We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

5

Using Learning Automata to Enhance Local-Search Based SAT Solvers with Learning Capability

Ole-Christoffer Granmo and Noureddine Bouhmala
University of Agder; Vestfold University College
Norway

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) Thathachar & Sastry (2004); Tsetlin (1973) 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. Furthermore, 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 problems. In Gale et al. (1990); Oommen & Ma (1988) 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 ProblemOommen & Croix (1996). Finally, the list organization problem has successfully been addressed by LA schemes. These schemes have been found to converge to the optimal arrangement with probability arbitrary close to unity Oommen & Hansen (1987).

In this chapter we study a new application domain for LA, namely, the Satisfiability (SAT) problem. In brief, we demonstrate how the learning capability of LA can be incorporated into selected classical local-search based solvers for SAT-like problems, with the purpose of allowing improved performance by means of learning. Furthermore, we provide a detailed empirical evaluation of the resulting LA based algorithms, and we analyze the effect that the introduced learning has on the local-search based solvers.

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. It is accordingly the case that SAT solver improvements will have a direct impact in all of these areas.

Most SAT solvers use a Conjunctive Normal Form (CNF) representation of propositional logic expressions. In CNF, an expression in propositional logic 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 $P \vee \bar{Q}$ consists of one *single* clause, containing the two literals P and $\bar{Q}$. P is simply a Boolean variable and $\bar{Q}$ denotes the negation of the Boolean variable Q. Thus, according to propositional logic, the expression $P \vee \bar{Q}$ becomes ***True*** if either P is ***True*** or Q is ***False***.

More formally, a SAT problem can be defined 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:

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

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 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. Note that since we have two choices for each of the n Boolean variables, 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 Chapter Contributions

Among the simplest and most effective algorithms for solving SAT problems are local-search based algorithms that mix greedy hill-climbing (exploitation) with random non-greedy steps (exploration). This chapter demonstrates how the greedy and random components of such local-search algorithms can be enhanced with LA-based stochastic learning. We will use both pure Random Walk as well as the well-known GSAT algorithm Selman et al. (1994), combined with Random Walk, as demonstration algorithms. The LA enhancements are designed so that the actions that the LA chose initially mimic the behavior of GSAT/Random Walk. 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.

We finally provide a detailed comparative analysis of the new LA based algorithms' performance, showing the effect that the introduced stochastic learning has on the enhanced local-search based algorithms. The benchmark set used contains randomized and structured problems from various domains, including SAT-encoded Bounded Model Checking Problems, Logistics Problems, and Block World Planning Problems.

1.3 Chapter Organization

The chapter is organized as follows. In section 2 we provide a brief overview of selected algorithms for the SAT problem. Furthermore, we take a closer look at the Random Walk

and GSAT with Random Walk algorithms, before we in section 3 explain how these latter algorithm can be enhanced with learning capability, using the basic concepts of LA. In section 4, we report the results obtained from testing the resulting new approaches on an extensive test suit of problem instances. Finally, in section 5 we present a summary of our work and provide ideas for further research.

2. Methods for SAT

The SAT has been extensively studied due to its simplicity and applicability. The simplicity of the problem coupled with its intractability makes it an ideal platform for exploring new algorithmic techniques. This has led to the development of several algorithms for solving SAT problems which usually fall into two main categories: systematic algorithms and local search algorithms. We hereby undertake the task of describing selected algorithms from these two categories.

2.1 Systematic Search Algorithms

Systematic search algorithms are guaranteed to return a satisfying truth assignment to a SAT problem if one exists and prove that it is unsatisfiable otherwise. The most popular and efficient systematic search algorithms for SAT are based on the Davis-Putnam (DP)Davis & Putnam (1960) procedure, which enumerates all possible variable assignments. This is achieved by constructing a binary search tree, which is expanded until one either finds a satisfying truth assignment or one can conclude that no such assignment exists. In each recursive call of the algorithm the propositional formula Φ is simplified by means of *unit propagation*. That is, a Boolean variable x_i is selected according to a predefined rule from the n Boolean variables available. Next, all the clauses that include the literal x_i are found, and the literal is deleted from all of these clauses. Let $\mathcal{C} = \{C_1, C_2, ...C_k\}$ be the set of $k(<= m)$ clauses obtained from this process. Similarly, let $\mathcal{D} = \{D_1, D_2, ...D_r\}$ denotes the set of $l(<= m)$ clauses obtained after deleting the literal $\bar{x}_i$ in the same manner. Moreover, let $\mathcal{R} = \{R_1, R_2, ...R_{(m-k-r)}\}$ represent the set (*m-k-r*) of clauses that does not include any of these two literals. Then, the original proportional formula is reduced to:

$$\Phi_{simpler} = \left(\bigwedge_{i=1}^{k} \bigwedge_{j=1}^{r} (A_i \vee B_j) \right) \bigwedge_{l=1}^{(m-r-k)} R_l.$$

Note that the propositional formula $\Phi_{simpler}$ does not contain the Boolean variable x_i because none of C, D or R does (by way of construction). If thus an empty clause is obtained, the current partial assignment cannot be extended to a satisfying one and backtracking is used to proceed with the search; if an empty formula is obtained, i.e., all clauses are satisfied, the algorithm returns a satisfying assignment. If neither of these two situations occur, an unassigned variable is chosen and the algorithm is called recursively after adding a unit clause containing this variable and its negation. If all branches are explored and no satisfying assignment has been reached, the formula is found to be unsatisfiable. For efficiency reasons, the search tree is explored in depth first search manner. Since we are only interested in whether the SAT problem is satisfiable or not, we stop as soon as the first solution is found. The size of the search tree depends on the branching rule adopted (how to select the branch variable) thereby affecting the overall efficiency of DP. This has led to the development of various improved DP variants which differ in the schemes employed to maximize the efficiency of unit propagation in their branching rules.

2.2 Stochastic Local Search Algorithms (SLS)

The above indicated class of algorithms can be very effective on specific classes of problems, however, when problems scales up, their solution effectiveness typically degrades in an exponential manner. Indeed, due to their combinatorial explosive nature, large and complex SAT problems are hard to solve using systematic search algorithms. One way to overcome the combinatorial explosion is to abandon the goal of systematically conducting a complete search.

2.2.1 Local Search as Iterative Optimization

Local search algorithms are based on what is perhaps the oldest optimization method — *trial and error*. Typically, they start with an initial assignment of truth values to variables, randomly or heuristically generated. The SAT problem can then be reformulated as an iterative optimization problem in which the goal is to minimize the number of unsatisfied clauses (the objective function). Thus, the optimum is obtained when the value of the objective function equals zero, which means that all clauses are satisfied. During each iteration, a new value assignment is selected from the "neighborhood" of the present one, by performing a "move". Most local search algorithms use a 1-flip neighborhood relation, which means that two truth value assignments are considered to be neighbors if they differ in the truth value of *only* one variable. Performing a move, then, consists of switching the present value assignment with one of the neighboring value assignments, e.g., if the neighboring one is better (as measured by the objective function). The search terminates if no better neighboring assignment can be found. Note that choosing a fruitful neighborhood, and a method for searching it, is usually guided by intuition — theoretical results that can be used as guidance are sparse.

2.2.2 GSAT, GSAT with Random Walk, and WalkSAT

One of the most popular local search algorithms for solving SAT is GSAT Selman et al. (1992). Basically, GSAT begins with a random generated assignment of truth values to variables, and then uses a so-called steepest descent heuristic to find the new variable-value assignment, i.e., the 1-flip neighbor with the *least* number of unsatisfied clauses is always selected as the new truth assignment. After a fixed number of such moves, the search is restarted from a new random assignment. The search continues until a solution is found or a fixed number of restarts have been performed. An extension of GSAT, referred to as random-walk Selman et al. (1994) has been realized with the purpose of escaping from local optima. In a random walk step, a randomly unsatisfied clause is selected. Then, one of the variables appearing in that clause is flipped, thus effectively forcing the selected clause to become satisfied. The main idea is to decide at each search step whether to perform a standard GSAT or a random-walk strategy with a probability called the *walk probability*. Another widely used variant of GSAT is the WalkSAT algorithm originally introduced in McAllester et al. (1997). It first picks randomly an unsatisfied clause, and then, in a second step, one of the variables with the lowest *break count*, appearing in the selected clause, is randomly selected. The break count of a variable is defined as the number of clauses that would be unsatisfied by flipping the chosen variable. If there exists a variable with break count equals to zero, this variable is flipped, otherwise the variable with minimal break count is selected with a certain probability. It turns out that the choice of unsatisfied clauses, combined with the randomness in the selection of variables, enable WalkSAT and GSAT with random walk to avoid local minima and to better explore the search space.

2.3 Weight-based Schemes

Recently, new algorithms Gent & T.Walsh (1993); Glover (1989); Hansen & Jaumand (1990); I.Gent & Walsh (1995) have emerged using history-based variable selection strategies in order to avoid flipping the same variable repeatedly. Apart from GSAT and its variants, several clause weighting based SLS algorithms Cha & Iwama (1995)Frank (1997) have been proposed to solve SAT problems. The key idea is to 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. In essence, clause weighting acts as a diversification mechanism rather than a way of escaping local minima. Finally, many other generic SLS algorithms have been applied to SAT. These includes techniques such as Simulated Annealing Spears (1993), Evolutionary Algorithms A.E.Eiben & van der Hauw (1997), and Greedy Randomized Adaptive Search Procedures Johnson & Trick (1996).

3. Solving SAT Problems Using Learning Automata

This section demonstrates how the greedy and random components of local-search algorithms can be enhanced with LA-based stochastic learning. We will use both pure Random Walk and GSAT with Random Walk, as demonstration algorithms. 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.

3.1 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

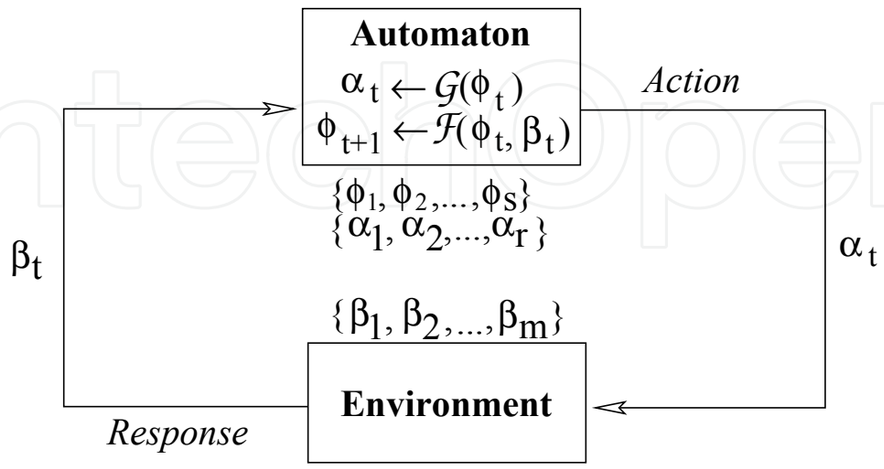


Fig. 1. A learning automaton interacting with an environment

quintuple Narendra & 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. 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 & Thathachar (1989); Thathachar & Sastry (2004) for an extensive treatment.

We now target the SAT problem, and our goal is to design a team of Learning Automata that seeks the solution of SAT problems. To achieve this goal, we build upon the work of Tsetlin and the linear two-action automaton Narendra & Thathachar (1989); Tsetlin (1973) 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: $\underline{\alpha} = \{\textbf{\textit{True}}, \textbf{\textit{False}}\}$.
- 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

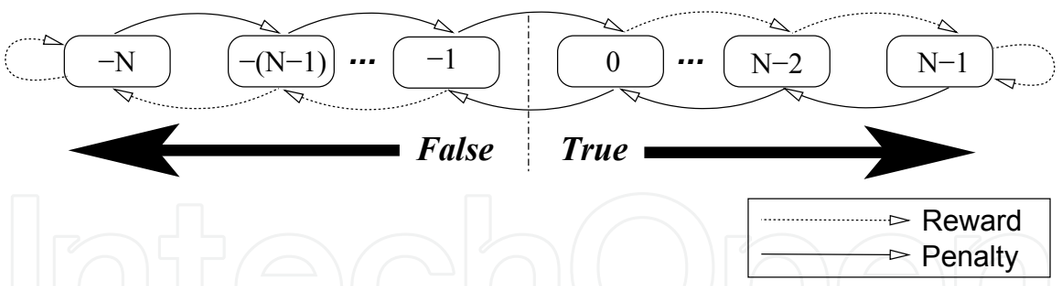


Fig. 2. The state transitions and action selection of the Learning SAT Automaton

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.

3.2 Learning Automata Random Walk (LARW)

```
Procedure learning_automata_random_walk()

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(C) Do
            /* Draw unsatisfied clause randomly */
            C_j = random_unsatisfied_clause(C);
            /* Draw clause literal randomly */
            i = random_element(I_j ∪ Ī_j);
            /* The corresponding automaton is penalized for choosing the "wrong" action */
            If i ∈ 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 ∈ Ī_j And state[i] > −N Then
               state[i]−−;
               /* Flip literal when automaton changes its action */
               If state[i] == -1 Then
                  flip(x_i);

            /* Draw satisfied clause randomly */
            C_j = random_satisfied_clause(C);
            /* Draw clause literal randomly */
            i = random_element(I_j ∪ Ī_j);
            /* Reward corresponding automaton if it */
            /* contributes to the satisfaction of the clause */
            If i ∈ I_j And state[i] ≥ 0 And state[i] < N − 1 Then
               state[i]++;
            Else If i ∈ Ī_j And state[i] < 0 And state[i] > −N Then
               state[i]−−;
    EndWhile
End
```

Fig. 3. Learning Automata Random Walk (LARW) Algorithm

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.

3.3 Learning Automata Random Walk (LARW)

With the above perspective in mind, we will now present the details of the LARW that we propose. Figure 3 contains the complete pseudo-code for solving SAT problems, using a team of LA. As seen from the figure, the LARW corresponds to an ordinary Random Walk, however, both satisfied and unsatisfied clauses are used in the search. Furthermore, the assignment of truth values to variables is indirect, governed by the states of the LA. At the core of the LARW 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. To elaborate, in each iteration of the algorithm, we randomly select a single clause. A variable is randomly selected from that clause, and the corresponding automaton is identified. If the clause is unsatisfied, the automaton is punished. Correspondingly, if the clause is satisfied, the automaton is rewarded, however, only if the automaton makes the clause satisfied. As also seen, the algorithm alternates between selecting satisfied and unsatisfied clauses.

3.4 Learning Automata GSATRW(LA-GSATRW)

Based on the same underlying principles that motivates the LARW, we will now present the details of the LA-GSATRW that we propose. Figure 4 contains the complete pseudo-code for solving SAT problems, using a team of LA. As seen from the figure, an ordinary GSATRW strategy is used to penalize an LA when it "disagrees" with GSATRW, i.e., when GSATRW and the LA suggest opposite truth values. Additionally, we use an "inverse" GSATRW strategy for rewarding an LA when it agrees with GSATRW. Note that as a result, the assignment of truth values to variables is indirect, governed by the states of the LA. Again, at the core of the LA-GSATRW algorithm is a punishment/rewarding scheme that guides the team of LA towards the optimal assignment. However, in this algorithm, the guidance is based on GSATRW rather than pure RW.

3.5 Comments to LARW and 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 appears.

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

In our case, the LA involved in the LARW are non-absorbing, i.e., every state can be reached from every other state with positive probability. This means that the probability of reaching

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] ≥ 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

Fig. 4. Learning Automata GSAT Random Walk Algorithm

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 both of the algorithms mimics their respective non-learning counterparts. 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.

4. Empirical Results

We here compare LARW and LA-GSATRW with their non-learning counterparts — the Random Walk (RW) and the GSAT with Random Walk (GSATRW) schemes. 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.

4.1 LARW Vs RW

As a basis for the empirical comparison of RW and LARW, we selected a benchmark test suite of 3-colorable graphs that shows so-called *phase transition*. All the instances are known to be hard and difficult to solve and are available from the SATLIB website (http://www.informatik.tu-darmstadt.de/AI/SATLIB). The benchmark instances we use are satisfiable and have been used widely in the literature.
Note that due to the stochastic nature of LARW, the number of flips required for solving a SAT problem varies widely between different runs. Therefore, for each problem, we run LARW and RW 100 times each, with a cutoff (maximum number of flips) which is sufficient (10^7) to guarantee a success rate close to 100%.

4.1.1 Search Trajectory

The manner in which each LA converges to an assignment is crucial for better understanding LARW's behavior. In Figure 5 we show how the best and current assignment progress during the search using a random 3-SAT problem with 150 variables and 645 clauses, taken from the SAT benchmark library.
The plot to the left in Figure 5 suggests that problem solving with LARW happens in two phases. In the first phase, which corresponds to the early part of the search (the first 5% of the search), LARW behaves as a hill-climbing method. In this phase, which can be described as a short one, up to 95% of the clauses are satisfied. The currently best score climbs rapidly at first, and then flattens off as we mount a plateau, marking the start of the second phase. The plateau spans a region in the search space where flips typically leave the best assignment unchanged. The long plateaus becomes even more pronounced as the number of flips increases. More specifically, the plateau appears when trying to satisfy the last few remaining clauses.
To further investigate the behavior of LARW once on the plateau, we looked at the corresponding average state of the LA as the search progresses. The plot to the right in Figure 5 shows the resulting observations. At the start of plateau, search coincides in general with an increase in the average state. The longer the plateau runs, the higher the average state

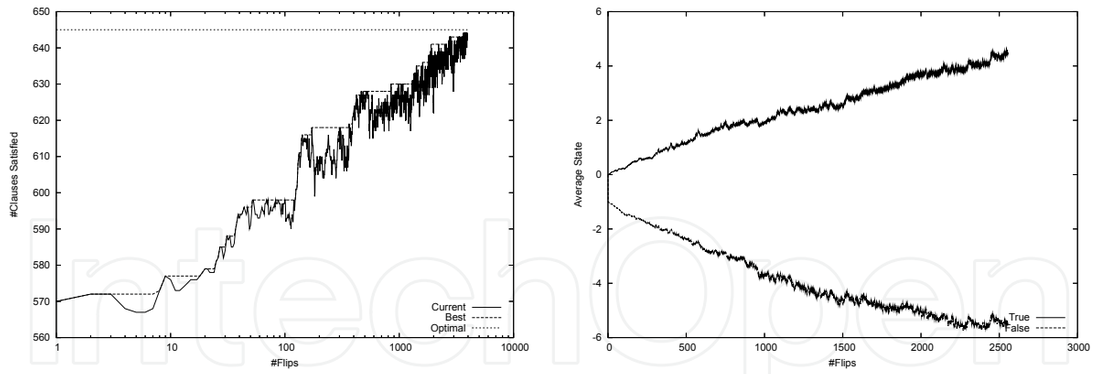


Fig. 5. (Left) LARW's search space on a 150 variable problem with 645 clauses (uf150-645). Along the horizontal axis we give the number of flips, and along the vertical axis the number of satisfied clauses. (Right) Average state of LA. Horizontal axis gives the number of flips, and the vertical axis shows the average state of automaton.

becomes. An automaton with high average state needs to perform a series of actions before its current state changes to either -1 or 0, thereby making the flipping of the corresponding variable possible. The transition between each plateau corresponds to a change to the region where a small number of flips gradually improves the score of the current solution ending with an improvement of the best assignment. The search pattern brings out an interesting difference between LARW and the standard use of SLS. In the latter, one generally stops the search as soon as no more improvements are found. This can be appropriate when looking for a near-optimal solution. On the other hand, when searching for a global maximum (i.e., a satisfying assignment) stopping when no flip yields an immediate improvement is a poor strategy.

4.1.2 Run-Length-Distributions (RLDs)

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. The reason behind choosing the median cost rather than the mean cost is due to the large variation in the number of flips required to find a solution. 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 LARW and RW for solving single instances. Due to non-deterministic decisions involved in the algorithms (i.e., initial assignment, random moves), the number of flips needed by both algorithms to find a solution is a random variable that varies from run to run. More formally, let k denotes the total number of runs, and let $f'(j)$ denotes the number of flips for the j-th successful run (i.e, run during which a solution is found) in a list of all successful runs, sorted according to increasing number of flips, then the cumulative empirical RLD is defined by $\hat{P}(f'(j) \leq f) = \frac{|\{j|f'(j) \leq f\}|}{k}$. For practical reasons we restrict our presentation here to the instances corresponding to small, medium, and large sizes from the underlying test-set.
Figures 6 and 7 show RLDs obtained by applying RW and LARW to individual SAT-encoded graph coloring problem instances. As can be seen from the leftmost plot in Figure 6, we observe that on the small size instance, the two algorithms show no cross-over in their corre-

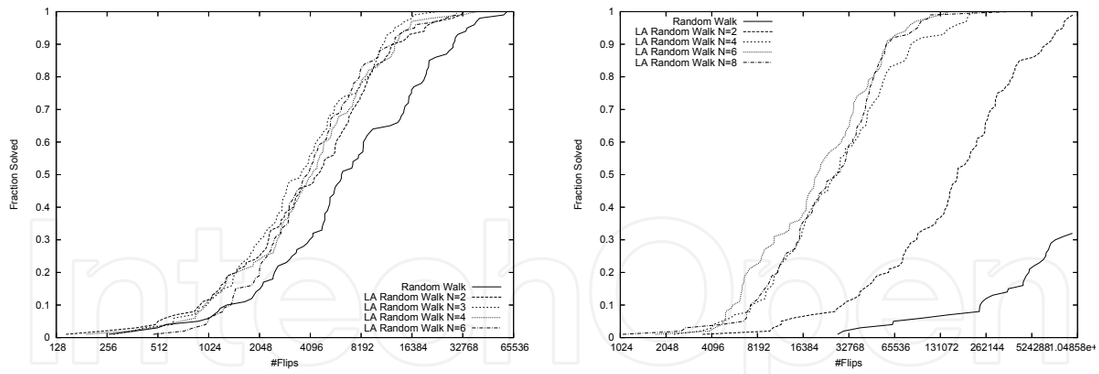


Fig. 6. (Left) Cumulative distributions for a 90-variable *graph coloring* problems with 300 clauses (flat90-300). (Right) Cumulative distribution for a 150-variable *graph coloring* problem with 545 clauses (flat375-1403). Along the horizontal axis we give the number of flips, and along the vertical axis the fraction of problems solved for different values of N.

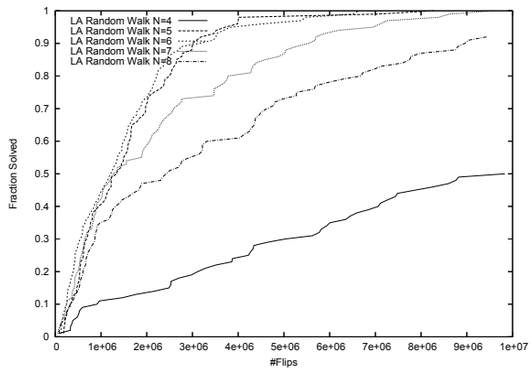


Fig. 7. Cumulative distributions for a 375-variable *graph coloring* problem with 1403 clauses (flat375-1403). Along the horizontal axis we give the number of flips, and along the vertical axis the fraction of problems solved for different values of N.

sponding RLDs. This provides evidence for the superiority of LARW compared to RW (i.e, $N = 1$) as it gives consistently higher success probabilities, regardless of the number of search steps.

On the medium sized instance, to the right in Figure 6, we observe a stagnation behavior with a low asymptotic solution probability corresponding to a value around 0.3. As can be easily seen, both methods 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 plot in Figure 7 shows that the performance of RW for the large instance (flat375-1403) degrades. Indeed, the probability of finding a feasible solution within the required number of steps is 0. Further, note that the distance between the minimum and the maximum number of search steps needed for finding a solution using RW is higher compared to that of LARW

and increases with the hardness of the instance. The learning automaton mechanism pays off as the instance gets harder. Finally, observe that the probability of success gets higher as N increases, to a certain level.

4.1.3 Mean Search Cost

In this section, we focus on the behavior of the two algorithms using 100 instances from a test-set of small and medium sized problem instances. We chose not to include the plot for the large instance (flat375-1403) because RW was incapable of solving it during the 100 trials. For each instance the median search cost (number of local search steps) is measured and we analyze the distribution of the mean search cost over the instances from each test-set. The different plots show the cumulative hardness distributions produced by 100 trials on 100 instances from a test-set.

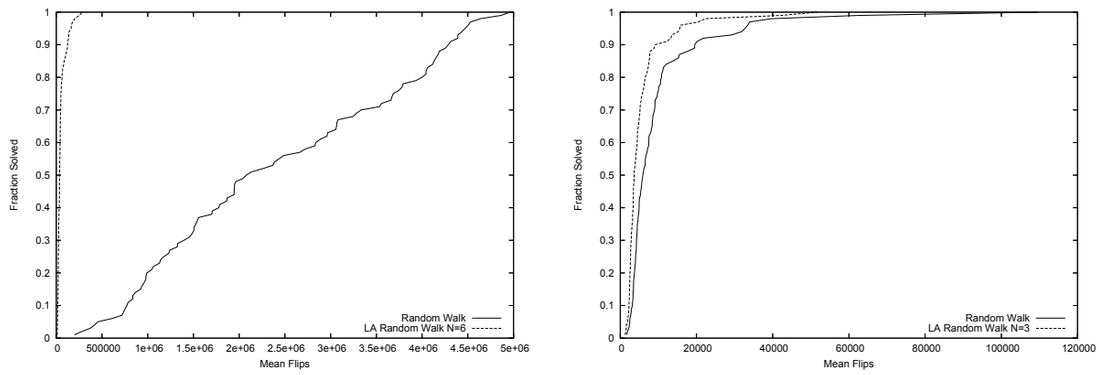


Fig. 8. (Left) Hardness distribution across test-set flat150-545. (Right) Hardness distribution across test-set for flat90-300. Along the horizontal axis we give the median number of flips per solution, and along the vertical axis the fraction of problems solved.

Several observations can be made from the plots in Figure 8, which show the hardness distributions of the two algorithms for SAT-encoding graph coloring problem instances. There exists no cross-overs in the plots in either of the figures, which makes LARW the clear winner. Note also RW shows a higher variability in search cost compared to LARW between the instances of each test-set.
The distributions of the two algorithms confirm the existence of instances which are harder to solve than others. In particular, as can be seen from the long tails of these distributions, a substantial part of problem instances are dramatically harder to solve with RW than with LARW. The harder the instance, the higher the difference between the average search costs of two algorithms (a factor of approximately up to 50). This can be explained by the fact that the automaton learning mechanism employed in LARW offers an efficient way to escape from highly attractive areas in the search space of hard instances leading to a higher probability of success, as well as reducing the average number of local search steps needed to find a solution. The empirical hardness distribution of SAT-encoded graph coloring problems to the right in Figure 8 shows that it was rather easy for both algorithms to find a feasible solution in each trial across the test set flat90-300, with LARW showing on average a lower search cost within a given probability compared to RW. The plot reveals the existence of some instances on which RW suffers from a strong search stagnation behavior.

The plot located to the left in Figure 8 shows a striking poor average performance of RW compared to LARW on the test set flat150-545. Conversely, LARW shows a consistent ability to find solutions across the instances on this test set. For LARW, we observe a small variability in search cost indicated by the distance between the minimum and the maximum number of local search steps needed to find a solution. The differences in performance between these two algorithms can be characterized by a factor of about 10 in the median. The performance differences observed between the two algorithms for small size instances are still observable and very significant for medium size instances. This suggests that LARW in general is considerably more effective for larger instances.

4.2 LA-GSATRW Vs GSATRW

Since LA-GSATRW is more sophisticated and far more effective than LARW, we used larger and harder problem instances to evaluate LA-GSATRW. In brief, we selected a benchmark suite from different domains including problem instances from the Bejing SAT competition held in 1996. Again, due to the random nature of the algorithms, when comparing LA-GSATRW with GSATRW, we run the algorithms 100 times using a maximum number of flips of 10^7 as a stopping criteria (guaranteeing a success rate close to 100%).

4.2.1 Search Space

The manner in which LA converges on assignment is crucial to a better understanding of LA-GSATRW behaviour. In Figure 9, we show how the best found score and the current score progress during the search on a SAT-encoded logistics problem.

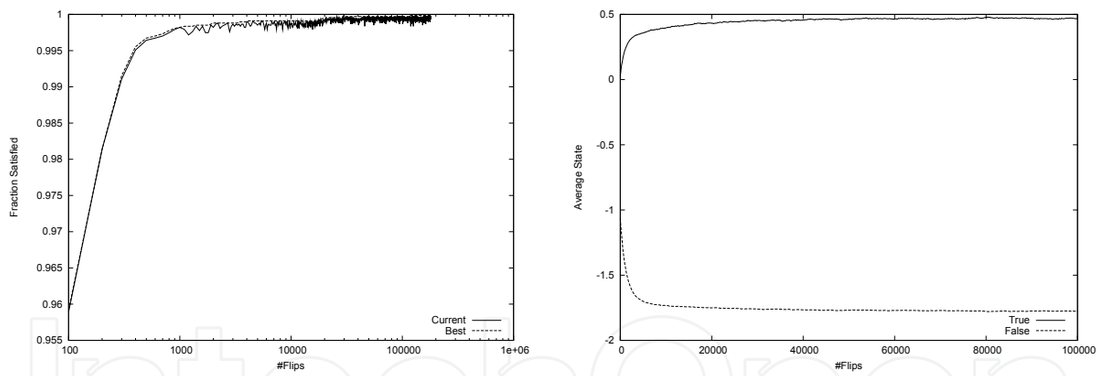


Fig. 9. (Left) LA-GSATRW's search space on a 828 variable problem with 6718 clauses (logistics.a). Along the horizontal axis we give the number of flips, and along the vertical axis the number of satisfied clauses. (Right) Average state of LA. Horizontal axis gives the number of flips, and the vertical axis shows the average state of automaton.

The leftmost plot suggests that problem solving with LA-GSATRW also happens in two phases. Again, in the first phase which corresponds to the early part of the search (the first 5% of the search) LA-GSATRW behaves as a hill-climbing method. In this phase, which can be described as a short one, up to 95% of the clauses are satisfied. The best obtained score climbs rapidly at first, and then flattens off as we reach a plateau, marking the start of the second phase. The plateau spans a region in the search space where flips typically leave the best

assignment unchanged. The long plateaus becomes even more pronounced as the number of flips increases, and occurs more specifically in trying to satisfy the last few remaining clauses. To further investigate the behaviour of LA-GSATRW once on the plateau, we looked at the corresponding average state of automaton as the search progresses. The rightmost plot in Figure 9 shows the reported observations. The start of plateau search coincides in general with an increase in the average state. The longer plateau, the higher average state.An automaton with high average state needs to perform a series of actions before its current state changes to either -1 or 0, thereby making the flipping of the corresponding variable possible. The transition between each plateau corresponds to a change to the region where a small number of flips gradually improves the score of the current solution ending with an improvement of the best assignment.

4.2.2 Run-Length-Distributions (RLD)

In order to observe the variability of the search cost of GSATRW and LA-GSATRW, we analyzed the cumulative distribution of the number of search flips needed by the algorithms, as defined in Section 4.1.2.

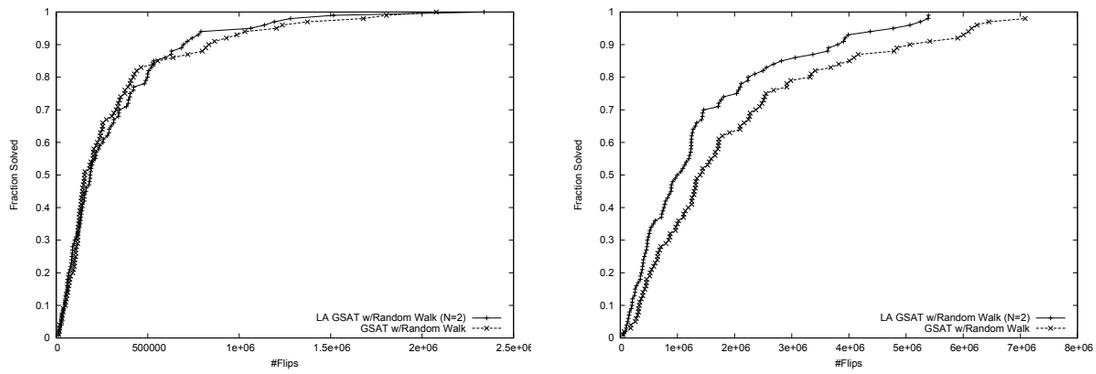


Fig. 10. Cumulative distributions for a 600-variable *random* problem with 2550 clauses (f600). (Right) Cumulative distribution for a 1000-variable *random* problem with 4250 clauses (f1000). Along the horizontal axis we give the number of flips, and along the vertical axis the the success rate.

Figures 10 and 11 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 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. Again, the 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

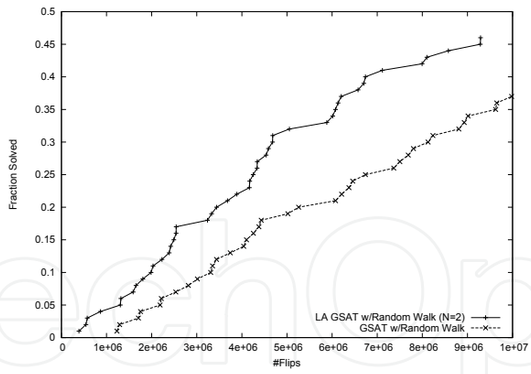


Fig. 11. (Left) Cumulative distributions for a 2000-variables random problem with 8500 clauses (f2000). Along the horizontal axis we give the number of flips, and along the vertical axis the success rate.

as it gives consistently higher success probabilities while requiring fewer search steps than GSATRW.

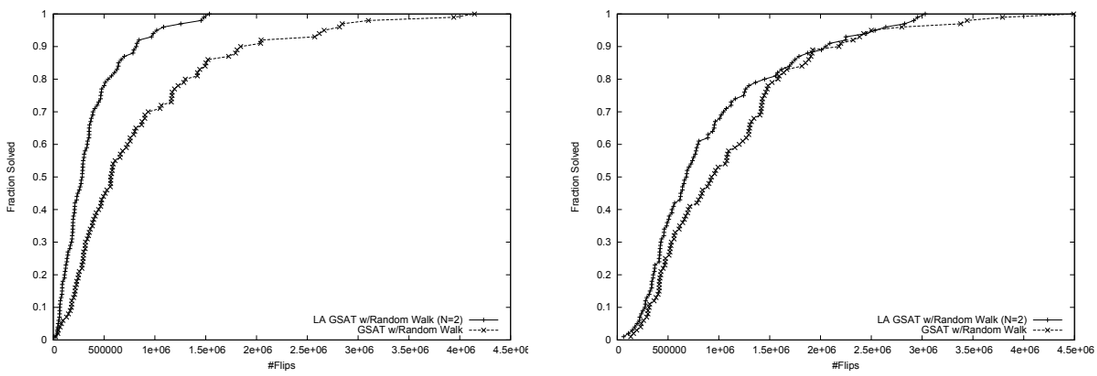


Fig. 12. (Left) Cumulative distributions for a 228-variable logistics problem with 6718 clauses (logistics.a). (Right) Cumulative distribution for a 843-variable logistics problem with 7301 clauses (logistics.b). Along the horizontal axis we give the number of flips, and along the vertical axis the success rate.

Figure 12 and 13 contains similar plots for SAT-encoded logistics problems. However, in this case it is difficult to claim a clear winner among the 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, respectively.
We now turn to single SAT-encoded instances from the Blocks World Planning domain. The crossing of the two RLDs at different points, as shown in figures 1516, indicates that there is no complete dominance of one algorithm over the other when applied to the same problem. Looking at figure 15 and taking the smallest problem (medium) as an example, we notice that for smaller cutoff points, GSATRW achieves higher solution probabilities, while for larger cutoff points LA-GSATRW shows increasingly superior performance.

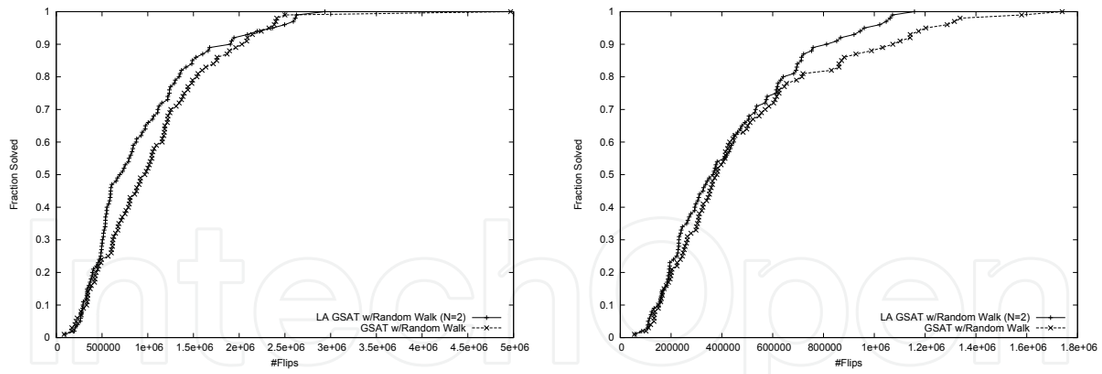


Fig. 13. (Left) Cumulative distributions for a 1141-variable logistics problem with 10719 clauses (logistics.c)). (Right) Cumulative distribution for a 4713-variable logistics problem with 21991 clauses (logistics.d). Along the horizontal axis we give the number of flips, and along the vertical axis the the success rate.

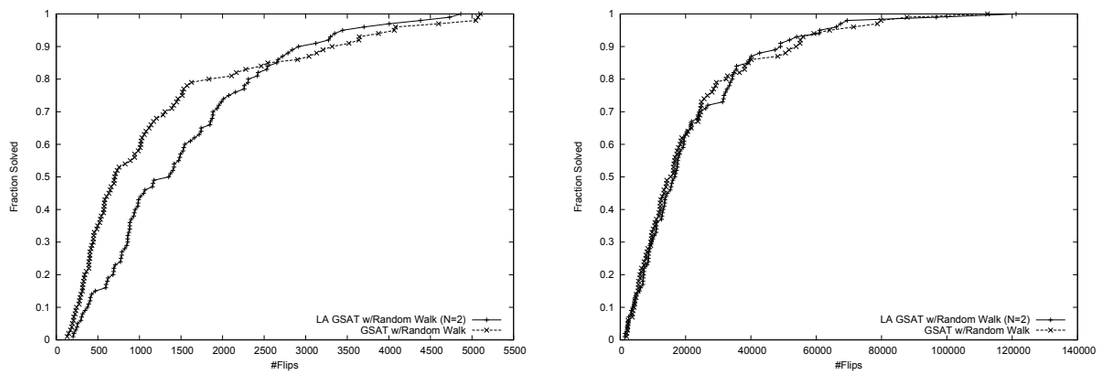


Fig. 14. (Left) Cumulative distribution for a 116-variable *Blocks World* problem with 953 clauses (medium). (Right) Cumulative distribution for a 459 -variable *Blocks World* problem with 4675 clauses (bw-large.a). Along the horizontal axis we give the number of flips, and along the vertical axis the fraction of problems solved for different values of N.

It may be noted that GSATRW performs better than LA-GSATRW for the smallest problem (up to 49% more steps than LA-GSATRW). However this gap is fairly small and is within 5% for medium sized problems (bw-large.a, huge). On the other hand, for the larger problem bw-large.b, the situation is reversed. GSATRW requires 16% more steps than LA-GSATRW.
An interesting observation that can be made from the above discussed plots is the ability of both algorithms to show an improved performance when applied to structured problems, such as SAT-encoded Blocks world and logistics problems. Taking for instance the large Block world problem bw-large (1087 variables, 13772 clauses), the median search cost of both methods is around 95% better compared to that measured for Random-3-SAT problem f1000 (1000 variables, 4250 clauses). Finally, the plots in Figures 17 and 18 explore the behaviour of the RLDs when for both algorithms when applied to BMC problems. Both algorithms reach a success rate of 100% with the one exception that,for the medium size problem (bmc-ibm3), the success rate was around 95%. Returning to Figure 17 then, for the smaller problem (bmc-

ibm-2), GSATRW dominates LA-GSATRW on the major part of the runs (i.e, approx 80%), as it reaches high solution probabilities while requiring lower search cost. On the other hand, for the medium sized problem (bmc-ibm-3), the situation is similar, but reversed.
Figure 18 shows the RLD for both algorithms for a larger problem (bmc-ibm-6). As can be seen from the figure, the RLDs for both algorithms have roughly the same shape. The presence of heavy tails in both RLDs indicates that both algorithms get stuck in local minima for a relatively small number of trials. The median search cost for GSATRW is 15% of that of LA-GSATRW for the bmc-ibm-2. However, LA-GSATRW shows a better performance for the medium (improvement of about 8% in the median) and larger problems (improvement of approximately 5%).
Table 1 shows the coefficient of variation (normalised standard deviations) for LA-GSATRW. As can be seen from the previous plots, there is a large variability between the search cost of the different runs. In particular, the long tails of the RLDs show that some of the runs requires much more effort than others. For increasing problem sizes, there is no clear indication that variability increases, as in the case of SAT-encoded BMC problems.

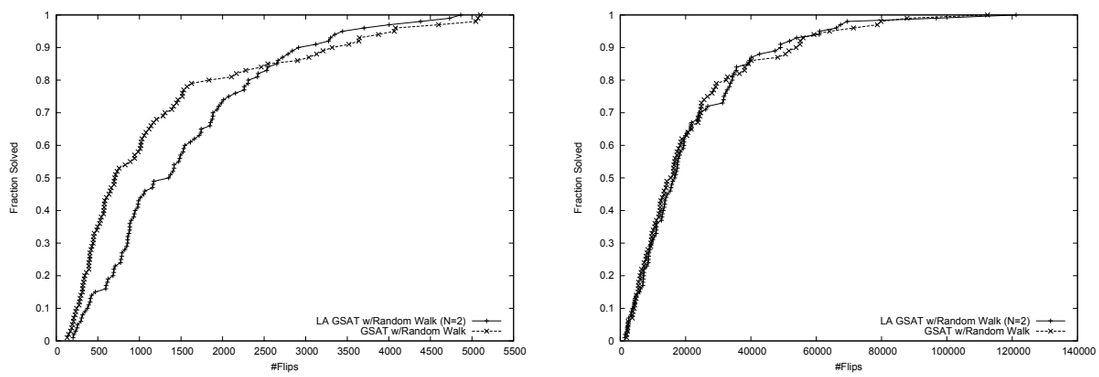


Fig. 15. (Left) Cumulative distribution for a 116-variable *Blocks World* problem with 953 clauses (medium). (Right) Cumulative distribution for a 459 -variable *Blocks World* problem with 4675 clauses (bw-large.a). Along the horizontal axis we give the number of flips, and along the vertical axis the fraction of problems solved for different values of N.

4.2.3 Excess deviation from the solution

Quite often, we observed stagnation behaviour with extremely low asymptotic solution probabilities when applied to SAT-encoded quasigroup problems. The two algorithms were executed to the allowed maximal number of steps and the percentage excess over the solution was recorded. Figures 19 and 20 show the excess deviation over the solution sorted in increasing order. As it can be seen from the plots, both algorithms suffers from severe stagnation indicating incomplete behaviour of the two algorithms when applied to this class of problems. At the exception of the problem qg3-08 where LA-GSATRW achieved a maximal success rate of 0.04% compared to 0.03% for GSATRW, we observe a rapid deterioration of their performance (success rate equals to 0%) with growing problem size. LA-GSATRW appears to have an asymptotic convergence which is better than GSATRW to around $3\% - 10\%$ in average excess of the solution.

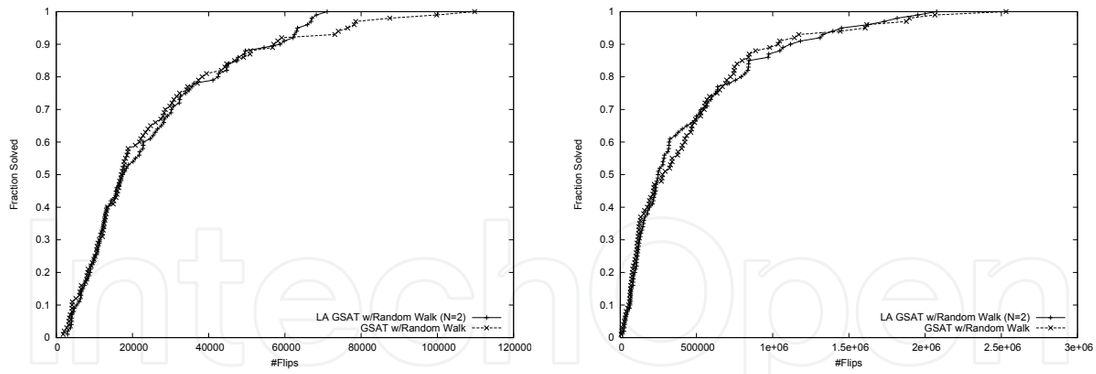


Fig. 16. (Left) Cumulative distributions for a 459-variable *Blocks World* problems with 7054 clauses (huge). (Right) Cumulative distribution for a 1087-variable *Blocks world* problem with 13772 (bw-large.b). Along the horizontal axis we give the number of flips, and along the vertical axis the success rate.

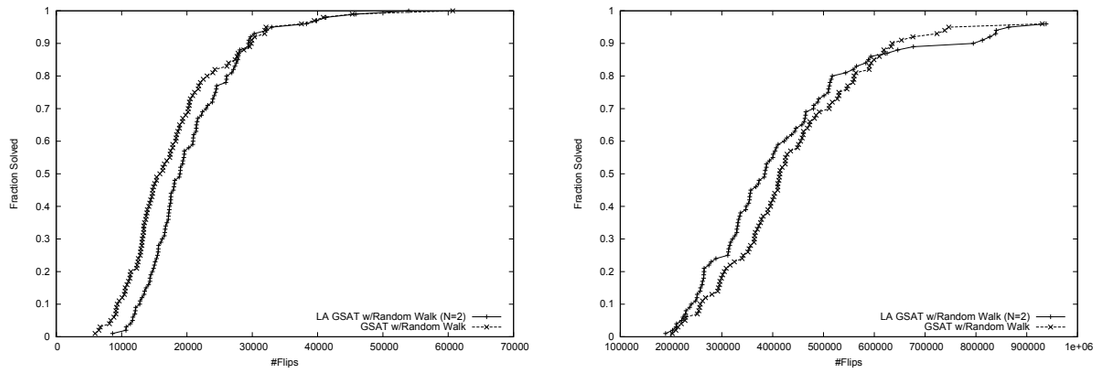


Fig. 17. (Left) Cumulative distributions for a 3628-variable *BMC* problem with 14468 clauses (bmc-ibm2). (Right) Cumulative distribution for a 14930-variable *BMC* problem with 72106 clauses (bmc-ibm3). Along the horizontal axis we give the number of flips, and along the vertical axis the success rate.

4.2.4 Wilcoxon Rank-Sum Test

The number of search flips needed by a meta heuristic to find a feasible solution may vary significantly from run to run on the same problem instance due to random initial solutions and subsequent randomized decisions. As RLDs are unlikely to be normally distributed, we turn to the non-parametric Wilcoxon Rank test in order to test the level of statistical confidence in differences between the median search cost of the two algorithms. The test requires that the absolute values of the differences between the mean search costs of the two algorithms are sorted from smallest to largest and these differences are ranked according to absolute magnitude. The sum of the ranks is then formed for the negative and positive differences separately. As the size of the trials increase, the rank sum statistic becomes normal. If the null hypothesis is true, the sum of ranks of the positive differences should be about the same

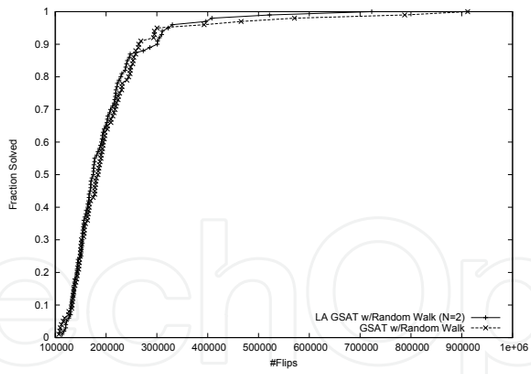


Fig. 18. Cumulative distributions for a 8710-variable *BMC* problems with 8710 clauses (bmc-ibm6). Along the horizontal axis we give the number of flips, and along the vertical axis the success rate.

Test-Problem	**LA-GSATRW**
f600	11.36
f1000	9.42
f2000	3.90
logistics.a	8.85
logistics.b	7.76
logistics.d	6.14
medium	6.84
bw-large.a	9.12
huge	7.62
bw-large.b	10.49
ibm-bmc2	3.71
ibm-bmc3	3.89
ibm-bmc6	4.30

Table 1. Coefficient of variation.

as the sum of the ranks of the negative differences. Using two-tailed P value, significance performance difference is granted if the Wilcoxon test is significant for $P < 0.05$
An initial inspection of Table 2 reveals two results. Firstly, the success rate of LA-GSATRW was better in 12 problems and the difference in the median search cost was significant in 6 problems. On the other hand, GSASTRW gave better results in 5 problems in terms of success rate but its performance was significant in only 2 cases.

5. Conclusions and Further Work

In this work, we have introduced a new approach based on combining Learning Automata with Random Walk and GSAT w/Random Walk. In order to get a comprehensive overview of the new algorithms' performance, we used a set of benchmark problems containing different problems from various domains. In these benchmark problems, both RW and GSATRW suffers from stagnation behaviour which directly affects their performance. This phenomenon is, however, only observed for LA-GSATRW on the largest problem instances. Finally, the

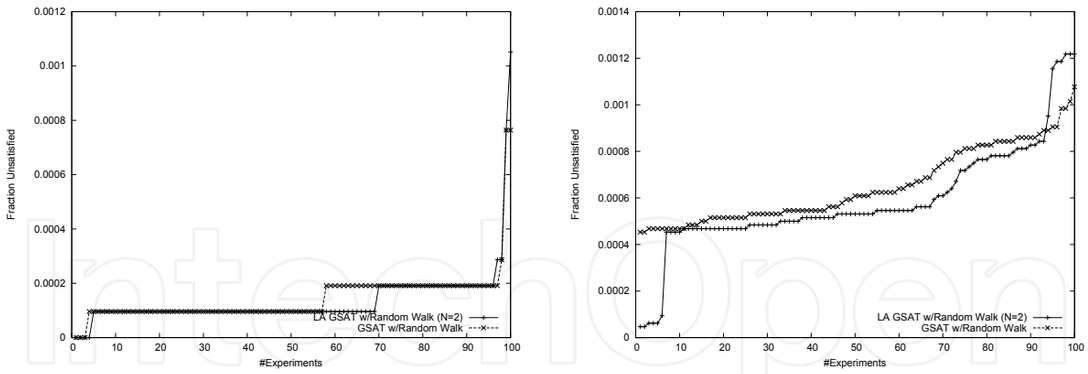


Fig. 19. Excess deviation over the solution for LA-GSATRW and GSATRW. (Left) qg3-08 (512 variables, 10469 clauses). qg5-11 (1331 variables, 64054 clauses). Along the horizontal axis we give the number of trials and along the vertical axis the percentage deviation over the solution.

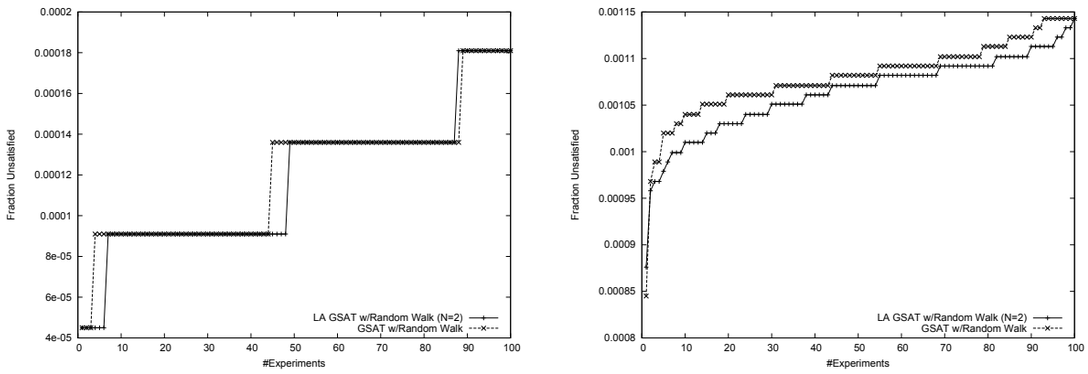


Fig. 20. Excess deviation over the solution for LA-GSATRW and GSATRW. (Left) qg7-09 (729 variables, 22060 clauses). (Right) qg7-13 (2197 variables, 97072 clauses). Along the horizontal axis we give the number of trials and along the vertical axis the percentage deviation over the solution.

success rate of LA-GSATRW was better in 12 of the problems, 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 5 of the problems in terms of success rate, while its performance was significantly better only in 2 problems.

Based on the empirical results, it can be seen that the Learning Automata mechanism employed in LARW and 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, such as WalkSAT. 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.

Problem	SR: LA-GSATRW	SR: GSATRW	P value	NULL Hypotheis
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
qg-03-08	20%	33%	0.16	Accept
qg-5-11	59%	38%	0.00	Reject
qg-7-9	33%	59%	0.61	Accept
qg-7-13	59%	33%	0.00	Reject

Table 2. Success rate (SR) and Wilcoxon statistical test.

6. References

A.E.Eiben & van der Hauw, J. (1997). Solving 3-sat with adaptive genetic algorithms, *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, IEEE Press, pp. 81–86.

Cha, B. & Iwama, K. (1995). Performance Tests of Local Search Algorithms Using New Types of Random CNF Formula, *Proceedings of IJCAI'95*, Morgan Kaufmann Publishers, pp. 304–309.

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

Davis, M. & Putnam, H. (1960). A computing procedure for quantification theory, *Journal of the ACM* **7**: 201–215.

Frank, J. (1997). Learning short-term clause weights for gsat, *Proceedings of IJCAI'97*, Morgan Kaufmann Publishers, pp. 384–389.

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

Gent, L. & T.Walsh (1993). Towards an Understanding of Hill-Climbing Procedures for SAT, *Proceedings of AAAI'93*, MIT Press, pp. 28–33.

Glover, F. (1989). Tabu search-part 1, *ORSA Journal on Computing* **1**: 190–206.

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., Oommen, B. J., Myrer, S. A. & 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.

Hansen, P. & Jaumand, B. (1990). Algorithms for the maximum satisfiability problem, *Computing* **44**: 279–303.

I.Gent & Walsh, T. (1995). Unsatisfied variables in local search, *Hybrid Problems,Hybrid Solutions*, IOS Press, pp. 73–85.

Johnson, D. & Trick, M. (1996). *Cliques, Coloring, and Satisfiability*, Volume 26 of DIMACS Series on Discrete Mathematics and Theoritical Computer Science, American Mathematical Society.

McAllester, D., Selman, B. & Kautz, H. (1997). Evidence for Invariants in Local Search, *Proceedings of AAAI'97*, MIT Press, pp. 321–326.

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

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

Oommen, B. J. & 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. & 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. & 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. & Cohen, B. (1994). Noise Strategies for Improving Local Search, *Proceedings of AAAI'94*, MIT Press, pp. 337–343.

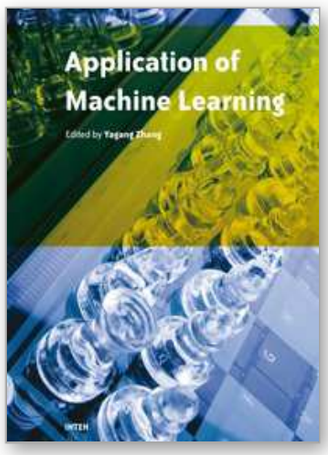
Selman, B., Levesque, H. & Mitchell, D. (1992). A new method for solving hard satisfiability problems, *Proceedings of AAA'92*, MIT Press, pp. 440–446.

Spears, W. (1993). Simulated Annealing for Hard Satisfiability Problems. Technical Report, Naval Research Laboratory, Washington D.C.

Thathachar, M. A. L. & 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.

IntechOpen



Application of Machine Learning
Edited by Yagang Zhang

ISBN 978-953-307-035-3
Hard cover, 280 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

The goal of this book is to present the latest applications of machine learning, which mainly include: speech recognition, traffic and fault classification, surface quality prediction in laser machining, network security and bioinformatics, enterprise credit risk evaluation, and so on. This book will be of interest to industrial engineers and scientists as well as academics who wish to pursue machine learning. The book is intended for both graduate and postgraduate students in fields such as computer science, cybernetics, system sciences, engineering, statistics, and social sciences, and as a reference for software professionals and practitioners. The wide scope of the book provides them with a good introduction to many application researches of machine learning, and it is also the source of useful bibliographical information.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen