GSAT Enhanced with Learning Automata and Multilevel Paradigm

Noureddine Bouhmala[1] and Ole-Christoffer Granmo[2]

[1] Department of Maritime Technology and Innovation, Vestfold University College
Borre, NO-3184, Norway

[2] Department of ICT, University of Agder
Grimstad, NO-4876, Norway

Abstract

A large number of problems that occur in knowledge-representation, learning, very large scale integration technology (VLSI-design), and other areas of artificial intelligence, are essentially satisfiability problems. The satisfiability problem refers to the task of finding a satisfying assignment that makes a Boolean expression evaluate to *True*. The growing need for more efficient and scalable algorithms has led to the development of a large number of SAT solvers. This paper introduces two new techniques that combine finite learning automata and multilevel paradigm with the Greedy Satisfiability Algorithm (GSAT). We present a detailed comparative analysis of the new approaches using a benchmark set containing randomized and practical engineering applications from various domains.

Keywords: *Satisfiability problem, learning automata, multilevel techniques, combinatorial optimization.*

1. Introduction

The satisfiability problem (SAT) which is known to be NP-complete (nondeterministic polynomial time) [2] plays a central role in many applications in the fields of VLSI (Very-large-scale integration) Computer-Aided design, Computing Theory, and Artificial Intelligence. Generally, a SAT problem is defined as follows. A propositional formula $\Phi = \wedge_{j=1}^{m} C_j$ 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, 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 $I_j, \overline{I}_j \subseteq \{1,.....n\}$, $I \cap \overline{I}_j = \varnothing$, and $\overline{x}_i$ 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. 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. Since most known combinatorial optimization problems can be reduced to SAT [7], the design of special methods for SAT can lead to general approaches for solving combinatorial optimization problems.

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, and a literal being a Boolean variable or its negation. For example, $P \vee Q$ is a clause containing the two literals P and Q. The clause $P \vee Q$ is satisfied if either P is *True* or Q is *True*. When each clause in Φ contains exactly k literals, the resulting SAT problem is called k-SAT.

In essence, we here enhance the traditional GSAT Random Walk (GSATRW) strategy with learning capability, taking the form of Learning Automata and the multi-level paradigm. Learning Automata have been used to model biological systems [42] and have attracted considerable interest in the last decade 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. The multilevel paradigm is a simple technique which at its core involves recursive coarsening to produce smaller and smaller problems that are easier to solve than the original one.

The paper is organized as follows. In Section 2 we survey algorithms used for solving the SAT problem. In Section 3 we take a closer look at the GSAT Random Walk algorithm, before we in Section 4 explain how the latter algorithm can be enhanced with learning capability, using the basic concepts of Learning Automata. In Section 5, we report the results obtained from testing the new approach. Section 6 introduces the multilevel generic paradigm. In section 7 we describe the main components of the multilevel paradigm when applied to the satisfiability problem. Section 8 presents the results. In section 9 we apply the Wilcoxon Rank Sum test to test the significance of the results. Finally, in Section 10, we summarize the findings and suggest directions for future research.

2. Stochastic Local Search Algorithms (SLS)

The SAT problem 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 local search algorithms for solving SAT problems.

Local search algorithms typically start with an initial assignment of truth values to variables, randomly or heuristically generated. Satisfiability can then be formulated as an iterative optimization problem in which the goal is to minimize the number of unsatisfied clauses. 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.

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.

One of the earliest local search algorithms for solving SAT is GSAT [37]. Basically, GSAT begins with a randomly generated assignment of values to variables, and then uses the steepest descent heuristic to find the new variable-value assignment which best decreases the numbers of unsatisfied clauses. After a fixed number of 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 [39] 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 [38]. 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 equal to zero, this variable is flipped, otherwise, the variable with minimal break count is selected with a certain probability (noise probability). It turns out that the choice of unsatisfied clauses, combined with the randomness in the selection of variables, enable WalkSAT to avoid local minima and to better explore the search space.

Extensive tests have led to the introduction of new variants of the Walksat algorithm referred to as Novelty and R-Novelty [26]. These two variants use a combination of two criteria when choosing a variable to flip from within an unsatisfied clause. Quite often, these two algorithms can get stuck in local minima and fail to get out. To this end, recent variants have been designed [24][25][16] using a combination of search intensification and diversification mechanisms, leading to good performance on a wide range of SAT instances.

Other algorithms [10][13][8][9] use history-based variable selection strategies in order to avoid repeated flipping of the same variable. In parallel to the development of more sophisticated versions of randomized improvement techniques, other methods based on the idea of modifying the evaluation function [44][17][41][35][36] in order to prevent the search from getting stuck in non-attractive areas of the underlying search space have become increasingly popular in SAT solving. The key idea is to associate the clauses of the given CNF formula with

weights. Although these clause weighting SLS algorithms differ in the way 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.

A new approach to clause weighting known as Divide and Distribute Fixed Weights (DDFW) [18] exploits the transfer of weights from neighboring satisfied clauses to unsatisfied clauses in order to break out form local minima. Recently, a strategy based on assigning weights to variables [34] instead of clauses greatly enhances the performance of the Walksat algorithm, leading to the best known results on some benchmarks.

Lacking theoretical guidelines while being stochastic in nature, the deployment of several meta-heuristics involves extensive experiments to find the optimal noise or walk probability settings. To avoid manual parameter tuning, new methods have been designed to automatically adapt parameter settings during the search [23][33], and results have shown their effectiveness for a wide range of problems.

The work conducted in [12] introduced Learning Automata (LA) as a mechanism for enhancing random walk algorithm, thus laying the foundation for novel LA-based SAT solvers. Finally, a new approach based on an automatic procedure for integrating selected components from various existing solvers in order to build new efficient algorithms that draw upon the strengths of multiple algorithms was proposed in [45][22].

3. The GSAT-Random-Walk Algorithm (GSATRW)

This section is devoted to explaining the details of the GSATRW algorithm as it is embedded into our Finite Learning Automata based strategy. The main motivation behind choosing the GSATRW algorithm is the fact that all state-of-the-art SAT solving algorithms are derivatives of GSATRW. They all have the general GSATRW backbone architecture with some additional features such as random restart, clause weighting and Tabu list mechanisms, and therefore comparing the performance of GSATRW with and without enhancements are informative.

As argued previously, the introduction of an element of randomness (i.e., noise) into local search methods is common practice for improving effectiveness through diversification [1]. In this spirit, the GSATRW algorithm (shown in Fig. 1) starts with a randomly chosen assignment. Thereafter, two possible strategies are used for selecting the variable to be flipped at each iteration of the algorithm. The first strategy is taking a *walk-step*, which amounts to randomly selecting a currently unsatisfied clause and then flipping one of its variables, also in a random manner. Thus, at each walk-step, at least one unsatisfied clause becomes satisfied. The other strategy uses a greedy search to choose a random variable from the set *PossFlips*, which contains the variables that when flipped (individually) achieve the largest decrease (or the least increase) in the total number of unsatisfied clauses. Note that the walk-step strategy may lead to an increase in the total number of unsatisfied clauses even when purely improving flips would have been possible.

```
Procedure learning_automata_random_walk()
Input :A set of clauses, MAX-TRIES, MAX-FLIPS, Walking probability p;
Output : A satisfying truth assignment of the clauses, if found;
Begin
    /* Initialization */
    For i := 1 To MAX_TRIES Do
        T ← a random truth assignment; ;
        For j := 1 To MAX_FLIPS Do
            If T satisfies the clauses Then return T;
            If rnd(0, 1) ≤ p Then
                PossFlips ← a randomly selected variable with the
                largest decrease in unsatisfied clauses;
            Else
                PossFlips ← a random variable occurring in some unsatisfied clause;
            V ← Pick (PossFlips);
            T ← T with V flipped;
        EndFor ;
    EndFor
    return " no solution found "
End
```

Fig. 1 GSAT-Random-Walk Algorithm.

4. Solving SAT Using Finite Learning Automata

We base our work on the principles of Learning Automata [28][40]. Learning Automata have been used to model biological systems [42], 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 [30][6][11][27][32][31][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 GSATRW can be enhanced with learning capability, using Learning Automata.

4.1 A Learning SAT Automaton

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. Fig. 2 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 (LA).

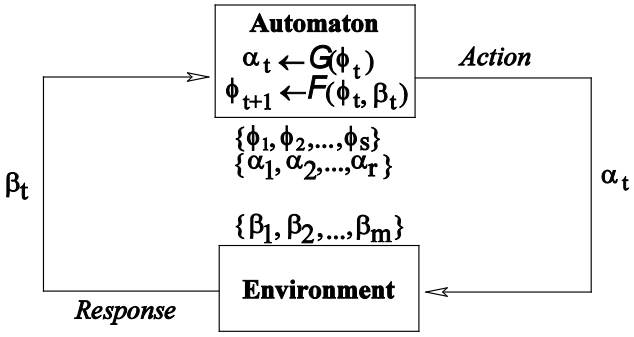

Fig. 2 A learning automaton interacting with an environment.

The finite learning automaton can be defined in terms of a quintuple [28]:

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

$\underline{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_s\}$ 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 $\alpha_t = G[\phi_t]$ determines the next action performed by the automaton given the current automaton state. Finally, a transition function $\phi_{t+1} = F[\phi_t, \beta_t]$ 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 [28][40] 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 [42][28]. 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: $\underline{\alpha} = \{True, False\}$.
- Inputs: $\underline{\beta} = \{reward, penalty\}$.

Fig. 3 specifies the G and F matrices.

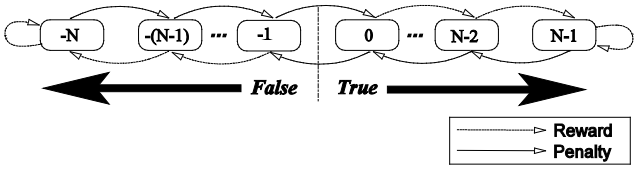

Fig. 3 The state transitions and actions of the Learning SAT automaton.

The 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 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.

4.2 Combining Learning Automata with GSATRW(LA-GSATRW)

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 SAT problem instance as defined in Section 1. Each variable of the 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 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-GSATRW that we propose. Fig. 4 contains the complete pseudo-code for solving SAT problem instances, 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. 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 automata based learning, this scheme is incremental, and learning is performed gradually, in small steps.

```
Procedure learning_automata_gsat_random_walk()
Input : A set of clauses 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(C) Do
        If rnd(0, 1) ≤ p Then
            /* Draw unsatisfied clause randomly */
            C_j = random_unsatisfied_clause(C);
            /* Draw clause literal randomly */
            i = random_element(I_j ∪ Ī_j);
        Else
            /* Randomly select one of the literals whose flipping */
            /* minimizes the number of unsatisfied clauses */
            i = random_element(Best_Literal_Candidates(C));

        /* 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);

        If rnd(0, 1) ≤ p Then
            /* Draw satisfied clause randomly */
            C_j = random_satisfied_clause(C);
            /* Draw clause literal randomly */
            i = random_element(I_j ∪ Ī_j);
        Else
            /* Randomly select one of the literals whose flipping maximizes the */
            /*number of unsatisfied clauses */
            i = random_element(Worst_Literal_Candidates(C))
            /*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. 4 Learning Automata GSAT Random Walk Algorithm.

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 [28]. That is, multiple LAs 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
[28]. In our case, the LA involved in the LA-GSATRW is 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 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 SAT problem instance at hand becomes increasingly focused.

5. Empirical Results: Learning Automata

5.1 Benchmark Instances

As a basis for the empirical evaluation of LA-GSATRW, we selected benchmark instances which are available from the SATLIB website (www.satlib.org). All the benchmark instances used in this experiment are satisfiable instances and have been used widely in the literature in order to give an overall picture of the performance of different algorithms. Due to the randomization of the algorithm, the number of flips required for solving a problem instance varies widely between different runs. Therefore, for each problem instance, we run LA-GSATRW and GSATRW 100 times with a cutoff parameter (maxflips) setting which is high enough (10^7) to guarantee a success rate close to 100% .

5.2 Run-Length-Distributions (RLDs)

As an indicator of the behavior of the two algorithms when trying to solve a given problem instance in 100 trials, and 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. Due to non-deterministic decisions involved in the algorithm (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 $\hat{P}(f'(j) \le f) = |\{j \mid f'(j) \le f\}/k$. Each problem was solved 100 times using an extremely high cutoff parameter setting of $Maxsteps = 10^7$ in order to obtain a maximal number of successful tries.

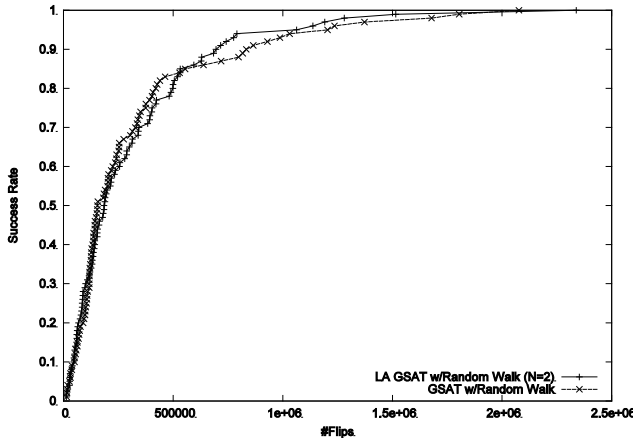

Fig. 5 LA-GSATRW Vs GSATRW: Cumulative distributions for a 600-variable *random* problem with 2550 clauses (f600).

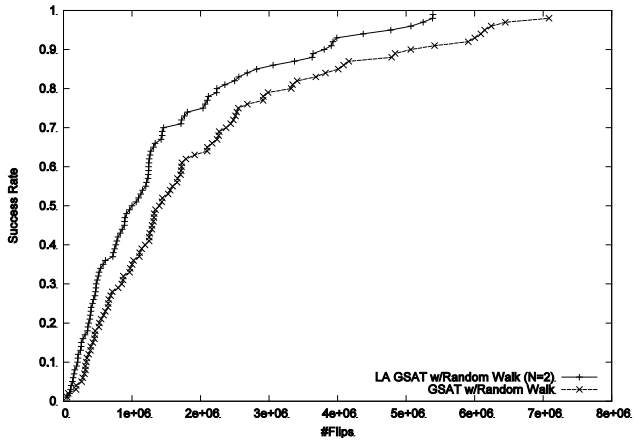

Fig. 6 LA-GSATRW Vs GSATRW: Cumulative distribution for a 1000-variable *random* problem with 4250 clauses (f1000).

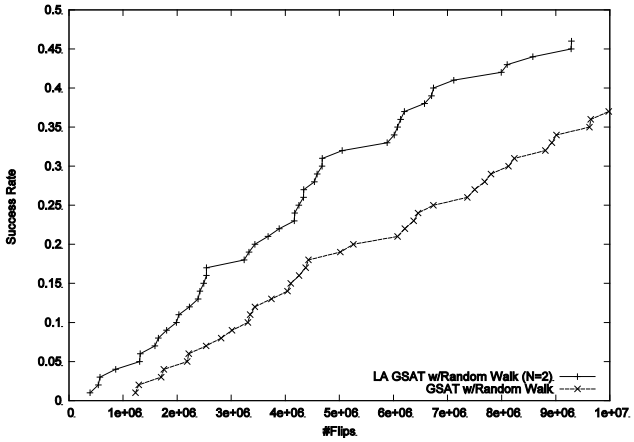

Fig. 7 LA-GSATRW Vs GSATRW: Cumulative distributions for a 2000-variables random problem with 8500 clauses (f2000).

Figs. Fig. 5-7 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. 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.

As can be seen from these plots, LA-GSATRW gives consistently higher success probabilities while requiring fewer search steps compared to GSATRW.

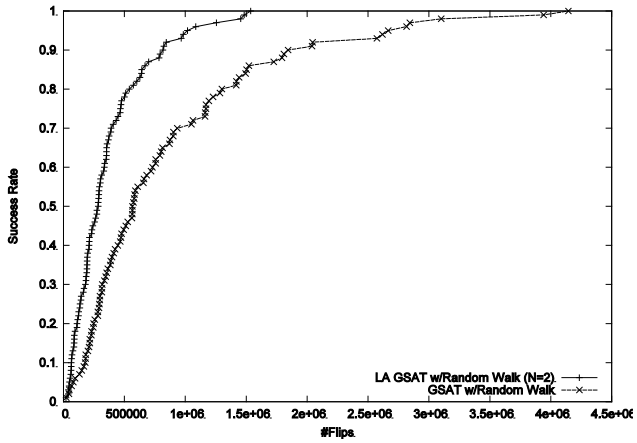

Fig. 8 LA-GSATRW Vs GSATRW: Cumulative distributions for a 228-variable *logistics* problem with 6718 clauses (logistics-a).

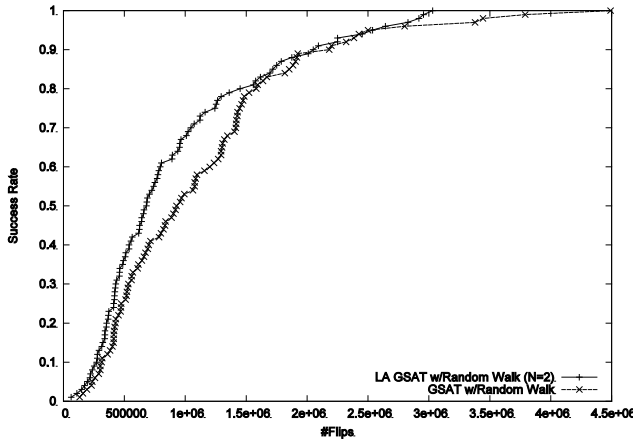

Fig. 9 LA-GSATRW Vs GSATRW: Cumulative distribution for a 843-variable *logistics* problem with 7301 clauses (logistics-b).

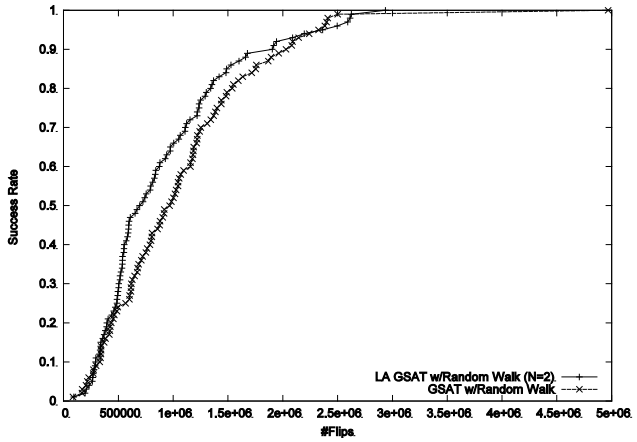

Fig. 10 LA-GSATRW Vs GSATRW: Cumulative distributions for an 1141-variable *logistics* problem with 10719 clauses (logistics-c).

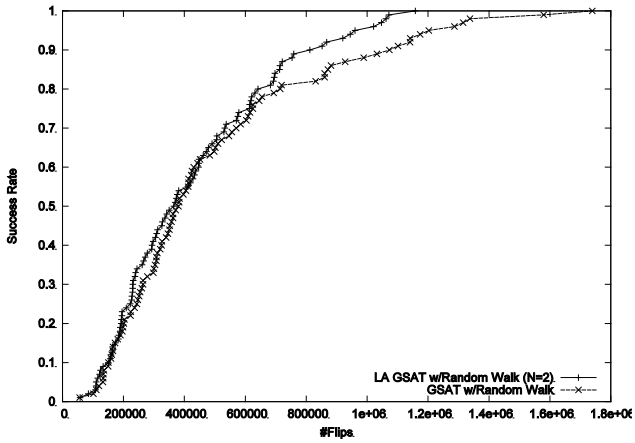

Fig. 11 LA-GSATRW Vs GSATRW: Cumulative distribution for a 4713-variable *logistics* problem with 21991 clauses (logistics-d).

Looking at Figs. 8-11 and applying the RLD analysis to SAT-encoded logistics problems, there is no clear success rate winner between the two algorithms. The number of search steps varies between the different trials and is significantly higher with GSATRW than that of LA-GSATRW. 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.

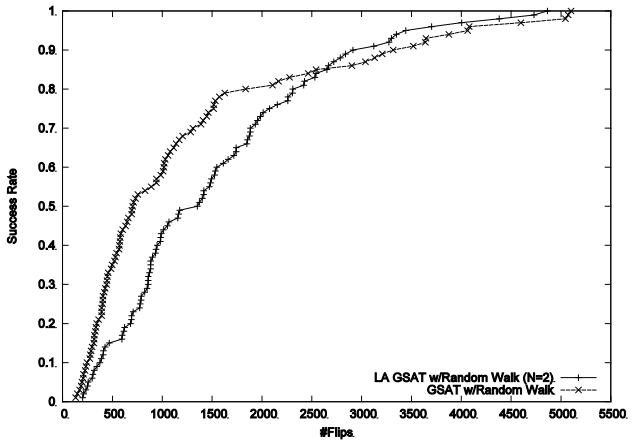

Fig. 12 LA-GSATRW Vs GSATRW: Cumulative distribution for a 116-variable *Blocks World* problem with 953 clauses (medium).

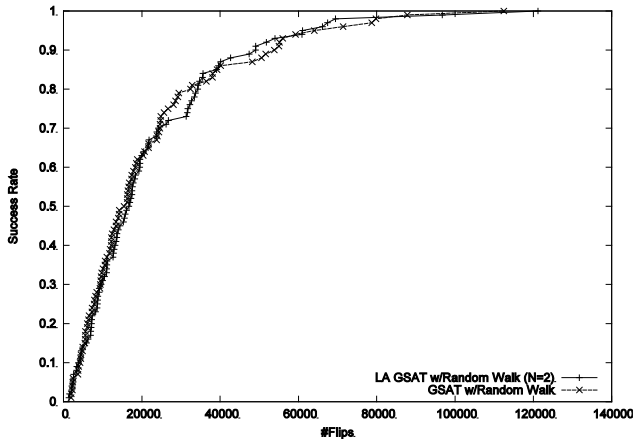

Fig. 13 LA-GSATRW Vs GSATRW: Cumulative distribution for a 459-variable *Blocks World* problem with 4675 clauses (bw-large.a).

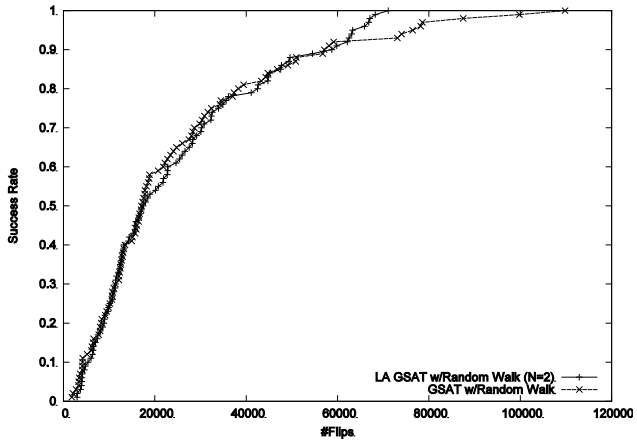

Fig. 14 LA-GSATRW Vs GSATRW: Cumulative distributions for a 459-variable *Blocks World* problem with 7054 clauses (huge).

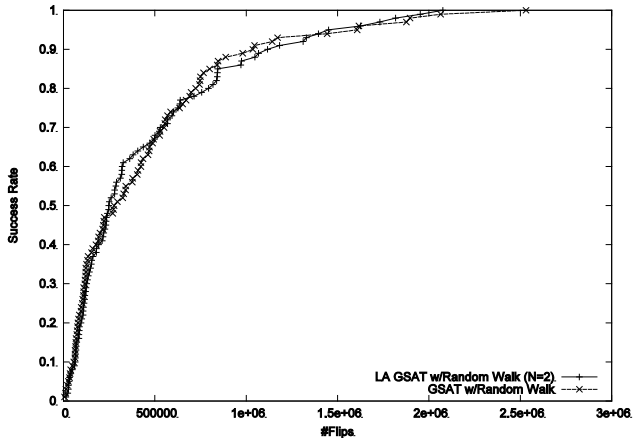

Fig. 15 LA-GSATRW Vs GSATRW: Cumulative distribution for a 1087-variable *Blocks World* problem with 13772 clauses (bw-large.b).

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 Figs. 12-15, indicates that there is no complete dominance of one algorithm over the other when applied to the same problem. Looking at Figs. 10-11 and taking the smallest problem (medium) as an example, we notice that for smaller cutoff times, GSATRW achieves higher solution probabilities, while for larger cutoff times, LA-GSATRW shows increasingly superior performance. 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 size problems (bw-large.a, bw-huge). On the other hand, for the large problem bw-large.b, the situation is reversed. GSATRW requires 16% more steps than LA-GSATRW.

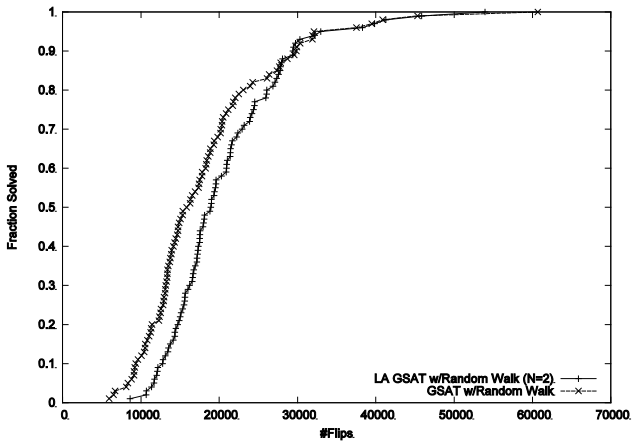

Fig. 16 LA-GSATRW Vs GSATRW: Cumulative distributions for a 3628-variable *BMC* problem with 14468 clauses (bmc-ibm2).

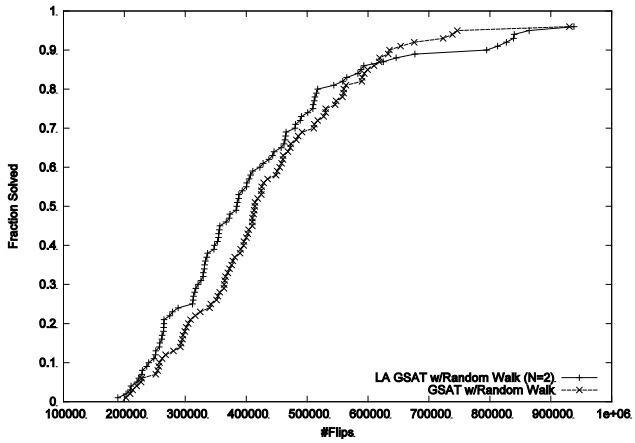

Fig. 17 LA-GSATRW Vs GSATRW: Cumulative distribution for a 14930-variable *BMC* problem with 72106 clauses (bmc-ibm3).

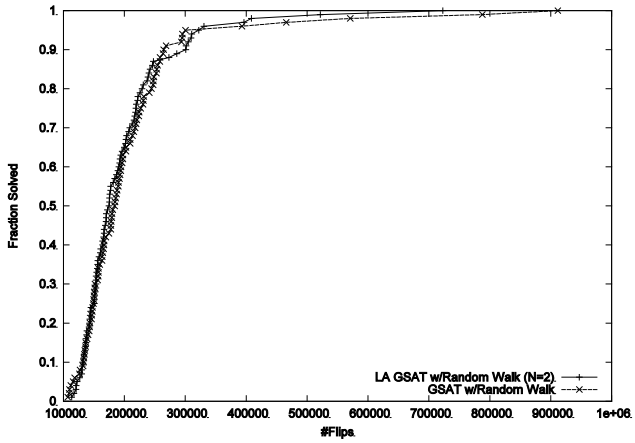

Fig. 18 LA-GSATRW Vs GSATRW: Cumulative distributions for a 8710-variable *BMC* problem with 8710 clauses (bmc-ibm6).

Finally, the plots in Figs. 16-18 explore the behavior of the RLDs of 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 Fig. 16 then, for the smaller problem (bmc-ibm-2) GSATRW dominates LA-GSATRW on the major part of the runs (i.e., approximately 80%), as it reaches high solution probabilities while requiring lower search cost. On the other hand, for the medium size problem (bmc-ibm-3) the situation is similar but reversed.

Fig. 18 shows the RLD for both algorithms for the large 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 approximately 8% in the median) and larger problem (improvement of approximately 5%).

6. The Multilevel Paradigm

The multilevel paradigm is a simple technique which at its core involves recursive coarsening to produce smaller and smaller problems that are easier to solve than the original one. Fig. 19 shows the process of the generic multilevel paradigm in pseudo-code. The multilevel paradigm consists of three phases: coarsening, initial solution, and multilevel refinement. During the coarsening phase, a series of smaller problems is constructed by matching pair of vertices of the input original problem in order to form clusters, use the clusters to define a related coarser problem, and recursively iterate the coarsening procedure until a sufficiently small problem is obtained.

Computation of an initial solution is performed on the coarsest level (smallest problem). Finally, the solution found at each level is extended to give an initial solution for the next level and then refined using a chosen local search algorithm. A common feature that characterizes multilevel algorithms is that any solution in any of the coarsened problems is a legitimate solution to the original graph. This is always true as long as the coarsening is achieved in a way that each of the coarsened problems retains the original problem's global structure. The key success behind the efficiency of the multilevel techniques is the use of the multilevel paradigm. This paradigm offers two main advantages which enable local search techniques to become much more powerful in the multilevel context. First, by allowing local search schemes to view a cluster of vertices as a single entity, the search becomes restricted to only those configurations in the solution space in which the vertices grouped within a cluster are assigned the same label. During the refinement phase a local refinement scheme applies a local a transformation within the neighborhood (i.e., the set of solutions that can be reached from the current one) of the current solution to generate a new one. As the size of the clusters varies from one level to another, the size of the neighborhood becomes adaptive and allows the possibility of exploring different regions in the search space. Second, the ability of a refinement algorithm to manipulate clusters of vertices provides the search with an efficient mechanism to escape from local minima. Multilevel techniques were first introduced when dealing with the graph partitioning problem (GCP) [14] [20] [21] [43] and have proved to be effective in producing

high quality solutions at a lower cost than single level techniques.

```
Procedure Multilevel Paradigm:

Input: Problem P_0

Output: Solution S_final(P_0)

Begin
    level := 0
    /* Coarsening Phase */
    While (Not reached the desired number of levels )
          P_level+1 := Coarsen (P_level)
          level := level + 1
    end
    /* Initial Solution is computed at the lowest level */
    S(P_level) =Initial Solution (P_level)
    /* Uncoarsening and Refinement phase */
    While (level > 0)
          S_start(P_level-1): = Uncoarsen (S_final(P_level))
          S_final(P_level-1) := Refine (S_start(P_level-1))
          level := level - 1
    end
End
```

Fig. 19 Multilevel Generic Algorithm.

7. The multilevel framework for the satisfiability problem

```
Procedure GSAT-Refinement()

Input :P_i,A_i, MAX-FLIPS;

Output : A satisfying truth assignment of the clauses, if found;

Begin
    For j := 1 To MAX_FLIPS Do
        If A_i satisfies the clauses Then return A_i;
        PossFlips ← a randomly selected multivariable with the
        largest decrease in unsatisfied clauses;
        V ← Pick (PossFlips);
        A_i ← A_i with V flipped;
    EndFor ;
End
```

Fig. 20 GSAT-Refinement Algorithm.

Coarsening: The original problem P_0 is reduced into a sequence of smaller problems $P_0, P_2, ...P_m$ such that $|P_0| > |P_1| > |P_2| ..... > |P_m|$. It will require at least $O(logN / N')$ coarsening phases to coarsen the problem down to N' variables. Let V_i^v denotes the set of variables of P_i combined to form variable v of P_{i+1}. We will refer to v as multivariable. During the coarsening phase, a sequence of smaller problems, each with fewer variables is constructed. Let P_0 denotes the original problem. The next level coarser problem P_1 is constructed from P_0 by collapsing pairs of variables into multi-variables. The variables are visited in random order. If a variable has not been matched yet, then we randomly select one randomly unmatched variable, and a multivariable consisting of these two variables is created. Unmatched variables are simply copied the next level. The new formed multi-variables are used to define a new and smaller problem and recursively iterate the coarsening process until the size of the problem reaches some desired threshold.

- **Initial solution**: An initial assignment A_m of P_m is easily computed using a random assignment algorithm. The random assignment algorithms works by randomly assigning to each multivariable of the coarsest problem P_m the value of true or false.
- **Projection**: Having optimized the assignment A_{k+1} on a P_{k+1}, the assignment must be projected onto its parent P_k. Since each multivariable of P_{k+1} contains a distinct subset of multi-variables of P_k, obtaining A_k from A_{k+1} is done by simply assigning the set of variables V_i^v collapsed to $v \in P_{k+1}$ the same value as v (i.e., $A_i[u] = A_{i+1}[v]$, $\forall u \in V_i^v$).
- **Refinement**: At each level, the assignment from the previous level is projected back to give an initial assignment and further refined. Even though A_{i+1} is a local minimum of P_{i+1}, the projected assignment A_i may not be at a local optimum with respect to P_i. Since P_i is finer, it may still be possible to improve the projected assignment using a version of GSAT adapted to the multilevel paradigm. The idea of GSAT refinement as shown in Fig. 20 is to use the projected assignment of P_{i+1} onto P_i as the initial assignment of GSAT. Since the projected assignment is already a good one, GSAT will converge quicker to a better assignment. During each level, GSAT is allowed to perform MAXFLIPS iterations before moving to the finer level. If a solution is not found at the finest level, a new round of coarsening, random initial assignment, and refinement is performed.

8. Empirical Results: Multilevel Paradigm

Figs. 21-39 show results which appear to follow the same pattern. Overall, at least for the instances tested here, we observe that the search pattern happens in two phases. The first phase which corresponds to the early part of the search, both algorithms behave as a hill-climbing method. These phases which can be described as a short one, a large number of the clauses are satisfied. The best

assignment climbs rapidly at first, and then flattens off as we mount the 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. 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 plateau is rather of short length with MLVGSAT compared to that of GSAT. The projected solution from one level to a finer one offers an elegant mechanism to reduce the length of the plateau as it consists of more degree of freedom that can be used for further improving the best solution. The plots show a time overhead for MLVGSAT especially for large problems due mainly to data structures settings at each level. We feel that this initial overhead which is a common feature in multilevel implementations is more susceptible to further improvements, and will be considerably minimized by a more efficient implementation. Comparing GSAT and MLVGSAT for small sized problems (up to 1500 clauses) and as can be seen from the left sides of Figs. 24, 25, 28 and 29, both algorithms seem to be reaching the optimal quality solutions. It is not immediately clear which of the two algorithms converges more rapidly. This is probably very dependent on the choice of the instances in the test suite. For example the run time required by MLVGSAT for solving instance flat100-239 is more than 12 times higher than the mean run-time of GSAT (25sec vs 2sec). The situation is reversed when solving the instance block-medium (20sec vs 70sec). The difference in convergence behavior of both algorithms starts to become more distinctive as the size of the problem increases. All the plots show a clear dominance of MLGSAT over GSAT throughout the whole run. MLVGSAT shows a better asymptotic convergence (to around $0.008\% - 0.1\%$) in excess of the optimal solution as compared with GSAT which only reach around (0.01%-11%). The performance of MLVGSAT surpasses that of GSAT although few of the curves overlay each other closely, MLVGSAT has marginally better asymptotic convergence.

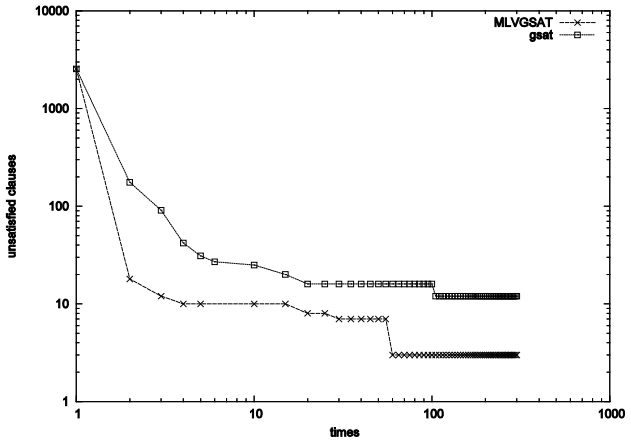

Fig. 21 Log-Log plot, Random: Evolution of the best solution on a 600 variable problem with 2550 clauses (f600.cnf).

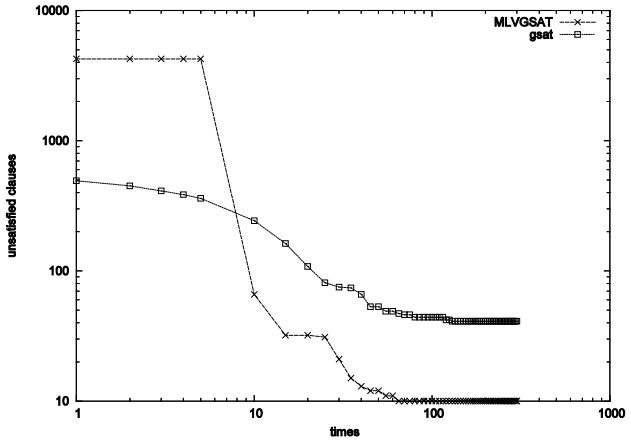

Fig. 22 Log-Log plot, Random: Evolution of the best solution on a 1000 variable problem with 4250 clauses. (f1000.cnf).

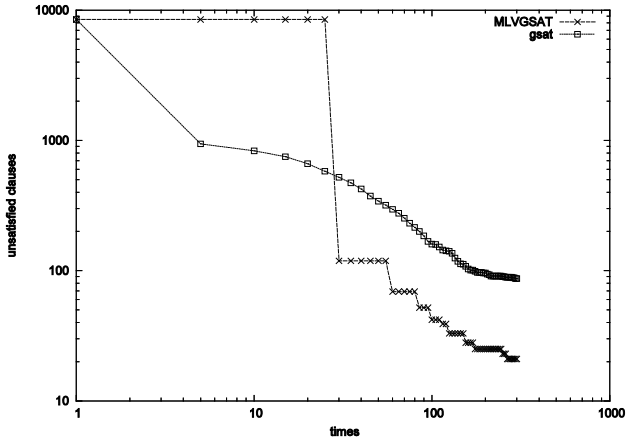

Fig. 23 Log-Log plot, Random: Evolution of the best solution on a 2000 variable problem with 8500 clauses (f2000.cnf).

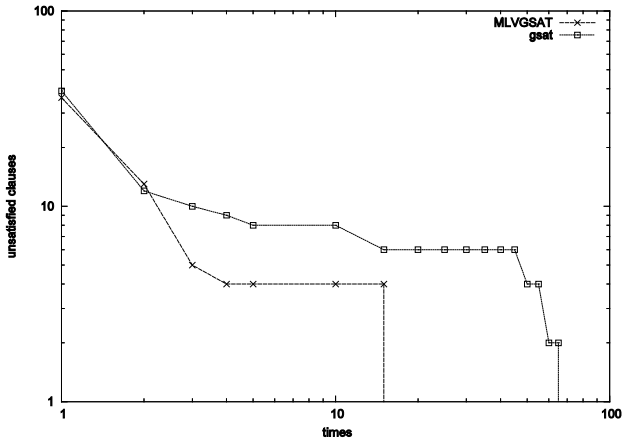

Fig. 24 Log-Log plot, SAT-encoded graph coloring: Evolution of the best solution on a 300 variable problem with 1117 clauses (flat100.cnf).

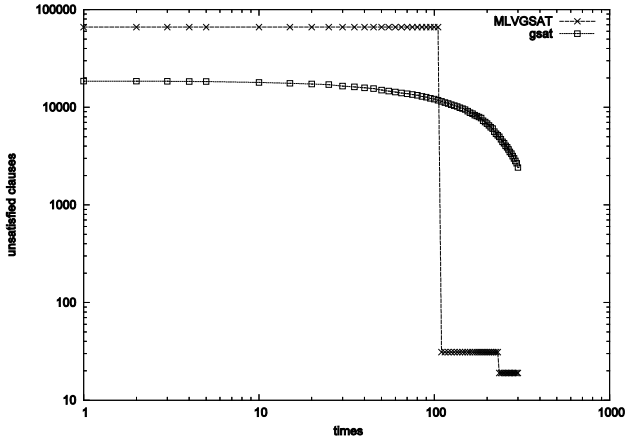

Fig. 25 Log-Log plot, SAT-encoded graph coloring: Evolution of the best solution on a 2125 problem with 66272 clauses (g125-17.cnf).

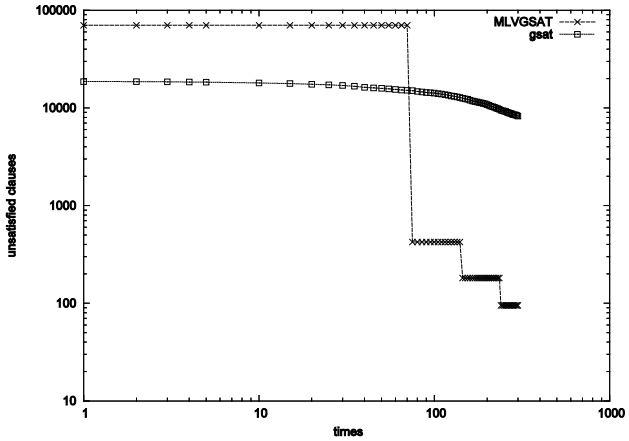

Fig. 26 Log-Log plot, SAT-encoded graph coloring: Evolution of the best solution on a 2250 variable problem with 70163 clauses (g125-18.cnf).

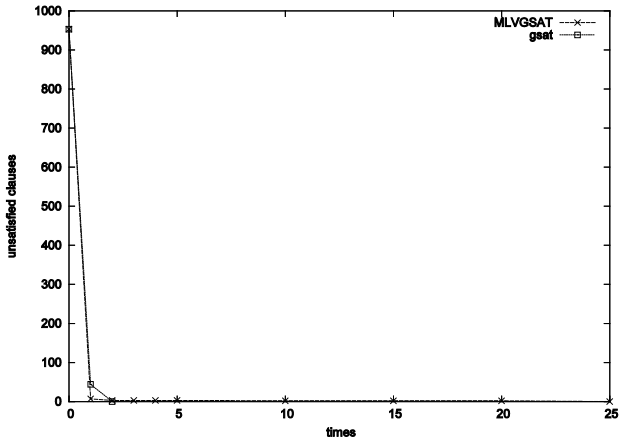

Fig. 27 Log-Log plot, SAT-encoded block world: Evolution of the best solution on a 116 variable problem with 953 clauses (medium.cnf).

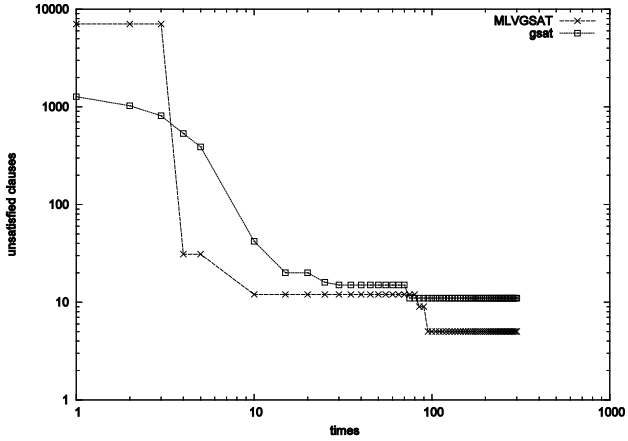

Fig. 28 Log-Log plot, SAT-encoded Block World: Evolution of the best solution on a 459 problem with 7054 clauses (huge.cnf).

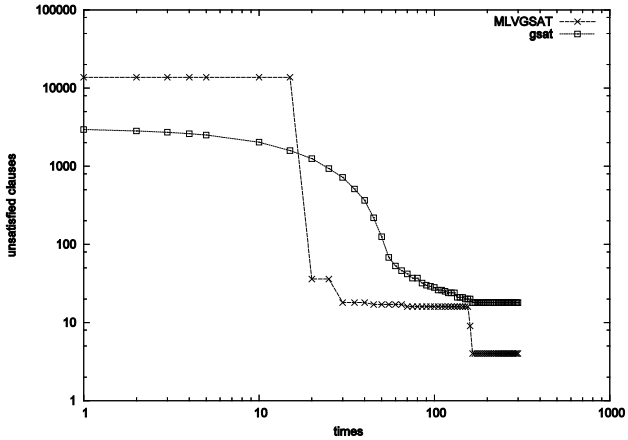

Fig. 29 Log-Log plot, SAT-encoded Block World: Evolution of the best solution on a 1087 variable problem with 13772 clauses (bw-largeb.cnf).

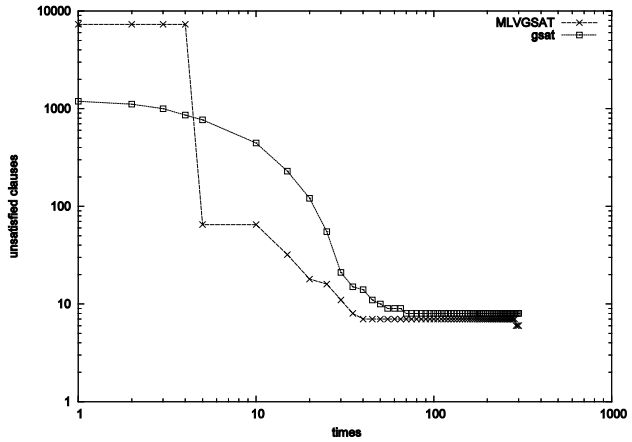

Fig. 30 Log-Log plot, SAT-encoded Logistics: Evolution of the best solution on a 843 variable problem with 7301 clauses (logisticsb.cnf).

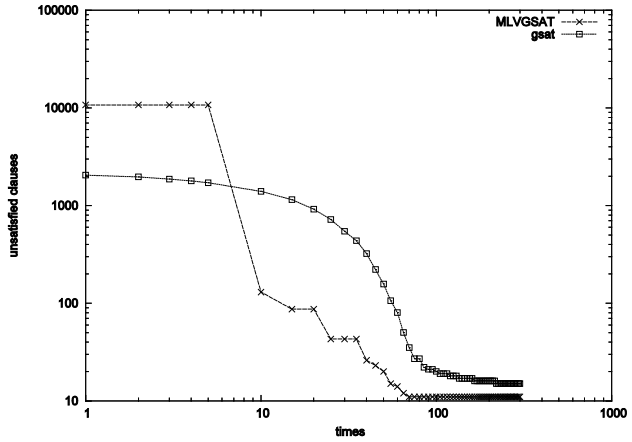

Fig. 31 Log-Log plot, SAT-encoded Logistics: Evolution of the best solution on a 1141 problem with 10719 clauses (logisticsc.cnf).

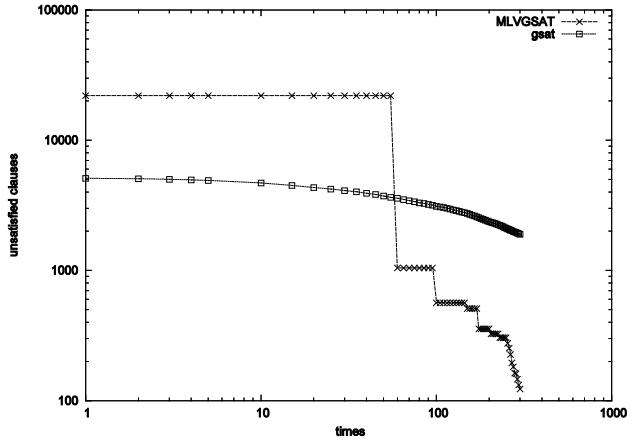

Fig. 32 Log-Log plot, SAT-encoded Logistics: Evolution of the best solution on a 4713 problem with 21991 clauses (logisticsd.cnf).

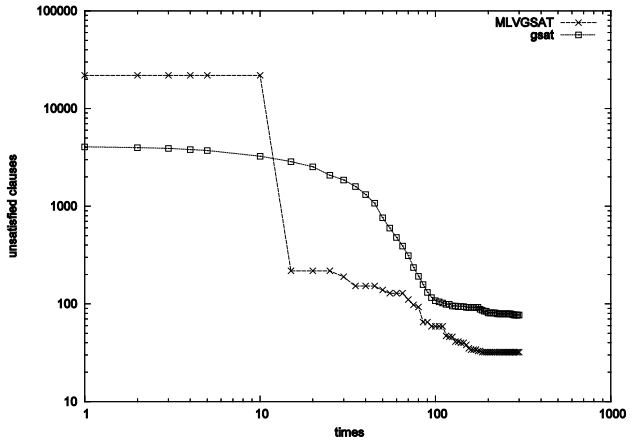

Fig. 33 Log-Log plot, SAT-encoded Quasigroup: Evolution of the best solution on a 129 variable problem with 21844 clauses (qg6-9.cnf).

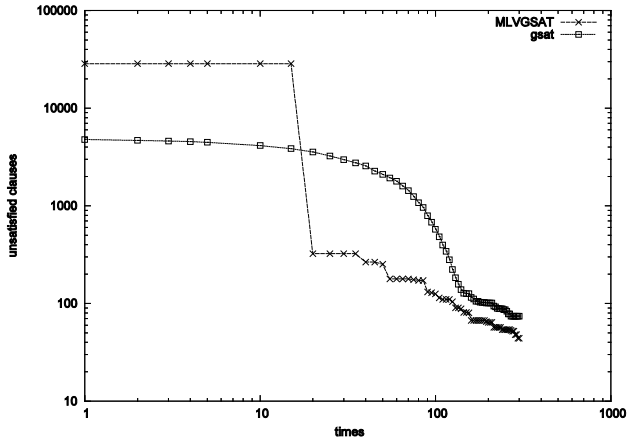

Fig. 34 Log-Log plot, SAT-encoded Quasigroup: Evolution of the best solution on a 729 variable problem with 28540 clauses (qg5-9.cnf).

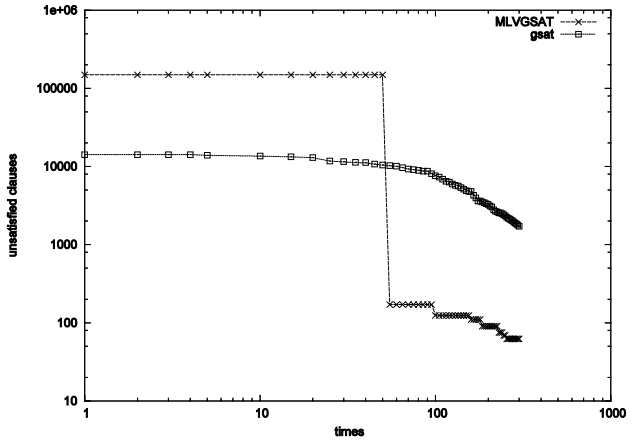

Fig. 35 Log-Log plot, SAT-encoded Quasigroup: Evolution of the best solution on a 512 variable problem with 148957 clauses (qg1-8.cnf).

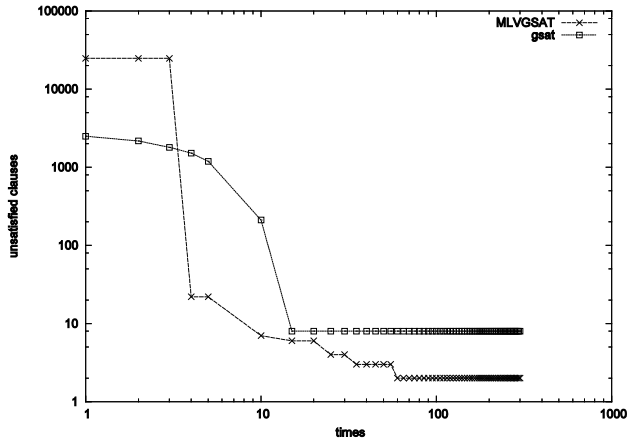

Fig. 36 Log-Log plot, SAT competition Beijing: Evolution of the best solution on a 410 variable problem with 24758 clauses (4blockb.cnf).

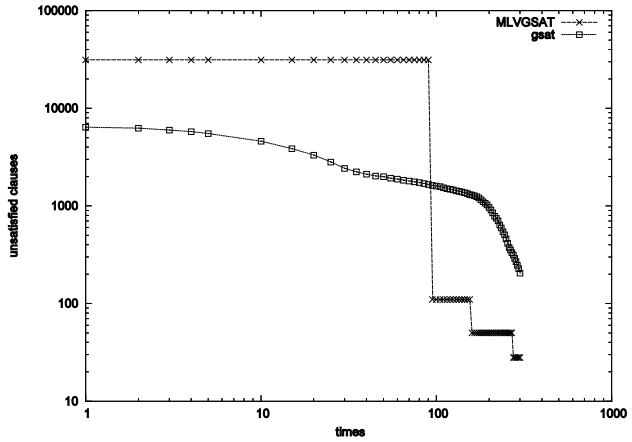

Fig. 37 Log-Log plot, SAT competition Beijing: Evolution of the best solution on a 8432 problem with 31310 clauses (3bitadd-31.cnf).

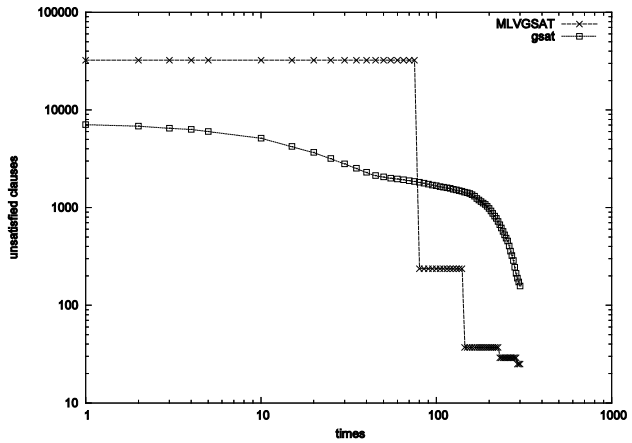

Fig. 38 Log-Log plot, SAT competition Beijing: Evolution of the best solution on a 8704 variable problem with 32316 clauses (3bitadd32.cnf).

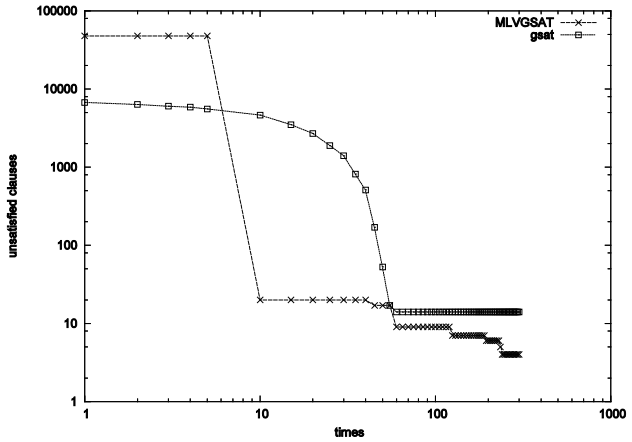

Fig. 39 Log-Log plot, SAT competition Beijing: Evolution of the best solution on a 758 problem with 47820 clauses (4blocks.cnf).

9. Wilcoxon Rank-Sum Test

The quality of the solution may vary significantly from run to run on the same problem instance due to random initial solutions and subsequent randomized decisions. We choose the Wilcoxon Rank test in order to test the level of statistical confidence in differences between the mean percentage excess deviations from the solution of the two algorithms. The test requires that the absolute values of the differences between the mean percentage excess deviations from the solution 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 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 $P < 0.05$.

Looking at Table 1, we observe that the difference in the mean excess deviation from the solution is significant for large problems and remains insignificant for small sized problems.

Table 1: Wilcoxon statistical test.

Problem	*%EX: MLVGSAT*	*%EX: GSAT*	*Null Hypothesis*
f600	0.001	0.004	Accept
f1000	0.002	0.009	Accept
f2000	0.002	0.01	Accept
flat100	0	0	Reject
g125-17	0.0002	0.001	Accept
g125-18	0.001	0.11	Accept
logistics-b	0.00008	0.0001	Accept
logistics-c	0.001	0.004	Accept
logistics-d	0.005	0.08	Accept
bw-medium	0	0	Reject
bw-huge	0.0007	0.001	Accept
bw-large-b	0.0002	0.001	Accept
qg6-9	0.001	0.003	Accept
qg5-9	0.001	0.002	Accept
qg1-8	0.0004	0.011	Accept
4block	0.00008	0.0008	Accept
3bitadd-31	0.0008	0.006	Accept
3bitadd-32	0.0007	0.004	Accept
4blocksb	0.00008	0.0002	Accept

10. Conclusions and Future Work

In this work, we have described and tested two new approaches in order to enhance GSAT. GSATRW suffers from stagnation behavior which directly affects its performance. This same phenomenon is however observed with LA-GSATRW only for large instances. Based on the analysis of RLD's, we observe that the probability of finding a solution within any arbitrary number of search steps is higher with GSATRW compared to that of LA-GSATRW.

Results indicated that the harder the problem, the better asymptotic convergence reached by LA-GSATRW as opposed with GSATRW.

The finite automaton learning mechanism employed in LA-GSATRW offers an efficient way to escape from highly attractive areas in the search space leading to a higher probability success as well as reducing the number of local search steps to find a solution.

As far as the second approach is concerned, we observe that within the same computational time, MLVGSAT provides high quality solution compared to that of GSAT. The broad conclusions that we may draw from the results are that the multilevel paradigm can either speed up (GSAT) or even improve its asymptotic convergence. Finally, obvious subjects for further work include the design of different coarsening strategies and tuning the refinement process so that more CPU time is given during the coarse levels.

References

[1] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, 35 (3) pp. 268-308, 2003.
[2] S.A. Cook. The complexity of theorem-proving procedures. Proceedings of the Third ACM Symposium on Theory of Computing, pp. 151-158, 1971.
[3] M. Davis and H. Putnam. A computing procedure for quantification theory. Journal of the ACM, 7, pp. 201-215, 1960.
[4] A.E. Eiben and J.K. Van der Hauw. Solving 3-SAT with Adaptive Genetic Algorithms. Proceedings of the 4th IEEE Conference on Evolutionary Computation, pp. 81-86. IEEE Press, 1997.
[5] B.S. Everit. The analysis of contingency tables. Chapman and Hall, London, 1977.
[6] W. Gale, S. Das, and C.T. Yu. Improvements to an Algorithm for Equipartitioning. IEEE Transactions on Computers, 39 (5), pp. 706-710, IEEE, 1990.
[7] M.R. Gary and D.S. Johnson. Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman and Company, New York, 1979.
[8] I. Gent and T. Walsh. Unsatisfied Variables in Local Search. In J. Hallam, editor, Hybrid Problems, Hybrid Solutions, pp. 73-85. IOS Press, 1995.
[9] L.P. Gent and T. Walsh. Towards an Understanding of Hill-Climbing Procedures for SAT. Proceedings of AAAI'93, pp. 28-33. MIT Press, 1993.
[10] F. Glover. Tabu Search-Part1. ORSA Journal on Computing, 1(3): 190-206, 1989.
[11] O.C. Granmo, B.J. Oommen, S.A. Myrer, and M.G. Olsen. Learning AutomataBased Solutions to the Nonlinear Fractional Knapsack Problem With Applications to Optimal Resource Allocation. IEEE Transactions on Systems, Man and Cybernetics, Vol.SMC-37(B), pp. 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), pp. 15-29, 2007.
[13] P. Hansen and B. Jaumand. Algorithms for the Maximum Satisfiability Problem. Computing, 44 pp. 279-303, 1990.
[14] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs.In S.Karin,editor, In Proceeding of Supercomputing'95, San Diego, 1995.ACM Press, New York.

[15] H. Hoos. On the run-time behavior of stochastic local search algorithms for SAT. In Proceedings of AAAI-99, pp. 661-666, 1999.
[16] H.Hoos. An adaptive noise mechanism for Walksat. In Proceedings of the Eighteen National Conference in Artificial Intelligence (AAAI-02), pp. 655-660, 2002.
[17] F. Hutter, D. Tompkins, H. Hoos. Scaling and probabilistic smoothing:Efficient dynamic local search for SAT. In Proceedings of the Eight International Conference of the Principles and Practice of Constraint Programming (CP'02), pp. 233-248, 2002.
[18] A. Ishtaiwi, J. Thornton, A. Sattar, and D.N.Pham. Neighborhood clause weight redistribution in local search for SAT. Proceedings of the Eleventh International Conference on Principles and Practice Programming(CP-05), volume 3709 of Lecture Notes in Computer Science, pp. 772-776, 2005.
[19] D.S. Johnson and M.A. Trick, editors. Cliques, Coloring, and Satisfiability, Volume 26 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.
[20] G.Karypis and V.Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs.SIAM J.Sci. Comput.,20 (1):359-392, 1998.
[21] G.Karypis and V. Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs. J.Par, Dist. Comput., 48(1):96-129, 1998.
[22] A.R. KhudaBukhsh, L. Xu, H. Hoos, K. Leyton-Brown. SATenstein: Automatically building local search SAT solvers from components. In Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-09), 2009.
[23] C.M. Li, W. Wei, and H. Zhang. Combining adaptive noise and look-ahead in local search for SAT. Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing(SAT-07), volume 4501 of Lecture Notes in Computer Science, pp. 121-133, 2007.
[24] C.M. Li, W.Q. Huang. Diversification and determinism in local search for satisfiability. In proceedings of the Eight International Conference on Theory and Applications of Satisfiability Testing (SAT-05), volume 3569 of Lecture Notes in Computer Science, pp. 158-172, 2005.
[25] C.M Li, W. Wei, and H. Zhang. Combining adaptive noise and look-ahead in local search for SAT. In Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT-07), volume 4501 of Lecture Notes in Computer Science, pp. 121-133, 2007.
[26] D. McAllester, B. Selman, and H.Kautz. Evidence for Invariants in Local Search. In Proceedings of AAAI'97, pp. 321-326. MIT Press, 1997.
[27] 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), pp. 1179-1192, 2005.
[28] K.S. Narendra and M.A.L. Thathachar. Learning Automata: An Introduction. Prentice Hall, 1989.
[29] B.J. Oommen and E.R. Hansen. List organizing strategies using stochastic moveto-front and stochastic move-to-rear operations. SIAM Journal on Computing, 16, SIAM, pp. 705-716, 1987.
[30] B.J. Oommen and D.C.Y. Ma. Deterministic Learning Automata Solutions to the Equipartitioning Problem. IEEE Transactions on Computers, 37,1, pp. 2-13, IEEE, 1988.
[31] B.J. Oommen and E.V. St.Croix. Graph partitioning using learning automata. IEEE Transactions on Computers, 45, 2, pp.195-208, IEEE, 1996.
[32] 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),pp. 821-831, 2002.
[33] D.J. Patterson and H. Kautz. Auto-Walksat: A Self-Tuning Implementation of Walksat. Electronic Notes on Discrete Mathematics 9, 2001.
[34] S. Prestwich. Random walk with continuously smoothed variable weights. Proceed- ings of the Eight International Conference on Theory and Applications of Satisfiability Testing(SAT-05), volume 3569 of Lecture Notes, pp. 203-215, 2005.
[35] D. Schuurmans, and F. Southey. Local search characteristics of incomplete SAT procedures. In Proc.AAAI-2000, pp. 297-302, AAAI Press, 2000.
[36] D. Schuurmans, F.Southey, and R.C. Holte. The exponentiated sub-gradient algorithm for heuristic Boolean programming. In Proc. IJCAI-01, pp. 334-341, Morgan Kaufman Publishers, 2001.
[37] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. Proceedings of AAA'92, pp.440-446, MIT Press, 1992.
[38] B. Selman, H.A. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. Proceedings of AAAI'94, pp. 337-343. MIT Press, 1994.
[39] B. Selman and H.A. Kautz. Domain-Independent extensions to GSAT: Solving large structured satisfiability problems. In R.Bajcsy,editor, Proceedings of the in ternational Joint Conference on Artificial Intelligence,1, pp. 290-295. Morgan Kaufmann Publishers Inc., 1993.
[40] M.A.L. Thathachar and P.S. Sastry. Network of Learning Automata: Techniques for On line Stochastic Optimization. Kluer Academic Publishers, 2004.
[41] J. Thornton, D.N. Pham, S. Bain, and V. Ferreira Jr. Additive versus multiplicative clause weighting for SAT. Proceedings of the Ninteenth National Conference of Artificial Intelligence (AAAI-04), pp. 191-196, 2004.
[42] M.L. Tsetlin. Automaton Theory and Modeling of Biological Systems. Academic Press, 1973.
[43] C. Walshaw and M. Cross. Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm. SIAM J.Sci. Comput.,22(1):63-80,2000.
[44] Z.Wu., and B.Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00), pp. 310-315, 2000.
[45] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research, 32 (1)pp.565-606, 2008.

Noureddine Bouhmala obtained his MSc from Federal Polytechnic of Lausanne 1994 and PhD from the University of Neuchatel in Switzerland. He is currently working at the

Department of Maritime Technology and Innovation at Vestfold University College, Norway. His research interests include combinatorial optimization, data mining, and parallel computing.

Ole-Christoffer Granmo was born in Porsgrunn, Norway. He obtained his M.Sc. in 1999 and the Ph.D. degree in 2004, both from the University of Oslo, Norway. He is currently a Professor in the Department of ICT, University of Agder, Norway. His research interests include Intelligent Systems, Stochastic Modelling and Inference, Machine Learning, Pattern Recognition, Learning Automata, Distributed Computing, and Surveillance and Monitoring. He is the author of more than 55 refereed journal and conference publications.