulative distribution of the number of search flips needed by both LA-GSATRW and GSATRW for solving single instances. For practical reasons we restrict our presentation here to the instances corresponding to small, medium, and large sizes from the underlying test-set.

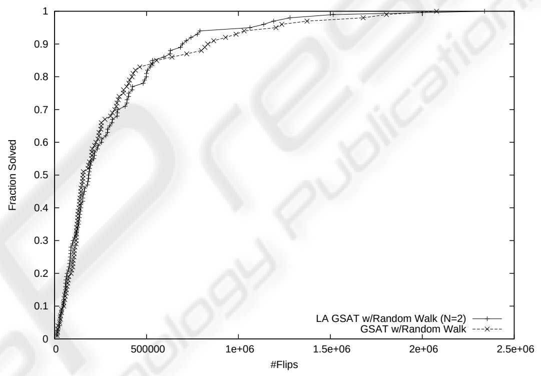

Figure 4: Cumulative distributions for a 600-variable random problem with 2550 clauses (f600).

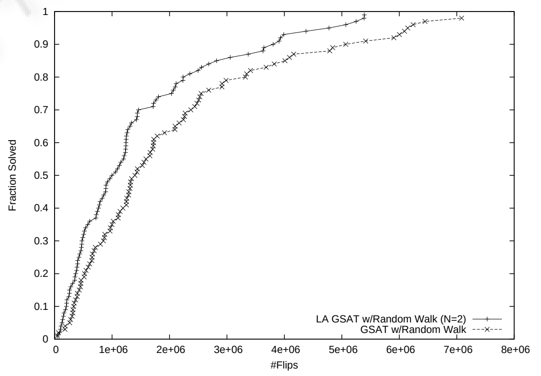

Figure 5: Cumulative distribution for a 1000-variable random problem with 4250 clauses (f1000).

Figures 4, 5, and 6 show RLDs obtained by applying LA-GSATRW and GSATRW to individual large random problems. As can be seen from the three plots, we observe that both algorithms reach a success rate of 100% for f600 and f1000. However, on the large problem f2000, GSATRW shows a low asymptotic solution probability corresponding to 0.37 compared to 0.45 for LA-GSATRW. Note also, that there

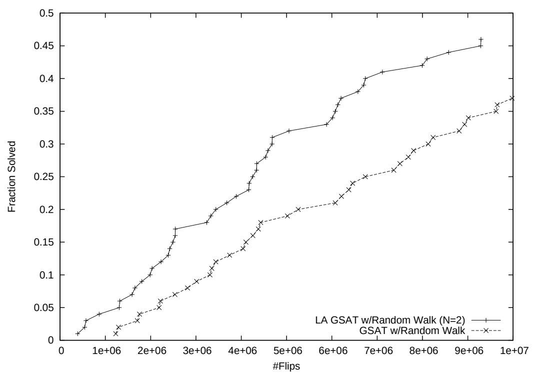

Figure 6: Cumulative distributions for a 2000-variables random problem with 8500 clauses (f2000).

is a substantial part of trials that are dramatically hard to solve, which explains the large variability in the length of the different runs of the two algorithms.

Both algorithms show the existence of an initial phase below which the probability for finding a solution is 0. Both methods start the search from a randomly chosen assignment which typically violates many clauses. Consequently, both methods need some time to reach the first local optimum which possibly could be a feasible solution. The two algorithms show no cross-over in their corresponding RLDs even though it is somewhat hard to see for f600, but it becomes more pronounced for f1000 and f2000. The median search cost for LA-GSATRW is 3%, 29%, and 17% of that of GSATRW for f600,f1000 and f2000 respectively. The three plots provides evidence for the superiority of LA-GSATRW compared to GSATRW as it gives consistently higher success probabilities while requiring fewer search steps than GSATRW.

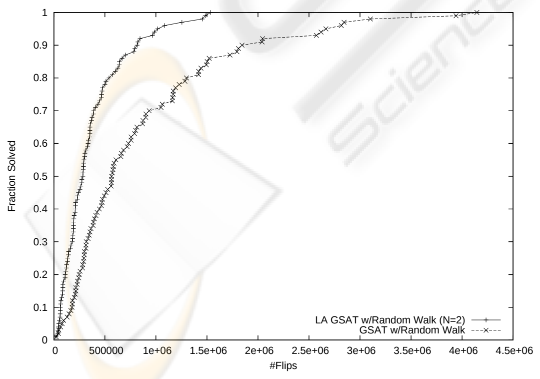

Figure 7: Cumulative distributions for a 228-variable logistics problem with 6718 clauses (logistics.a).

Figures 7, 8, 9, and 10 contains similar plots for SAT-encoded logistics problems. However, in this case it is difficult to claim a clear winner among the

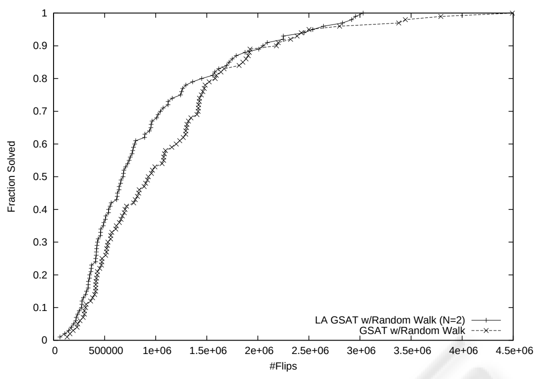

Figure 8: Cumulative distribution for a 843-variable logistics problem with 7301 clauses (logistics.b).

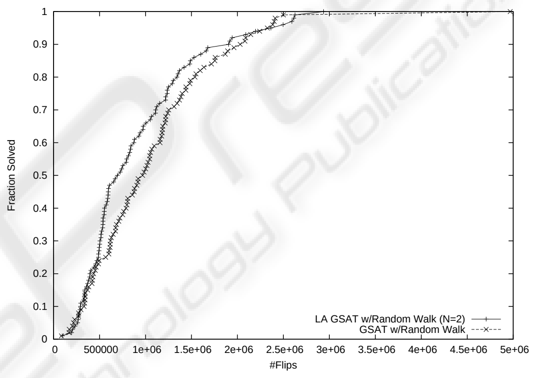

Figure 9: Cumulative distributions for a 1141-variable logistics problem with 10719 clauses (logistics.c).

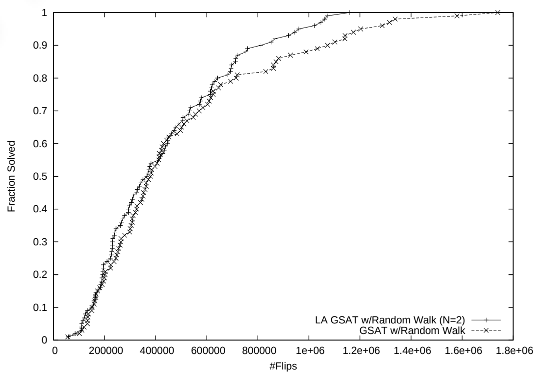

Figure 10: (Left) Cumulative distribution for a 4713-variable logistics problem with 21991 clauses (logistics.d).

algorithms. The number of search steps varies between the different trials and is significantly higher with GSATRW than that of LA-GSATRW. However, note that the median search cost for LA-GSATRW is 4%, 29%, 34% and 51% of that of GSATRW for Logistics-d, Logistics-b, Logistics-c, and Logistics-a.

3.2 Wilcoxon Rank-Sum Test

Table 1 summarizes Wilcoxon Rank-Sum Test results for a number of SAT encoded problems. It table reveals two pertinent observations. Firstly, the success rate of LA-GSATRW was better in 10 problems and this difference in the median search cost was significant in 6 of the problems. On the other hand, GSATRW gave better results in 2 problems in terms of success rate, but this performance difference was significant in only *one* case.

Table 1: Success rate (SR) and Wilcoxon statistical test.

Problem	LA	GSAT	P value	Null-H.
f600	53%	47%	0.19	Accept
f1000	62%	37%	0.00	Reject
f2000	32%	14%	0.00	Reject
logistic-a	74%	26%	0.00	Reject
logistic-b	54%	46%	0.09	Accept
logistic-c	59%	41%	0.02	Reject
logistic-d	54%	46%	0.29	Accept
bw-medium	36%	64%	0.02	Reject
bw-large-a	49%	51%	0.52	Accept
bw-huge	50%	50%	0.91	Accept
bw-large-b	53%	47%	0.82	Accept
bmc-ibm2	39%	61%	0.01	Reject
bmc-ibm3	52%	44%	0.18	Accept
bmc-ibm6	51%	49%	0.98	Accept

4 CONCLUSIONS AND FURTHER WORK

In this work, we have introduced a new approach based on combining Learning Automata and GSAT w/Random Walk. The success rate of LA-GSATRW was better in 10 of the benchmark problems used, and the difference in the median search cost was significantly better for 6 of the problems. GSASTRW, on the other hand, gave better results in 2 of the problems in terms of success rate, while its performance was significantly better for only one of these problems.

Based on the empirical results, it can be seen that the Learning Automata mechanism employed in LA-GSATRW offers an efficient way to escape from highly attractive areas in the search space, leading to a higher probability of success as well as reducing the number of local search steps to find a solution.

As further work, it is of interest to study how Learning Automata can be used to enhance other Stochastic Local Search based algorithms. Furthermore, more recent classes of Learning Automata, such as the Bayesian Learning Automata family (Granmo, 2009) may offer improved performance in LA based SAT solvers.

REFERENCES

Cook, S. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third ACM Symposuim on Theory of Computing*, pages 151–158.

Gale, W., S.Das, and Yu, C. (1990). Improvements to an Algorithm for Equipartitioning. *IEEE Transactions on Computers*, 39:706–710.

Granmo, O.-C. (2009). Solving Two-Armed Bernoulli Bandit Problems Using a Bayesian Learning Automaton. *To Appear in International Journal of Intelligent Computing and Cybernetics (IJICC)*.

Granmo, O.-C. and Bouhmala, N. (2007). Solving the Satisfiability Problem Using Finite Learning Automata. *International Journal of Computer Science and Applications*, 4:15–29.

Granmo, O.-C., Oommen, B. J., Myrer, S. A., and Olsen, M. G. (2007). Learning Automata-based Solutions to the Nonlinear Fractional Knapsack Problem with Applications to Optimal Resource Allocation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(1):166–175.

Narendra, K. S. and Thathachar, M. A. L. (1989). *Learning Automata: An Introduction*. Prentice Hall.

Oommen, B. J. and Croix, E. V. S. (1996). Graph partitioning using learning automata. *IEEE Transactions on Computers*, 45(2):195–208.

Oommen, B. J. and Hansen, E. R. (1987). List organizing strategies using stochastic move-to-front and stochastic move-to-rear operations. *SIAM Journal on Computing*, 16:705–716.

Oommen, B. J. and Ma, D. C. Y. (1988). Deterministic learning automata solutions to the equipartitioning pr oblem. *IEEE Transactions on Computers*, 37(1):2–13.

Oommen, B. J., Misra, S., and Granmo, O.-C. (2007). Routing Bandwidth Guaranteed Paths in MPLS Traffic Engineering: A Multiple Race Track Learning Approach. *IEEE Transactions on Computers*, 56(7):959–976.

Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise Strategies for Improving Local Search. In *Proceedings of AAAI'94*, pages 337–343. MIT Press.

Thathachar, M. A. L. and Sastry, P. S. (2004). *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers.

Tsetlin, M. L. (1973). *Automaton Theory and Modeling of Biological Systems*. Academic Press.