UNIVERSITY OF AGDER

Solving the Boolean Satisfiability Problem Using Multilevel Techniques

Sirar Salih
Yujie Song

Supervisor
Associate Professor Noureddine Bouhmala

This Master's Thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education.

University of Agder, 2011
Faculty of Engineering and Science
Department of ICT

To our beloved family, here and abroad, and to the beauty of science.

Abstract

There are many complex problems in computer science that occur in knowledge-representation (artificial thinking), artificial learning, Very Large Scale Integration (VLSI) design, security protocols and other areas. These complex problems may be deduced into satisfiability problems where the Boolean Satisfiability Problem (SAT) may be applied. This deduction is made in order to simplify complex problems into a specific propositional logic problem. The SAT problem is the most well-known nondeterministic polynomial time (NP) complete problem in computer science. It is a Boolean expression which is composed of a specific amount of variables (literals), clauses that contain disjunctions of the literals and conjunctions of the clauses. The literals have the logical values TRUE and FALSE, the task is to find a truth assignment that makes the entire expression TRUE. The main goal of the thesis is to solve the SAT problem using a clustering technique - Multilevel - combined first with Tabu Search and combined thereafter with finite Learning Automata. Tabu Search and finite Learning Automata are two very efficient approaches that have been used to solve SAT. Benchmark experiments are conducted in order to disclose whether combining Multilevel with existing solutions to solve SAT will provide better results - than the two mentioned approaches alone - mainly in terms of computational efficiency.

Preface

Imagination is more important than knowledge. - Albert Einstein

This report is the documented result of the master's thesis in the IKT590 (Master's Thesis) course in the master programme at the Faculty of Engineering and Science, Department of ICT at University of Agder in Grimstad, Norway.

The main purpose of the IKT590 course is to allow students to write their master's thesis as a final step of their master programme education at the University of Agder. The work performed by the students in the thesis must be scientific and elements of the work must be new, contributing knowledge. The work is to be presented by a theoretical report that describes the problem and results, along with a poster and an oral presentation at the university.

The project group consists of students Sirar Salih and Yujie Song who attend the Information and Communication Technology master programme, at the University of Agder. The first student has a bachelor degree in computer science from the University of Agder and the second student has a bachelor degree in electrical engineering from the University of Wuhan in Wuhan, China. The work on this thesis and the writing of this report was done at the University of Agder. The supervisor of the thesis is associate professor Noureddine Bouhmala.

Publication

The work conducted in this thesis and some of the experimental results have been included in the paper *A Tabu Search Algorithm Combined with Learning Automata for the Satisfiability Problem* by N. Bouhmala, O-C. Granmo, Sirar Salih and Yujie Song, to be submitted for publication as a chapter in book.

Acknowledgements

We would like to thank our supervisor, associate professor Noureddine Bouhmala for his continuous support and guiding throughout the thesis period. Especially for his patience, it is safe to say that without his ideas and assistance the thesis would have faced endless difficulties. We would also like to thank our contact at the University of Agder, associate professor Ole-Christoffer Granmo for providing us the opportunity to start and complete this thesis.

Grimstad
25 May 2011
Sirar Salih
Yujie Song

Contents

List of Figures

List of Tables

1 Introduction

In this chapter the background of the problem is explained in detail. The problem is then stated, the hypothesis, the motivation and the limitations and key assumptions. Finally a literature review is given followed by a short outline of the rest of the thesis report.

1.1 Background

There are many complex problems in computer science that occur in knowledge-representation (artificial thinking), artificial learning, Very Large Scale Integration (VLSI) design, security protocols and other areas. These complex problems may be deduced into satisfiability problems where the Boolean Satisfiability Problem (SAT) may be applied. This deduction is made in order to simplify a complex problem into a specific mathematical problem. Once the deduction is made, one only needs to solve the SAT problem in order to solve the more complex problem. Therefore, efficient ways to solve the SAT problem draw a growing attention in the field of computer science.

One example application is the SAT-based analysis of protocol insecurity problems in [1]. In this paper, A. Armando and L. Compagna from the University of Florence in Italy have managed to represent protocol insecurity problems as SAT, and have built an automatic model-checker for security protocols based on SAT solver algorithms. By doing this, they could use the model-checker to help solve the complex protocol insecurity problems. Similarly, F. Guillaume presented in his paper the SAT representation of MU-calculus over Petri Nets [2]. The model checking problem for Petri Nets has been known to be undecidable for almost fifteen years [3]. Guillaume showed that this undecidability can be represented in SAT, making the problem context much simpler.

The ability to represent a complex problem as a propositional logic problem such as SAT, makes things very easy in terms of solving complexity - since one only needs to satisfy the set of logical values. As a result of this, efficient ways to solve SAT are also important. There has been an increase in the development of SAT solver algorithms. Two notable approaches to solve SAT are Tabu Search and finite Learning Automata, these two approaches have recently been proved to be very efficient. However, there is no boundary on the efficiency aspect and it is believed that the efficiency of the two latter approaches could still be increased. Because the two mentioned approaches use a single level technique, which could be replaced by a multi level that gives a better sampling of the solution space.

1.2 Problem and Hypothesis

The SAT problem is a well-known nondeterministic polynomial time (NP) complete [1] problem in computer science. It is composed of an N amount of literals, clauses that contain disjunctions of the literals and conjunctions of the clauses. The literals can either have the value TRUE or FALSE. To solve the SAT problem, the total set of clauses must give the value TRUE; it is then said that the problem is satisfied. A SAT problem with two literals, two clauses and two literals per clause is shown in figure 1.

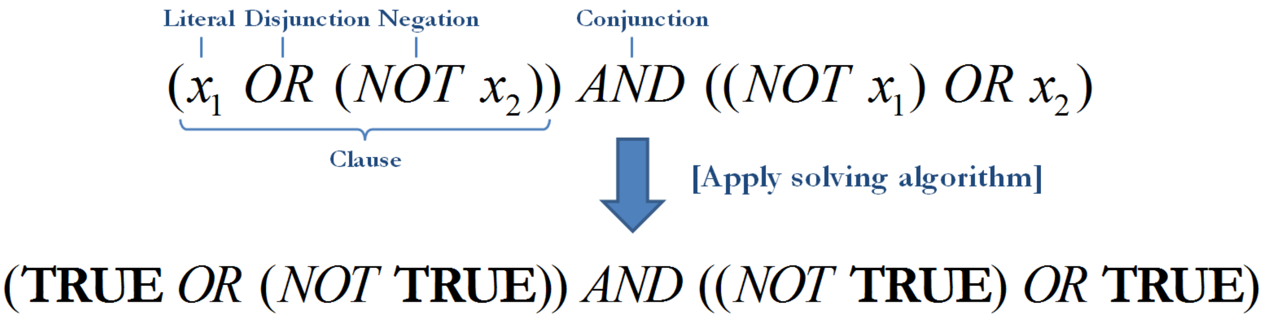

Figure 1: The Boolean Satisfiability Problem. After solving SAT, all clauses get the logical value TRUE.

The problem is represented as the following propositional formula:

$$\Phi = \bigwedge_{j=1}^{m} C_j$$

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

Where C_j is the disjunction of literals, m is the number of clauses, n is the number of literals and x_i is a literal, $i \in \{1, \dots n\}$. $I_j, \bar{I}_j \subseteq \{1, \dots n\}$, $I_j \cap \bar{I}_j = \emptyset$ and $\bar{x}_i$ denotes the negation of x_i. The assignment is satisfied if the propositional formula Φ evaluates to TRUE. This formula representation can be found in [3].

As the growing need of efficient ways to solve SAT continues, it is in this paper hypothesised that combining the Multilevel technique with existing approaches will drastically increase efficiency of solving SAT. The reason to this is that the Multilevel technique simplifies the problem drastically by clustering literals together. Tabu Search and Finite Learning Automata are two existing approaches (see [13] and [3], respectively) which have been proved to be very efficient methods to solve SAT, therefore these two approaches have been chosen to be combined with the Multilevel technique, in order to prove if combining Multilevel will increase the efficiency to solve SAT or not. This combination will create the Multilevel Tabu Search and Multilevel Learning Automata algorithms.

1.3 Importance of Topic

Many complex problems in computer science can be simplified by representing them in SAT, therefore SAT is very important for helping solve complex problems in computer science and has played a major role. One example application is the SAT-based analysis of protocol insecurity problems in [4]. In this paper, A. Armando and L. Compagna from the University of Florence in Italy have managed to represent protocol insecurity problems as SAT, and have built an automatic model-checker for security protocols based on SAT solver algorithms. By doing this, they could use the model-checker to help solve the complex protocol insecurity problems. Similarly, F. Guillaume presented in his paper the SAT representation of MU-calculus over Petri Nets [5]. The model checking problem for Petri Nets has been known to be undecidable for almost fifteen years [6]. Guillaume showed that this undecidability can be represented in SAT, making the problem context much simpler.

The ability to represent a complex problem as a propositional logic problem such as SAT, makes things very easy in terms of solving complexity - since one only needs to satisfy the set of logical values. As a result of this, efficient ways to solve SAT are also important. There has been an increase in the development of SAT solver algorithms. Two notable approaches to solve SAT are Tabu Search and finite Learning Automata, these two approaches have been proved to be very efficient. However, there is no boundary on the efficiency aspect and it is believed that the efficiency of the two mentioned approaches could still be increased. Because these two approaches use a single level technique, which could be replaced by a multi level that gives a better sampling of the solution space. Therefore, combining Multilevel technique with these two existing methods may increase the efficiency of using them alone.

If the hypothesis is proved to be true, the research will introduce a new, more efficient way of solving SAT by introducing the Multilevel technique. For example, combining Multilevel with Tabu Search or combining Multilevel with finite Learning Automata will become the new, more efficient way to solve SAT.

1.4 Motivation

If the problem is solved, the research will introduce a new, more efficient way of solving SAT by introducing the Multilevel technique. Adding a new algorithm approach to the collection of solver algorithms is a step further of solving complex problems that can be represented as SAT, and an advancement in the science of SAT.

If however not solved, the work will be used further for research. Continuing the research will no doubt improve the proposed solutions in this thesis.

1.5 Limitations and Key Assumptions

The implementation of our proposed Multilevel Tabu Search and Multilevel Learning Automata algorithms will be done in the C++ programming language. Although Tabu Search and Learning Automata algorithms may require much work to be implemented very efficiently in C++, they will still be implemented in this research, because the efficiencies of these two algorithms have to be used to compare with the efficiencies of the two new proposed algorithms. However, we cannot state with 100 % certainty that the combination of the Multilevel technique is the reason to the efficiency increase or decrease. That is because personal programming experiences might have side effects on the implementation results. The implementation will also prove difficult due to the nature of the context, thus a clear understanding of the problems prior to implementation is a vital step.

The Multilevel technique is assumed to increase the efficiency of existing SAT solver algorithms, for example, Tabu Search and Learning Automata. However, if this hypothesis is disproved, then the implementation will need to be revised and fixed. Attempts will be made to find out the cause, it is expected that this part will take a significant amount of time. If the hypothesis is disproved and also no reasonable cause could be found, then a discussion will be engaged as to why this happened. Ideas for further work will also be provided.

1.6 Contributions to Research

Potential outcomes of the research are summarized in the list below.

- Simplification of SAT instances prior to solving them by using the Multilevel clustering technique, because literals can be clustered together. This allows metaheuristic algorithms to handle clusters of literals as a single entity, making the search space guided and restricted to only those literals within the clusters. This offers a better sampling of the solution space compared to single level computations.

- Increase in SAT solving efficiency. Applying the Multilevel clustering technique will increase the efficiency of solving SAT instances; this is due to the previous bullet point.

- Introduction of two new, efficient SAT solver algorithms. Given that the hypothesis is proved, the thesis will introduce two, new SAT solver algorithms; Multilevel Tabu Search and Multilevel Learning Automata. Based on the properties of Multilevel, these two new algorithms will be more efficient than their predecessors.

1.7 Literature Review

Many complex problems have been successfully represented as SAT, equally many efficient algorithms have been implemented to solve the latter and the state-of-the-art is very wide. The focus here is on the literature of two specific (as an example) complex problems and the most popular solver algorithms for SAT; local search algorithms. This chapter gives a quick review of all the relevant papers, while chapter 2 provides an in-depth explanation of each.

In their paper, The SAT-based Analysis of Protocol Insecurity Problems [1], A. Armando and L. Compagna from the University of Florence in Italy managed to represent protocol insecurity problems as SAT, in an attempt to build an automatic model-checker for security protocols based on SAT solver algorithms. Similarly, F. Guillaume presented in his paper the SAT representation of MU-calculus over Petri Nets [2]. The model checking problem for Petri Nets has been known to be undecidable for almost fifteen years [3]. Guillaume showed that this undecidability can be represented in SAT, making the problem context simpler.

B. Selman, H. Levesque and D. Mitchell presented in their paper a new method for solving hard SAT problems; GSAT [7]. GSAT is one of the most popular local search algorithms that has been used to solve SAT. B. Selman, Henry A. Kautz and B. Cohen made an extension of GSAT; GSAT with Random Walk [8] with the purpose of escaping local optima, thus preventing stagnation. Another variant of GSAT is Walk SAT [9], introduced by D. McAllester, B. Selman and H. Kautz.

W. M. Spears presented in his paper the Simulated Annealing (SASAT) [12] algorithm which managed to scale up better as the number of literals increased and managed to solve many hard SAT instances with little effort.

A.E. Eiben and J.K. van der Hauw from Leiden University in The Netherlands presented in their paper a way of adapting Genetic Algorithms [13] (GAs) that increases GAs' performance of solving 3-SAT (3 literals pr. clause) instances. This adaptation called Stepwise Adaptation of Weights (SAW).

B. Mazure, L. Saïs and E. Gregoire presented in their paper the Tabu Search (TSAT) [14] algorithm.

Associate professors O-C. Granmo and N. Bouhmala from the Univeresity of Agder and Vestfold University College in Norway wrote the first paper on combining finite Learning Automata with traditional Random Walk algorithm [6] to solve SAT.

B. Cha and K. Iwama presented in their paper [15] a way of assigning weight values to SAT clauses. J. Frank wrote an extensive study on the same method in his paper [16].

P. Hansen and B. Jaumand, I. Gent and T. Walsh presented in their papers [25, 26, 27] algorithms using history based literal selection strategies.

1.8 Thesis Report Outline

The rest of the thesis report is structured as follows:

In chapter 2 the theoretical background of the problem is given. Here the state-of-the-art is discussed in detail giving an insight of the background and prior work. Significant prior work is discussed in this chapter.

In chapter 3 the proposed solutions are discussed. An in-depth explanation of the solutions is provided here including the pseudo-code of each proposed solution, and the best solutions are then selected in this chapter.

In chapter 4 the experimental results are presented in the form of running benchmark tests and a comparative analysis of the algorithms is made.

In chapter 5 the experimental results of the algorithms from chapter 4 are discussed in detail, focusing on efficiency among other factors.

In chapter 6 a brief look is made on the problem, the proposed solutions to the latter and the outcome of the experimental results. The hypothesis and further work are also briefly discussed in this chapter.

2 Significant Prior Research

The focus in this chapter is on significant prior work. Efficient methods that have been used to solve SAT will be explained in the following sections. Local search algorithms have been widely used to solve SAT. This is due to their ability to give up completeness. Since SAT is NP-complete, local search algorithms are therefore appropriate to use in contrast to systematic search algorithms which are guaranteed to return a solution to a problem, or otherwise prove it unsolvable. In the following chapters, the focus will be on these.

2.1 Solving SAT Using GSAT

GSAT is one of the most famous local search algorithms that have been used to solve SAT. B. Selman, H. Levesque and D. Mitchell introduced GSAT in their paper as a new method for solving hard satisfiability problems [7]. GSAT randomly assigns TRUE values to the literals, it then flips the assignment of the literals that lead to the largest increase in the total number of satisfied clauses. The flips are repeated until either the problem is solved or a maximum number of flips (MAX-FLIPS) is reached. The process is repeated up to a maximum number of tries (MAX-TRIES). So basically, GSAT performs greedy local search. The pseudo code below shows the GSAT procedure.

```
Procedure GSAT
Begin
for i:= 1 to MAX-TRIES
    T := a randomly generated TRUE assignment
    for j:= 1 to MAX-FLIPS
        if T satisfies set_of_clauses then return T
        p := a propositional value such that a change
               in its TRUE assignment gives the largest
               increase in the total number of clauses
              of set_of_clauses that are satisfied by T
        T := T with TRUE assignment of p reversed
    end-for
end-for
return "no satisfying assignment found"
End
```

A comparative analysis of GSAT and Davis-Putman (DP) [10] was made in [7]. The latter is a systematic search algorithm which does a backtracking search on all TRUE assignments, assigning values to each literal. It returns a solution to the problem if it exists and does not give up completeness. For more on systematic searching, the reader is referred to [10]. From the results that can be seen in [7], GSAT is clearly better than DP. The former is faster than the latter in terms of efficiency and since the latter is a systematic search algorithm, it does not even return a solution to problems it cannot solve.

2.1.1 GSAT with Random Walk

An extension of GSAT is GSAT with Random Walk [8]. The idea of this extension is to escape local optima and avoid stagnation. When a random walk move is made, a randomly unsatisfied clause is selected, then one of the literals in the clause is flipped thus satisfying the selected clause. The idea is to decide at each step whether to perform a GSAT or Random Walk move. As can be seen in [8], GSAT with Random Walk solves more problems than its predecessor and doing so more efficiently.

2.1.2 Walk SAT

Another variant of GSAT is Walk SAT [9], introduced by D. McAllester, B. Selman and H. Kautz. Walk SAT maintains a "break count" associated with each literal. The break count is the number of clauses that would be unsatisfied by flipping the literal associated with that break count. An unsatisfied clause is first randomly picked, then the literal with the lowest break count is then randomly selected. One of the other literals in the clause may also be selected with a certain probability. The random picking of unsatisfied clauses and the random selection of literals inside helps Walk SAT to escape local optima and avoid stagnation. This also adds to the exploration factor of the search space.

B. Ferris and J. Froehlich from the University of Washington in the US made a comparative analysis of Walk SAT and a systematic search algorithm called DPLL [11] DPLL enumerates all possible assignment models in the search space. For more on the latter, the reader is referred to [11]. As can be seen in [9], the ISR of Walk SAT is relatively low in normally distributed and hard random problems. It can also be observed that DPLL has a harder time solving SAT instances than Walk SAT. As the clause/literal ratio increases, DPLL is gradually weakened whilst Walk SAT manages to solve the problems.

2.2 Solving SAT Using Simulated Annealing

Simulated Annealing [12] is an algorithm that outperformed GSAT [7] in the context of neural networks. Dropping the latter and focusing on SAT, the algorithm is deduced into SASAT. SASAT has a structure which is similar to GSAT, the pseudo code below shows the SASAT procedure.

```
Procedure SASAT
Begin
Input: number_of_clauses, MAX_TRIES, MAX_TEMP, MIN_TEMP
Output: T
i = 0 tries=0
while (tries < MAX_TRIES) do
   randomly assign TRUE/FALSE values to the literals
   T = number_of_trues
   while (T < number_of_clauses) do
         temperature = MAX_TEMP × e^(-j×decay_rate)
         if (temperature < MIN_TEMP) then break
        for v=1 to number_of_literals
           flip v
           Compute gain
           flip v
           flip v with probability 1 / (1 + e^(-gain/temperature))
           if v was flipped then update T
       end-for
       j++ tries++
    end-while
```

```
    i++
end-while
End
```

The outer while-loop generates a random solution for every iteration, this provides independent attempts at solving the problem. The temperature is set to MAX_TEMP for every iteration. The inner while-loop probabilistically updates the number of TRUE clauses based on the gain provided by the flip. Based on the function - which is the standard logistic function for simulated annealing - used, if the gain is positive then the flip is likely to be performed. Likewise, if the gain is negative then the flip is unlikely to be performed. The temperature measure is used to control the moves of SASAT. If the temperature is high, the moves are almost random. If the temperature is low, then the moves are similar to those of GSAT. As j increases, the temperature decreases according to the decay rate. When MIN_TEMP is reached, i is incremented and the algorithm tries again to solve the problem by randomly assigning TRUE/FALSE values to the literals. The decay rate is set as follows:

$$decay\_rate = \frac{1}{i \times number\_of\_literals}$$

Each time i is increased, the decay rate is decreased. Reducing the decay rate for every iteration of the outer while-loop allows the algorithm to perform more flips during each iteration of the inner while-loop. MAX_TEMP is set to 0.3 and MIN_TEMP to 0.01 [12]. What is desired here is to reduce the number of independent attempts to be able to search thoroughly during each given attempt. This is possible by increasing the temperature or decreasing the decay rate. According to Spears, it is not clear whether it is better to make more independent attempts or to search thoroughly during each attempt.

It was difficult for Spears to make a proper comparison between SASAT and GSAT because of the metrics used for measurement. However, using a combination of gains and flips, Spears was able to illustrate that SASAT scaled better on larger problems while GSAT had an advantage on easier problems [12]. SASAT managed to solve a higher percentage of problems doing fewer flips, while GSAT solved only few problems.

Since a proper comparison between the algorithms was difficult to make, Spears made a slight modification to SASAT to make it more similar to GSAT by using a zero temperature logistic function [12]. Spears then compared SASAT, zero temperature SASAT and GSAT. Zero temperature SASAT did indeed behave like GSAT, and it was observed that SASAT outperformed zero temperature SASAT, consequently outperforming GSAT.

2.2.1 SASAT with Random Walk

Similar to GSAT, SASAT is enhanced with a Random Walk approach. Recall that the purpose of Random Walk is to allow the algorithm to escape local optima (by randomly choosing an unsatisfied clause and randomly flipping a literal inside that clause), the same occurs in SASAT having the following modification to the algorithm:

```
flip v with probability p

 probability p{
        if v is in an unsatisfied clause then return 1.0
        else return 0.0
        }
 else probability 1 - p{
```

return $\dfrac{1}{1+e^{-\frac{gain}{temperature}}}$

So with probability p, if the literal is inside an unsatisfied clause, it is flipped. Otherwise, it is not flipped. With probability 1 - p, the standard logistic function is used. Doing this, the random walk moves are focused on the clauses that are difficult for the algorithm to handle. p is set to $\dfrac{1}{number\_of\_literals}$ [12].

This modification of SASAT slightly increased the performance of the algorithm, however according to Spears it is not clear whether the random walk, the annealing schedule or a combination of the two is the reason to the performance increase. This remains to be investigated in the future.

2.3 Solving SAT Using Adaptive Genetic Algorithms

Genetic Algorithms (GAs) have a challenge solving NP-complete problems such as SAT. Much of the challenge is due to constraints that make finding solutions to the problems difficult. A. E. Eiben and J. K. van der Hauw presented in their paper [13] a way of adapting constraints in the form of weights in order to solve 3-SAT problems. They called this method Stepwise Adaptation of Weights (SAW). They proved in their paper that using this method increased the performance of GAs and it made the latter superior to another heuristic method - WGSAT. WGSAT is a modification of GSAT which is based on [22] where each clause in a SAT problem is associated with a weight and the weights of all unsatisfied clauses at the end of a try are updated. In WGSAT, the weights are updated after each flip instead of after each try [23, 24].

The genetic representation of SAT is a bit representation where each literal is represented by a gene that can have the value 0 for FALSE and 1 for TRUE. A chromosome then represents a given clause. A fitness function is the truth value of the chromosome. In the case of SAT, the whole fitness landscape is not known. In [13], bit representation is used and a fitness function that counts unsatisfied clauses. 2-tournament selection and worst fitness deletion is applied. The maximum fitness evaluations is set to 300 000 and each problem instance is run 50 times [13]. The success rate (SR) is the percentage of all cases where a solution was found. Several tests were performed in order to find the best operators and optimal population sizes. The SAW procedure is shown on the next page.

Procedure **SAW**
Begin
initialize weights (and get fitness function f)
while not termination do
    for i=0 to T_p _fitness_evaluations
        run GA with f
    end-for
    get new f and recalculate fitness of individuals
end-while
End

SAW provides the ability to not needing to set constraint weights, hence removing the possibility of wrongly defining constraint weights (which gives bad results). Once the T_p fitness evaluations is reached, the best individual in the population is taken and the weights of the constraints that it violates are increased ($w_i = w_i + \Delta w$). Using the SAW mechanism increased the success rates in GA at the cost of more evaluations [13]. SAW-ing GA also gave better results than WGSAT in all cases that were tested. In addition, GAs were compared with traditional SAT solving heuristics (not mentioned which, however) and results showed that SAW-ing GAs outperformed these heuristics [13].

2.4 Solving SAT Using Tabu Search

Tabu Search (TSAT) [14] has been proved as an efficient method in solving SAT. Many local search algorithms tend to stagnate while attempting to solve SAT after an amount of time, that is being unable to generate a flip that will make a difference in the results - thus giving an incorrect (unsatisfied) result (local minima). TSAT avoids this problem by maintaining a so called tabu list. The tabu list contains information about the literals, it does this to avoid recurrent flips and thus escape local minima. The tabu list is updated each time a flip is made. The list is a fixed length, chronologically ordered, First In First Out (FIFO) list of flipped variables [14]. Using the list, TSAT prevents the variables in the list from being flipped again during the computation. Figure 2 illustrates a tabu list.

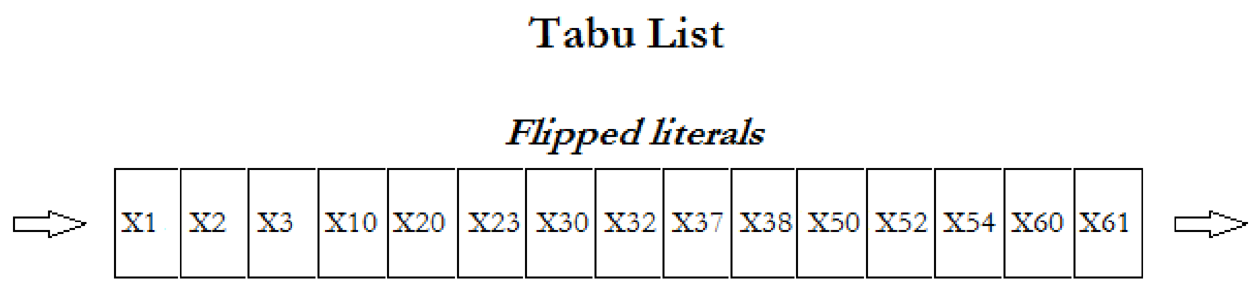

Figure 2: A tabu list contains chronologically ordered flipped literals in a FIFO fashion.

B. Mazure, L. Saïs and E. Gregoire from the University of d'Artois in France showed in their paper that the length of the tabu list plays a major role in the performance of the algorithm [14]. To that end, the optimal length of the tabu list is desired. The curve illustrated by Mazure, Saïs and Gregoire appears to be linear in the number of literals given. That is:

$$\text{optimal length of tabu list} = 0.01875 \times n + 2.8125$$

where n = number of literals

A slight change of the optimal length of a tabu list, leads to a decrease in the performance of TSAT. Similarly, a big change leads to a dramatic decrease of performance. As seen, the optimal length depends on the number of variables.

A comparative analysis was made by Mazure, Saïs and Gregoire of TSAT and Random Walk GSAT (RW-GSAT) [14]. The former solved more problems than the latter and showed better performance. As can be seen from the results in [14], TSAT successfully managed to satisfy more clauses than RW-GSAT in each SAT problem using less time and making fewer flips overall. Based on these results, TSAT is no doubt more efficient in solving SAT instances than RW-GSAT.

2.5 Solving SAT Using Finite Learning Automata

Another efficient method to solve SAT is to use finite Learning Automata. Associate professors O-C. Granmo and N. Bouhmala from the Univeresity of Agder and Vestfold University College in Norway wrote the first paper on combining finite Learning Automata with traditional Random Walk algorithm [6] to solve SAT. They presented a comparative analysis of the algorithm's efficiency, by solving benchmark sets containing SAT instances as well as SAT-represented problems from various complex domains.

Learning Automata have been successful in solving many optimization problems including the Equipartitioning Problem [17, 18], the Graph Partitioning Problem [19] and the List Organization Problem [20]. Learning Automata excels in solving problems due to their ability to learn the optimal actions when operating in unknown, stochastic environments. In addition, they combine fast and accurate convergence with low computational complexity [6].

In their paper, associate professors Granmo and Bouhmala defined a learning SAT automaton as well as an unknown environment that the automaton would interact with. A finite learning automaton interacts with the environment by performing actions, the environment then responds to each action with some sort of reward or penalty based on that action. Based on the responses from the environment, the aim of the automaton is to find the action that minimizes the number of penalties received. Figure 3 illustrates the interaction between an automaton and the environment.

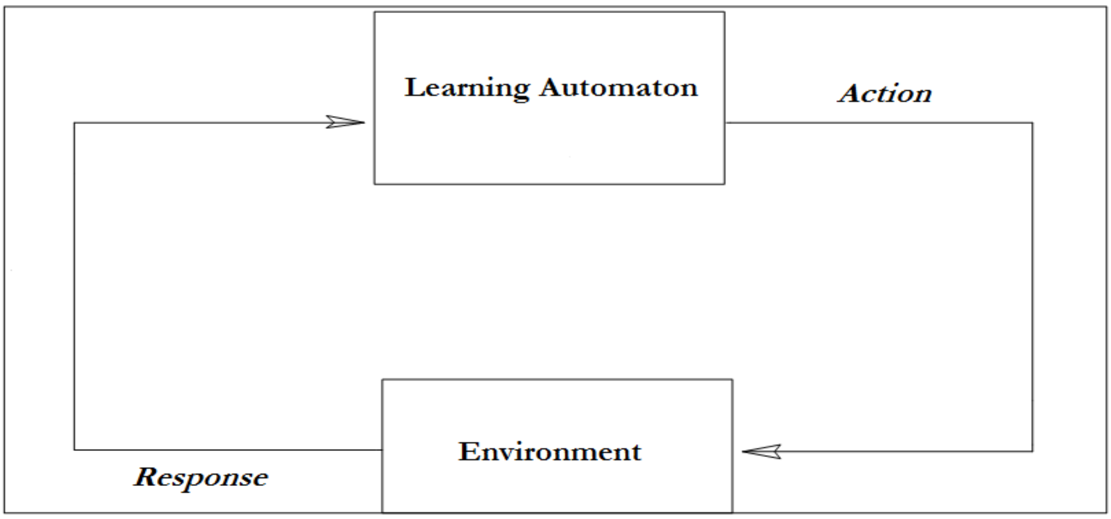

Figure 3: A learning automaton sends an action to an environment, which responds with either a reward or penalty. [6]

Each literal in SAT is assigned a learning automaton, which results in a team of learning automata. The goal of the Learning Automata is to find the solution of the SAT instance. Figure 4 illustrates each automaton associated with a literal.

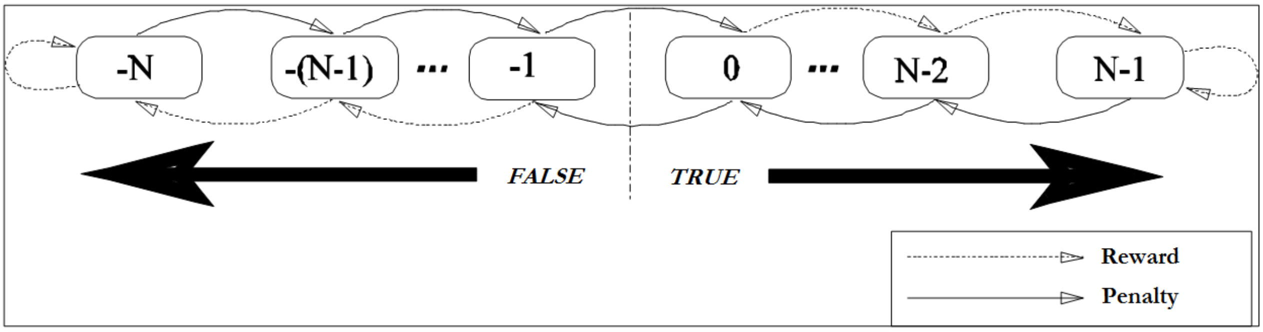

Figure 4: Each literal in SAT is assigned a learning automaton. If the automaton state is positive, action TRUE is chosen by the automaton. If the state is negative, action FALSE is chosen. [6]

If the state of the learning automaton is positive, then the action TRUE will be performed by the automaton. If the state is negative, then the action FALSE will be performed. The optimal action is not known initially, therefore the initial state of each automaton is randomly set to either -1 or 0.

The environment is the SAT instance. Providing a reward response from the environment to the automaton strengthens the currently chosen action, this makes it less likely that the other action will be chosen in the future. Similarly, a penalty response weakens the current action by making it more likely that the other action will be chosen in the future.

Since SAT is NP-complete, local search algorithms have been used to solve SAT because they give up completeness. Granmo and Bouhmala combined Learning Automata with Random Walk algorithm and below are the steps of how to use this new algorithm.

1. *Each LA assigns a truth value to its corresponding variable.*

2. *Pick an unsatisfied clause randomly.*

3. *Randomly select a literal within that clause*
   (a) Penalize the LA corresponding to the literal variable.
   (b) Ask the penalized LA to assign a truth value to its variable.

4. *Pick a satisfied clause randomly.*

5. *Randomly select a literal within that clause*
   (a) If the literal evaluates to TRUE, reward the LA corresponding to the literal variable.
   (b) Ask the LA to assign a truth value to its variable.

6. *If all clauses are satisfied, stop. Otherwise, go to 2.* [5]

To evaluate the results, Granmo and Bouhmala solved benchmark sets containing SAT instances. The results were compared with the results obtained by using the Random Walk algorithm. The SAT instances that were solved in their paper range from a 125-literal random problem with 528 clauses to a 459-literal Blocks World problem with 4675 clauses. In all cases Granmo and Noureddine proved in their paper that solving SAT using finite Learning Automata combined with Random Walk algorithm drastically outperformed the latter alone. The harder the SAT instances were, the better their algorithm performed compared to Random Walk. Based on this conclusion, the Learning Automata combination with Random Walk has proved much more efficient in solving SAT than the latter alone.

2.6 Others

Clause weighting algorithms [15, 16] have been introduced to solve SAT problems. The idea is to associate weight values to the clauses and to increase the weights of all clauses that are unsatisfied as soon as a local minimum is discovered.

Other algorithms [25, 26, 27] use history based literal selection strategies to solve SAT in order to keep track of truth value assignments (similar to the method discussed in chapter 2.4).

3 Research Approach

The proposed solutions of the problem are presented in this chapter. In the following sections, the Multilevel paradigm is explained in detail as well as the combination of the latter with Tabu Search and finite Learning Automata. The implementations of the new algorithms are then discussed in detail.

3.1 Multilevel Paradigm

Associate professor N. Bouhmala - our supervisor - from Vestfold University College in Norway along with other authors have introduced in his 1995 PhD paper a new multilevel technique for solving problems. It is Bouhmala's idea to use this technique, for the first time, in the SAT context to see if efficiency will be increased or not.

The Multilevel clustering technique simplifies the computation of SAT instances by dividing the number of literals in several levels - literals get clustered together. The Multilevel paradigm consists of three phases: clustering, initial solution and refinement.

As an example, consider a SAT instance with 20 literals. The literals are initialized in the initial level, in the first level the literals get clustered together (two and two) and this continues. So in the first level there would be 10 literals, in the second 5 literals, in the third 3 literals and so on. The amount of levels created is optional, however a relatively big amount is recommended. Figure 5 illustrates the clustering process of a SAT instance with 20 literals.

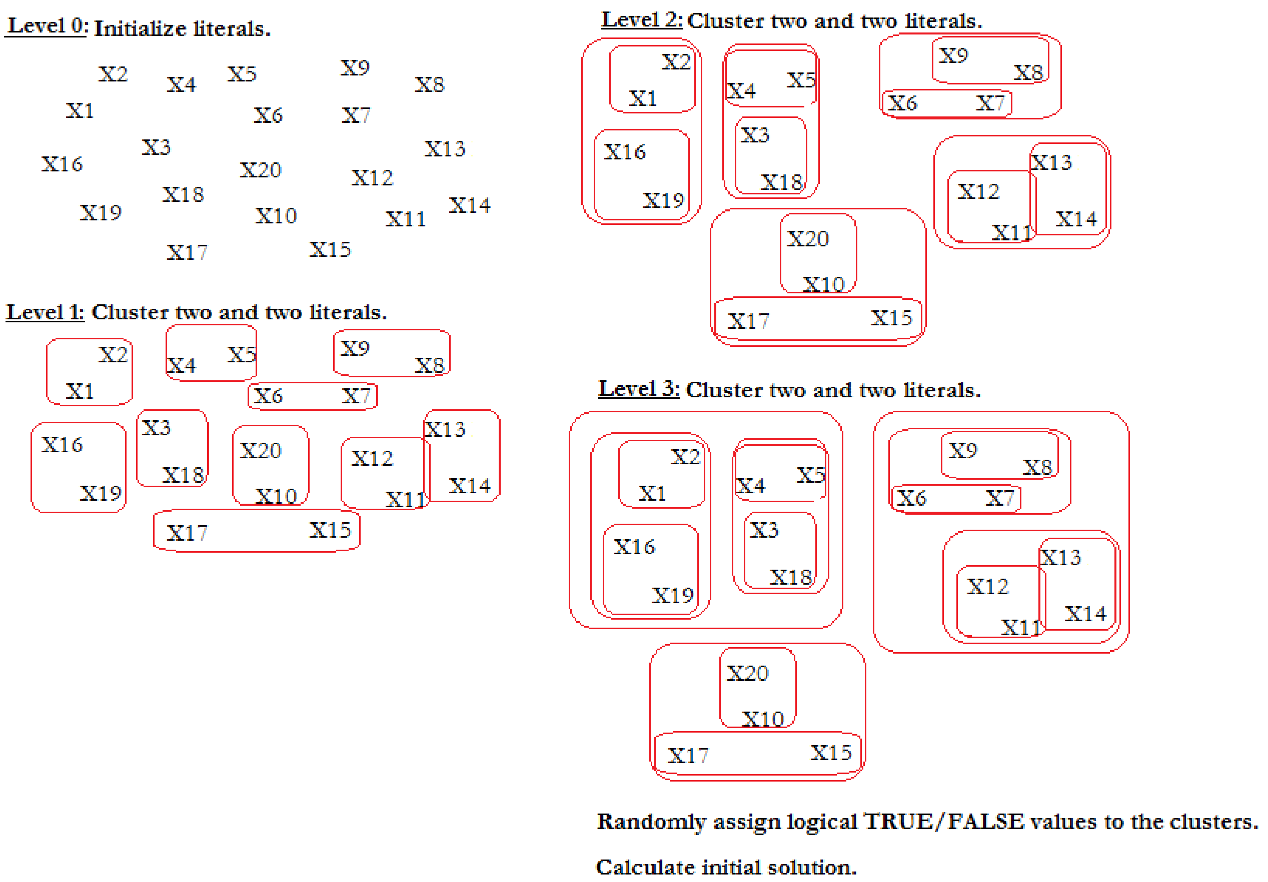

Figure 5: The Multilevel clustering technique used on a SAT instance with 20 literals.

Once the clustering phase is complete, the clusters in the final level are assigned logical TRUE/FALSE values and an initial solution is calculated. The solution found is then extended to provide a solution for the level above and then refined using a metaheuristic algorithm. This is done on all levels until a solution to SAT is found. If the initial level is reached and a solution is not found, then the SAT instance could not be solved.

The following steps summarize the Multilevel process:

1. Initialize literals.

2. Randomly cluster two literals together, or cluster two neighbouring literals in each level.

3. Do step 2 until the wished amount of clusters is reached.

4. Randomly assign logical TRUE/FALSE values to the clusters in the final level.

5. Compute initial solution.

6. Start refinement phase using metaheuristic algorithm.

The core strength of the Multilevel technique is that during the refinement phase, the algorithm used will compute clusters of literals instead of single literals at a time. This allows the algorithm to view clusters of literals as a single entity, making the search space guided and restricted to only those literals within the clusters. This offers a better sampling of the solution space compared to single level computation.

3.2 Combining Multilevel with Tabu Search

During the refinement phase of the Multilevel technique, an algorithm such as Tabu Search can be used. As discussed earlier in chapter 4, Tabu Search maintains a tabu list which contains flipped literals. Instead of containing flipped literals, it will in this case contain flipped clusters. The tabu list is updated each time a cluster of literals is flipped. Using the list, the algorithm prevents the clusters in the list from being flipped again during the computation.

3.3 Combining Multilevel with Finite Learning Automata

Similarly, during the refinement phase of the Multilevel technique, a technique such as finite Learning Automata can be used. Since this is a technique, it must be combined with an algorithm. For simplicity, it can be combined with the Tabu Search algorithm. As discussed earlier in chapter 4, finite Learning Automata assigns each literal a dedicated learning automaton. In this case, it will assign each cluster a learning automaton. It will then handle the cluster as a single entity, affecting all literals within the cluster. The state of the learning automaton can either be positive or negative. In the latter case, the action FALSE will be performed by the learning automaton. In the former, the action TRUE will be performed. The environment is the SAT instance, which can provide a reward or penalty response to the learning automaton depending on the automaton's action, as explained earlier in chapter 4.

3.4 Tabu Search Implementations

Several Tabu Search variants (five in total) were implemented in order to find the most efficient of the latter. In the following sections each variant is thoroughly examined.

3.4.1 Tabu Search Version 1.0

This is the basic version of Tabu Search. Each version (with the exception of the greedy version) is based on this one. The pseudo code below shows the procedure of Tabu Search version 1.0 (TS v.1.0).

```
Procedure Tabu Search version 1.0
Begin
initialize tabu list
randomly assign TRUE/FALSE values to the literals
current = evaluate initial solution
bestSoFar = current
while current < number_of_clauses do
    bestGain = -999
    gain = 0
   for i = 1 to number_of_clauses - current
       pick an unsatisfied clause i
       randomly pick a literal inside the clause i which is not visited
       mark literal visited
       flip literal
       gain = compute new_gain
       if literal is tabu then
          store the literal and its gain
       else if literal is not tabu then
          if gain == bestGain then
              pick a gain randomly
              bestGain = gain
              store the literal and its gain if not stored already
           else if gain > bestGain then
              bestGain = gain
              store the literal and its gain
        flip literal
   end-for
   pick literal with best gain
   if literal is tabu AND gain + current < bestSoFar then
     do not flip
   else
     flip literal
     update clauses, current, bestSoFar
   tabuBestUnsatisfied = find the tabu literal from tabu list which has the lowest number of unsatisfied clauses
     if literal is not tabu AND number_of_clauses - current < tabuBestUnsatisfied then
        make literal tabu with the value (number_of_clauses - current)
     else if literal is not tabu AND number_of_clauses - current >= tabuBestUnsatisfied then
        make literal tabu with the value tabuBestUnsatisfied
  decrease all other literals in tabu list with value bigger than 0 by 1
end-while
End
```

The idea of Tabu Search is to use a history based selection strategy where flipped literals along with the number of unsatisfied clauses are stored in a so called tabu list. The algorithm will run as long as there are unsatisfied clauses present or a maximum amount of flips is reached (this constraint can be substituted by a time limit). A loop will go through all unsatisfied clauses, picking an unsatisfied clause each time and randomly choosing an unhandled/unvisited literal from the clause. The unvisited literal is then flipped and marked visited and its truth value gain is computed. The literal is then checked if it is tabu or not. If it is tabu, the literal and its gain is stored. If it is not tabu, its gain is checked with the best gain so far. If they are equal, one of them is stored randomly. If the gain is bigger, the best gain so far is updated and the new gain is stored. Once the loop is finished, the literal with the best gain so far is picked. This literal is first checked if it is tabu, and if its gain has improved the total number of satisfied clauses. If that is not the case, the literal is not flipped. Otherwise the literal is flipped and the clauses are updated. The next process is to make this literal tabu - if it is not already tabu - and update the tabu list. In this version of Tabu Search, the tabu literal with the lowest number of unsatisfied clauses (that is not zero) is located from the tabu list. This tabu literal's number of unsatisfied clauses is then checked with the current number of unsatisfied clauses. If the latter is smaller than the former, then the flipped literal is made tabu with the current number of unsatisfied clauses. If opposite or if they are equal, then the flipped literal is made tabu with the lowest number of unsatisfied clauses from the tabu list. If the flipped literal is already tabu, then this process is ignored. Once this is finished, all tabu literals (with values bigger than zero) are decreased by one. The idea here is to make a literal tabu for a certain length of time. The algorithm is then terminated if all clauses are satisfied or if a maximum number of flips is reached (a time limit could also be used).

3.4.2 Tabu Search Version 2.0

This version of Tabu Search (TS v.2.0) is similar to the one in the previous section except that here if a literal is tabu it is not handled. Simply put, tabu literals are ignored during the loop. This has the consequence of increasing the number of tabu literals in the tabu list.

3.4.3 Tabu Search with Fixed Tabu List Lengths

This version of Tabu Search is similar to the one in section 3.4.1 except that here static lengths are used for the tabu list. It has been observed in earlier research [14] that using static lengths has a positive effect on increasing the number of satisfied clauses. In the following sections, we investigate this by setting various lengths.

3.4.3.1 Static Lengths

Suggested by associate professor Bouhmala, when making a literal tabu we set its value to 1 (in contrast to setting this value to the number of unsatisfied clauses). The process of making the literal tabu and updating the tabu list would then look as shown in the pseudo code below.

```
if literal is not tabu then
    make literal tabu with the value 1
decrease all other literals in tabu list with value bigger than 0 by 1
```

The lengths from ten to thirty-five were also suggested by associate professor Bouhmala, and were tested in the following sequence; ten, fifteen, twenty, twenty-five, thirty, thirty-five.

3.4.3.2 Static Optimal Lengths

As mentioned earlier in chapter 2.4, B. Mazure, L. Saïs and E. Gregoire showed in their paper a certain optimal length that can be used when setting the tabu list length. This value seemed to be linear with the amount of literals present. Using this length, the pseudo code of making a literal tabu and updating the tabu list would then look as shown below.

```
if literal is not tabu then
    make literal tabu with the value (0.01875×number_of_literals+2.8125)
decrease all other literals in tabu list with value bigger than 0 by 1
```

3.4.4 Greedy Tabu Search

This version of Tabu Search is based on a greedy approach introduced by associate professor Bouhmala. This version is seen as a possible extension for future work on Tabu Search. The pseudo code below shows the procedure of Greedy Tabu Search (GTS).

```
Procedure Greedy Tabu Search
Begin
initialize tabu list
randomly assign TRUE/FALSE values to the literals
current = evaluate initial solution
bestSoFar = current
while current < number_of_clauses do
    gain = 0
    for i = 1 to number_of_clauses - current
        pick an unsatisfied clause i
        randomly pick a literal inside the clause i which is not visited
        mark literal visited
        flip literal
        gain = compute new_gain
        if literal is tabu then
            if gain <= 0 then
                flip literal
            else
                put the literal and its gain in a sequence list
        else if literal is not tabu then
            put the literal and its gain in a sequence list
    end-for
    find the sequence that best increases the gain from the sequence list
    flip back the literals after this sequence because they decrease the gain
    update clauses, current, bestSoFar
    tabuBestUnsatisfied = find the tabu literal from tabu list which has the lowest number of unsatisfied clauses
    for i=1 to literals_in_sequence_list
      if literal i is not tabu AND number_of_clauses - current < tabuBestUnsatisfied then
        make literal i tabu with the value (number_of_clauses - current)
      else if literal i is not tabu AND number_of_clauses - current >= tabuBestUnsatisfied then
        make literal i tabu with the value tabuBestUnsatisfied
    decrease all other literals in tabu list (which are not in the sequence list) with value bigger than 0 by 1
    clear sequence list
end-while End
```

This greedy approach for Tabu Search maintains a sequence list that contains literals and their gains. The main idea of this approach is to flip literals during the loop and put them in the sequence list, if a flipped literal is tabu and the gain gives no improvement, then this literal is flipped back. Otherwise, literals are flipped consecutively without being flipped back. Once the loop is finished, the sequence that gives the best gain increase is chosen from the sequence list. The literals after this sequence are flipped back because they decrease the gain. The flipped literals are then all made tabu using the same process in Tabu Search version 1.0 (discussed in section 3.4.1) and the tabu list is updated. After this is done, the sequence list is cleared and the same process is repeated for the next iteration of the while-loop. The algorithm will terminate if all clauses are satisfied or if a maximum number of flips is reached (a time limit could also be used).

3.5 Selecting the Best Tabu Search Implementation

In order to decide which Tabu Search implementation is the most efficient, the algorithms were tested on the following random benchmark problems from SATLIB [21]; 600 literals and 2550 clauses (f600), 1000 literals and 4250 clauses (f1000) and 2000 literals and 8500 clauses (f2000). Each algorithm ran each problem with a 600 seconds timeout, 10 times in order to make a mean estimate. Tables 1, 2 and 3 show the results of solving each problem.

Algorithm	**Problem**	**Mean solved (%)**	**Mean time (seconds)**
TS v.1.0	f600	99.4 %	605.4 s.
TS v.2.0	f600	99.7 %	606 s.
TS with fixed lengths	f600	99.4 %	617.2 s.
GTS	f600	97.1 %	604.6 s.

Table 1: Tabu Search implementations solving a 600 literals and 2550 clauses (f600) random SATLIB benchmark problem. The percentage solved is the number of satisfied clauses. Time limit set to 600 seconds.

Algorithm	**Problem**	**Mean solved (%)**	**Mean time (seconds)**
TS v.1.0	f1000	99 %	607 s.
TS v.2.0	f1000	99.1 %	606.6 s.
TS with fixed lengths	f1000	99 %	606.4 s.
GTS	f1000	97.3 %	606.4 s.

Table 2: Tabu Search implementations solving a 1000 literals and 4250 clauses (f1000) random SATLIB benchmark problem. The percentage solved is the number of satisfied clauses. Time limit set to 600 seconds.

Algorithm	**Problem**	**Mean solved (%)**	**Mean time (seconds)**
TS v.1.0	f2000	93.7 %	615.3 s.
TS v.2.0	f2000	93.4 %	617.2 s.
TS with fixed lengths	f2000	93.4 %	617.2 s.
GTS	f2000	92.7 %	617.5 s.

Table 3: Tabu Search implementations solving a 2000 literals and 8500 clauses (f2000) random SATLIB benchmark problem. The percentage solved is the number of satisfied clauses. Time limit set to 600 seconds.

Tables 4, 5 and 6 show the mean solved, variance and standard deviation.

Algorithm	Problem	Mean solved (%)	Variance	Standard deviation
TS v.1.0	f600	99.4 %	0.5	0.71
TS v.2.0	f600	99.7 %	0.4	0.63
TS with fixed lengths	f600	99.4 %	0.4	0.63
GTS	f600	97.1 %	8.2	2.86

Table 4: Tabu Search implementations solving a 600 literals and 2550 clauses (f600) random SATLIB benchmark problem. The mean solved, variance and standard deviation are shown.

Algorithm	Problem	Mean solved (%)	Variance	Standard deviation
TS v.1.0	f1000	99 %	0.5	0.71
TS v.2.0	f1000	99.1 %	0.3	0.55
TS with fixed lengths	f1000	99 %	0.3	0.55
GTS	f1000	97.3 %	80.2	8.96

Table 5: Tabu Search implementations solving a 1000 literals and 4250 clauses (f1000) random SATLIB benchmark problem. The mean solved, variance and standard deviation are shown.

Algorithm	Problem	Mean solved (%)	Variance	Standard deviation
TS v.1.0	f2000	93.7 %	0.2	0.45
TS v.2.0	f2000	93.4 %	0.1	0.32
TS with fixed lengths	f2000	93.4 %	0.1	0.32
GTS	f2000	92.7 %	8.1	2.85

Table 6: Tabu Search implementations solving a 2000 literals and 8500 clauses (f2000) random SATLIB benchmark problem. The mean solved, variance and standard deviation are shown.

In addition to these problems that were tested, seven other random problems (specifically f100, f125, f150, f175, f200, f225 and f250) from SATLIB benchmarks were tested and all algorithms gave high success rates ranging from 97.8 % to 100 %. Tabu Search version 2.0 was the only version to have solved six of these problems 100 %, and the seventh 99.3 %. Based on these results and the results shown in tables 1, 2 and 3, Tabu Search version 2.0 proved to be the best overall algorithm. It was therefore selected to be combined with the Multilevel paradigm (and later with Learning Automata).

Tables 4, 5 and 6 further illustrate the variance and standard deviation of each algorithm solving f600, f1000 and f2000. As can be seen, the variance and standard deviation are relatively low in almost all cases (with the exception of the GTS algorithm). This indicates that the algorithms are overall stable and the results are not widely spread around the mean. GTS seems to be the only algorithm that contradicts this, as it in all cases gave a rather high variance and standard deviation.

3.6 Learning Automata with Tabu Search Implementation

The implementation of Learning Automata with Tabu Search (LATS) is based on the algorithm discussed in section 2.5. The idea here is to integrate the Learning Automata implementation for SAT into Tabu Search, the pseudo code below shows the procedure of this.

```
Procedure Learning Automata with Tabu Search
Begin
initialize tabu list
for i=1 to number_of_literals
   randomly set the state of literal i to -1 or 1
   if state == -1 then
      set literal i to FALSE
   else
      set literal i to TRUE
end-for
current = evaluate initial solution
bestSoFar = current
while current < number_of_clauses do
     /*Learning Automata start*/
     randomly pick an unsatisfied clause
     randomly pick a literal or its negation from inside the clause
     if literal was picked AND state < (number_of_clauses – current) then
        increase the state of the literal by 1
        if state == 0 then
            flip literal
            update clauses, current, bestSoFar
     else if negated literal was picked AND state > -(number_of_clauses - current) then
        decrease the state of the negated literal by 1
        if state == -1 then
            flip negated literal
            update clauses, current, bestSoFar
     randomly pick a satisfied clause
     randomly pick a literal or its negation from inside the clause
    if literal was picked AND state >= 0 AND state < (number_of_clauses – current) then
            increase the state of the literal by 1
    else if negated literal was picked AND state < 0 AND state > -(number_of_clauses - current) then
            decrease the state of the negated literal by 1
    /*Tabu Search start*/
    bestGain = -999
    gain = 0
    for i = 1 to number_of_clauses - current
        pick an unsatisfied clause i
        randomly pick a literal inside the clause i which is not visited
        mark literal visited
        flip literal
        gain = compute new_gain
        if literal is not tabu then
           if gain == bestGain then
              pick a gain randomly
              bestGain = gain
              store the literal and its gain if not stored already
           else if gain > bestGain then
              bestGain = gain
              store the literal and its gain
```

```
        flip literal
     end-for
     pick literal with best gain
     if literal is tabu AND gain + current < bestSoFar then
        do not flip
    else
        flip literal
        update clauses, current, bestSoFar
         tabuBestUnsatisfied = find the tabu literal from tabu list which has the lowest number of unsatisfied
clauses
        if literal is not tabu AND number_of_clauses - current < tabuBestUnsatisfied then
            make literal tabu with the value (number_of_clauses - current)
        else if literal is not tabu AND number_of_clauses - current >= tabuBestUnsatisfied then
            make literal tabu with the value tabuBestUnsatisfied
    decrease all other literals in tabu list with value bigger than 0 by 1
end-while
End
```

The idea of Learning Automata with Tabu Search is to use the techniques of the latter and former together. As discussed in section 2.5, Learning Automata with a Random Walk approach provided good results when solving SAT instances. Therefore, combining Learning Automata (using a Random Walk approach) with Tabu Search should in theory provide better results than using the latter alone.

In Learning Automata, each literal has an automaton resulting in a team of automata. Each automaton starts randomly with a certain state value; -1 or 1. Literals with negative state values are assigned FALSE values, and literals with positive state values are assigned TRUE values (as illustrated in figure 6). The algorithm randomly picks an unsatisfied clause and a literal or its negation from inside the clause. The state value of the literal or its negation is strengthened by either increasing it (if it is positive) or decreasing it (if it is negative). If the state value of the literal or its negation changes from negative to positive - or vice versa - then it is flipped. The minimum state value is set to minus the number of unsatisfied clauses and the maximum state value is set to the number of unsatisfied clauses (we set these limitations in order to have a finite amount of state values). The algorithm then randomly picks a satisfied clause and a literal or its negation from inside the clause, this literal or its negation is then strengthened (rewarded) if its truth assignment contributes to the satisfaction of the clause. Its state value is increased (if it is positive) or decreased (if it is negative). Eventually, literals found in unsatisfied clauses are penalized and frequently flipped while literals found in satisfied clauses (which with their truth assignments contribute to the satisfaction of the clauses) are rewarded. Once the Learning Automata process is complete, Tabu Search - which we have covered in sections 3.4.1 and 3.4.2 - starts its process. The algorithm is then terminated if all clauses are satisfied or if a maximum number of flips is reached (a time limit could also be used).

3.7 Multilevel Paradigm Implementation

As mentioned in section 3.1, the Multilevel paradigm consists of three phases; clustering, initial solution and refinement. A metaheuristic algorithm is used in the last phase of the paradigm, in our case, Tabu Search and Learning Automata.

3.7.1 Clustering, Evaluation of Initial Solution and Refinement Phases

Clustering, evaluation of initial solution and refinement are the three phases of the Multilevel paradigm. The pseudo code below shows how the first two phases work. The third phase will be explained in the next sections.

```
Procedure Multilevel Paradigm
Begin
level = 0
clusterCollection = initialize literals
while clusterCollectionSize != size_limit do
    randomly cluster two literals or clusters together and put them in clusterCollection
    update clusterCollection, clusterCollectionSize
    if reached_end_of_clusterCollection then
        increase level by 1
end-while
randomly assign TRUE/FALSE values to the clusters in clusterCollection (final level)
current = evaluate initial solution
start refinement phase
End
```

The Multilevel paradigm works as explained in section 3.1. A size limit on the number of clusters at the final level decides how far the clustering process will go. Setting this value to 10 % of the total number of literals is a good measurement. Once the clustering process is complete, the clusters at the final level are randomly assigned TRUE/FALSE values. The initial solution is then computed and the refinement phase is ready to start.

3.8 Multilevel Tabu Search Implementation

Tabu Search is slightly modified in order to work properly in the refinement phase of the Multilevel paradigm. The modifications to the algorithm will be explained in the next section.

3.8.1 Refinement Phase Using Tabu Search

The implementation of Tabu Search is slightly modified in order to handle clusters of literals. The pseudo code below shows the Tabu Search refinement procedure.

```
Procedure Multilevel Tabu Search (refinement phase)
Begin
bestSoFar = current
while current < number_of_clauses do
    initialize tabu list for level
    bestGain = -999
    gain = 0
    if level != 0 then
        for i = 1 to number_of_clusters_in_level
            randomly pick a cluster i
            mark cluster i visited
            flip cluster i
            gain = compute new_gain
            if cluster i is not tabu then
                if gain == bestGain then
                    pick a gain randomly
                    bestGain = gain
                else if gain > bestGain then
                    bestGain = gain
                store cluster i and its gain
            flip cluster i
        end-for
        decrease level by 1
        pick cluster with best gain
        if cluster is tabu AND gain + current < bestSoFar then
            do not flip
        else
            flip cluster
            update clauses, current, bestSoFar
  tabuBestUnsatisfied = find the tabu cluster from tabu list which has the lowest number of unsatisfied clauses
            if cluster is not tabu AND number_of_clauses - current < tabuBestUnsatisfied then
                make cluster tabu with the value (number_of_clauses - current)
            else if cluster is not tabu AND number_of_clauses - current >= tabuBestUnsatisfied then
                make cluster tabu with the value tabuBestUnsatisfied
            decrease all other clusters in tabu list with value bigger than 0 by 1
    else
        start procedure Tabu Search
end-while
End
```

The Multilevel variant of Tabu Search (MTS) works as the latter except that here clusters of literals instead of single literals are handled at a time; a loop runs through all the clusters in a level and handles each cluster. Once finished with a level, the best cluster is flipped and made tabu (if not already tabu). The tabu list is then updated. The algorithm will then proceed to the next level and repeat the process. Once the final level is reached, the Tabu Search procedure discussed in sections 3.4.1 and 3.4.2 will start running. The algorithm will terminate if all clauses are satisfied or if a maximum number of flips set for each level is reached (a time limit could also be used and a number of iterations per level).

3.9 Multilevel Learning Automata with Tabu Search Implementation

Learning Automata with Tabu Search is slightly modified in order to work properly in the refinement phase of the Multilevel paradigm. In addition, the evaluation of initial solution phase is changed to accommodate the clusters of literals in the final level. The pseudo code below shows the change.

```
for i=1 to clusterCollection
    randomly set the state of cluster i to -1 or 1
    if state == -1 then
        set cluster i to FALSE
    else
        set cluster i to TRUE
end-for
current = evaluate initial solution
```

In the Multilevel variant of Learning Automata with Tabu Search (MLATS), state values are set to clusters of literals instead of single literals. Once the final level is reached, a loop runs through all clusters in the final level and randomly sets state values to -1 or 1. The state values set to the clusters are propagated to the literals inside. If a cluster has state value -1, it is assigned a FALSE value. Similarly if it has a state value 1, it is assigned a TRUE value.

3.9.1 Refinement Phase Using Learning Automata with Tabu Search

The implementation of Learning Automata with Tabu Search is slightly modified in order to handle clusters of literals in the refinement phase of the Multilevel paradigm. The pseudo code below shows the Learning Automata with Tabu Search refinement procedure.

```
Procedure Multilevel Learning Automata with Tabu Search (refinement phase)
Begin
bestSoFar = current
while current < number_of_clauses do
    if level != 0 then
        /*Learning Automata start*/
         randomly pick a cluster from current level
         for i=1 to number_of_literals_in_cluster
             randomly pick literal i or its negation
             if literal i was picked then
                pick a clause that has literal i
             else if negated literal i was picked then
                pick a clause that has negated literal i
             if the clause is unsatisfied then
                if literal i was picked AND state < (number_of_clauses – current) then
                     increase the state of the literal i by 1
                     if state == 0 then
                        flip literal i
                        update clauses, current, bestSoFar
                else if negated literal i was picked AND state > -(number_of_clauses - current) then
```

```
                    decrease the state of the negated literal i by 1
                    if state == -1 then
                        flip negated literal i
                        update clauses, current, bestSoFar
                else if the clause is satisfied then
                   if literal i was picked AND state >= 0 AND state < (number_of_clauses – current) then
                       increase the state of the literal i by 1
                   else if negated literal i was picked AND state < 0 AND state > -(number_of_clauses - current)
then
                       decrease the state of the negated literal i by 1
         end-for
        /*Tabu Search start*/
        initialize tabu list for level
        bestGain = -999
        gain = 0
        for i = 1 to number_of_clusters_in_level
              randomly pick a cluster i
              mark cluster i visited
              flip cluster i
              gain = compute new_gain
              if cluster i is not tabu then
                 if gain == bestGain then
                      pick a gain randomly
                      bestGain = gain
                 else if gain > bestGain then
                      bestGain = gain
                 store cluster i and its gain
              flip cluster i
          end-for
          decrease level by 1
          pick cluster with best gain
          if cluster is tabu AND gain + current < bestSoFar then
              do not flip
          else
              flip cluster
              update clauses, current, bestSoFar
  tabuBestUnsatisfied = find the tabu cluster from tabu list which has the lowest number of unsatisfied clauses
             if cluster is not tabu AND number_of_clauses - current < tabuBestUnsatisfied then
                 make cluster tabu with the value (number_of_clauses - current)
            else if cluster is not tabu AND number_of_clauses - current >= tabuBestUnsatisfied then
                 make cluster tabu with the value tabuBestUnsatisfied
            decrease all other clusters in tabu list with value bigger than 0 by 1
    else
        start procedure Learning Automata with Tabu Search
end-while
End
```

The Multilevel variant of Learning Automata with Tabu Search (MLATS) works as its predecessor except that here clusters of literals instead of single literals are handled at a time. A cluster is randomly picked from a level and a loop runs through all literals inside the cluster, a literal or its negation is picked during this loop and a clause that has the literal or its negation. The clause is then handled as we previously discussed in section 3.6; if it is unsatisfied, the state value of the picked literal or its negation is strengthened by either increasing it (if it is positive) or decreasing it (if it is negative). If the state value of the literal or its negation changes from negative to positive - or vice versa - then it is flipped. The minimum state value is set to minus the number of unsatisfied clauses and the maximum state value is set to the number of unsatisfied clauses (we set these limitations in order to have a finite amount of state values). If the clause is satisfied however, the picked literal or its negation is then strengthened (rewarded) if its truth assignment contributes to the satisfaction of the clause. Its state value is increased (if it is positive) or decreased (if it is negative). As mentioned in section 3.6, eventually literals found in unsatisfied clauses are penalized and frequently flipped while literals found in satisfied clauses (which with their truth assignments contribute to the satisfaction of the clauses) are rewarded. Once the loop iterates through all clusters in the given level and performs this process, Multilevel Tabu Search - which we have covered in section 3.8.1 - starts its process. Once the final level is reached, the Learning Automata with Tabu Search procedure discussed in section 3.6 will start running. The algorithm will terminate if all clauses are satisfied or if a maximum number of flips set for each level is reached (a time limit could also be used and a number of iterations per level).

4 Experimental Results

Benchmarks from SATLIB (Random, Planning, SAT Competition Beijing, AIM, All Interval Series, Graph Colouring SW and Quasi Groups) [21] and Max SAT (Industry) [29] were tested by the algorithms. Each instance was tested 10 times, each with a maximum flip set to 10^6 or in the case where time was used, the time limit set to 900 seconds. The average of flips, time and satisfied clauses were computed. In the end of the chapter, the mean solved, variance and standard deviation of the tested instances are shown.

4.1 Tabu Search vs. Multilevel Tabu Search

The performances of Tabu Search and Multilevel Tabu Search are compared and the results of the algorithms are shown in the next sections.

4.1.1 SATLIB Benchmark Problems

4.1.1.1 Random

Figures 6, 7 and 8 illustrate the results of solving the following random problems; 600 literals and 2550 clauses (f600), 1000 literals and 4250 clauses (f1000) and 2000 literals and 8500 clauses (f2000).

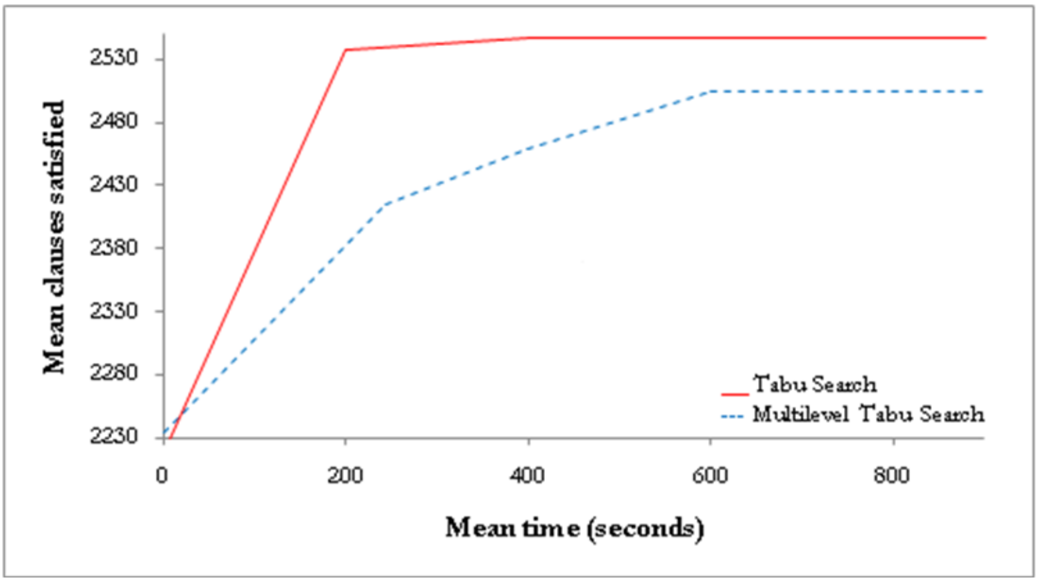

Figure 6: Tabu Search vs. Multilevel Tabu Search solving a 600 literals and 2550 clauses (f600) random problem. Along the horizontal axis the mean time is given and along the vertical axis the mean number of satisfied clauses.

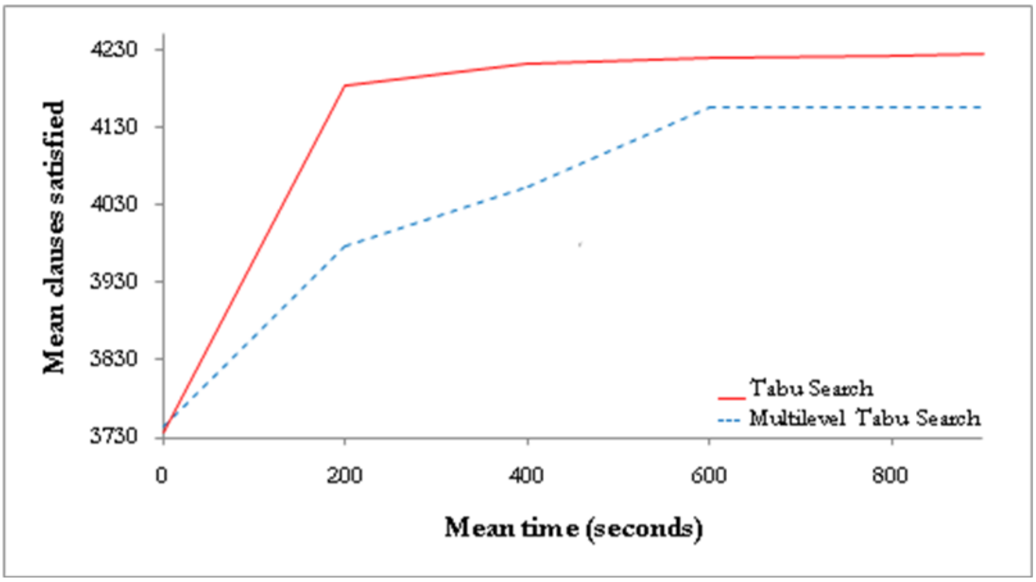

Figure 7: Tabu Search vs. Multilevel Tabu Search solving a 1000 literals and 4250 clauses (f1000) random problem. Along the horizontal axis the mean time is given and along the vertical axis the mean number of satisfied clauses.

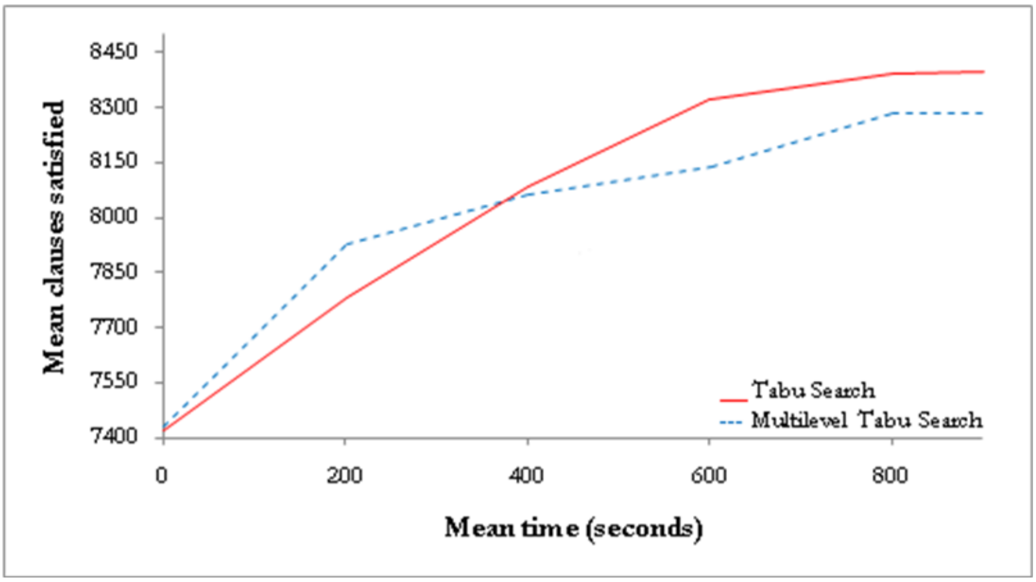

Figure 8: Tabu Search vs. Multilevel Tabu Search solving a 2000 literals and 8500 clauses (f2000) random problem. Along the horizontal axis the mean time is given and along the vertical axis the mean number of satisfied clauses.

Figure 6 illustrates that f600 is a small problem, and thus the Multilevel variant is not as effective as assumed. However, as can be seen from the graph, while Tabu Search starts to stagnate after around 400 seconds the Multilevel variant continues to converge until around 600 seconds from which it starts to stagnate as well. This occurs because the problem is small and the search space is restricted. As a result, it seemed that single space searching seemed most efficient for this problem.

f1000 is also a small problem and thus the Multilevel variant is also not as effective as assumed, looking at figure 7. As a result, it seemed that single space searching seemed most efficient for this problem.

f2000 is also a small problem. As observed in figure 8, the Multilevel variant's performance is approaching Tabu Search's as the problems grow bigger. Both algorithms steadily converge and Tabu Search manages to satisfy more clauses than the Multilevel variant.

4.1.1.2 Planning

Figures 9, 10, 11 and 12 illustrate the results of solving the following Blocks World problems; 116 literals and 953 clauses (medium), 459 literals and 7054 clauses (huge), 3016 literals and 50457 clauses (bw_large.c) and 6325 literals and 131973 clauses (bw_large.d).

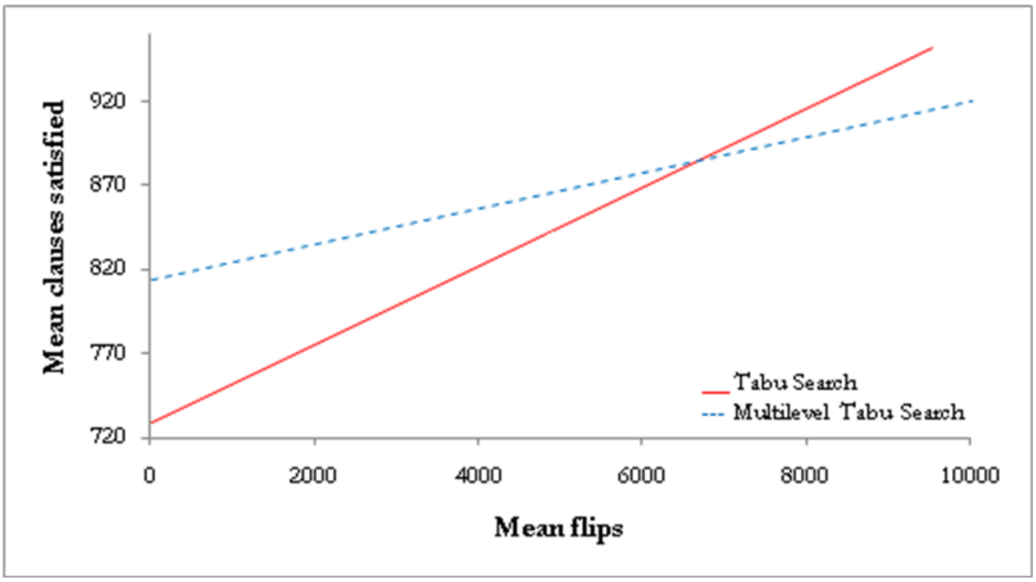

Figure 9: Tabu Search vs. Multilevel Tabu Search solving a Blocks World problem with 116 literals and 953 clauses problem (medium). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

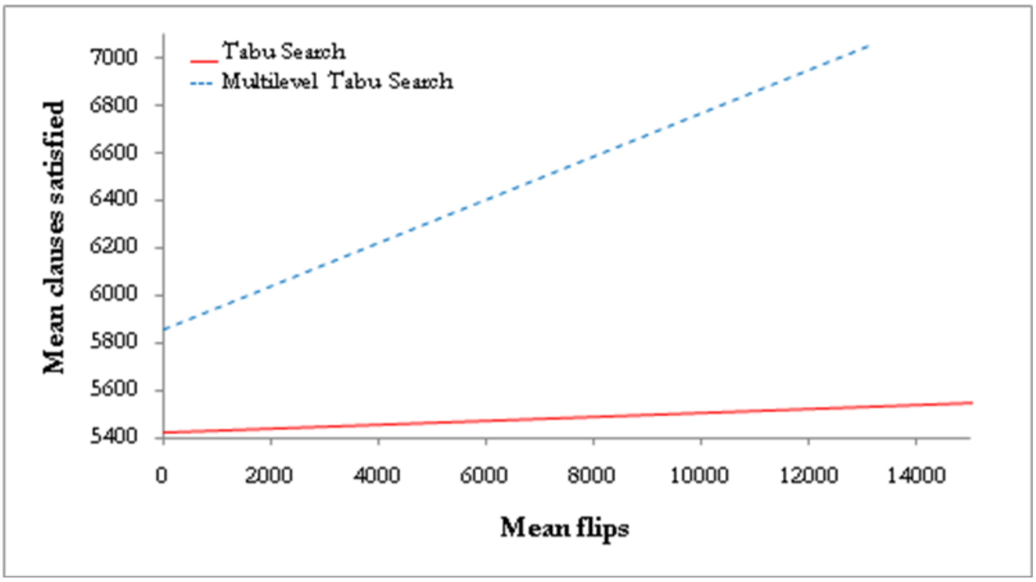

Figure 10: Tabu Search vs. Multilevel Tabu Search solving a Blocks World problem with 459 literals and 7054 clauses problem (huge). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

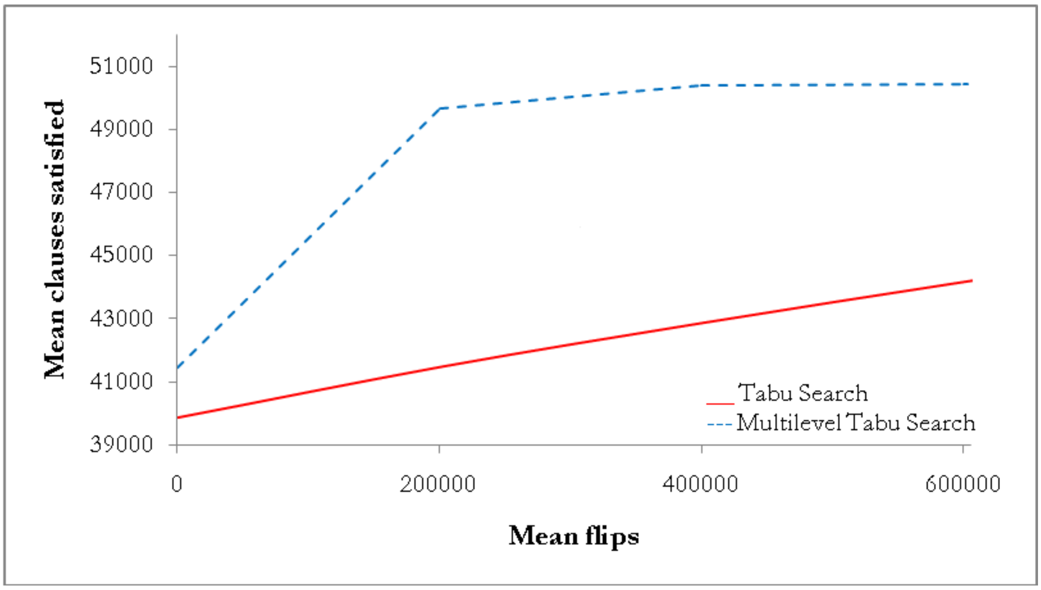

Figure 11: Tabu Search vs. Multilevel Tabu Search solving a Blocks World problem with 3016 literals and 50457 clauses problem (bw_large.c). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

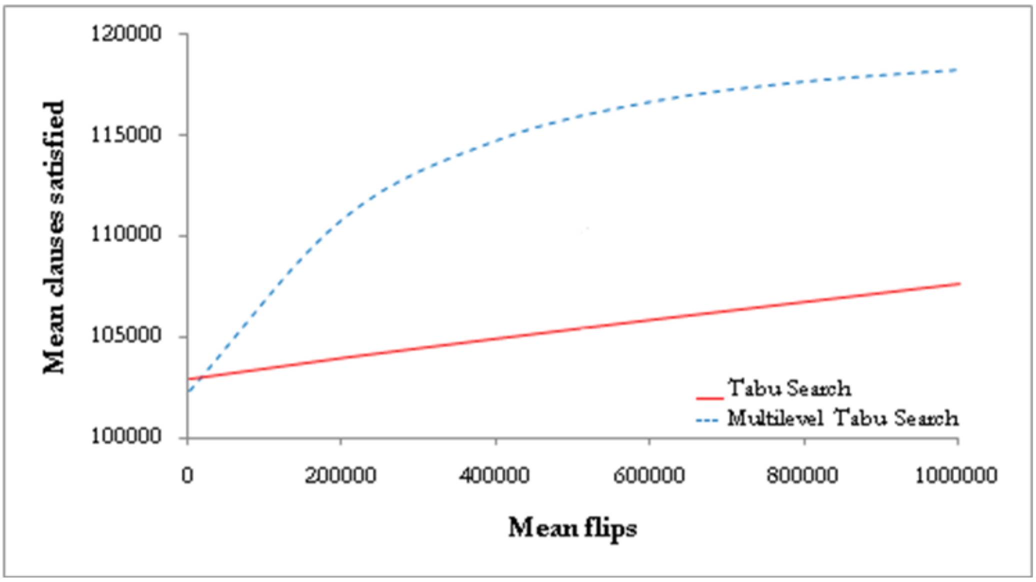

Figure 12: Tabu Search vs. Multilevel Tabu Search solving a Blocks World problem with 6325 literals and 131973 clauses (bw_large.d). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figure 9 illustrates that Tabu Search converges in a linear manner and manages to solve the problem at close to 10000 flips. The Multilevel variant shows a good convergence at the start, however it crosses with Tabu Search and manages to solve the problem at 10000 flips.

Figure 10 illustrates that the Multilevel variant managed to solve this problem after around 13000 flips, while Tabu Search started to stagnate from around 200000 flips and continued up to 10^6 without managing to solve the problem. As seen in the graph, the Multilevel variant is clearly superior to Tabu Search in terms of convergence efficiency.

As illustrated in figure 11, the Multilevel variant is superior to Tabu Search. It managed to solve this problem after around 600000 flips, while Tabu Search did not solve the problem after reaching 10^6 flips.

As illustrated in figure 12, the Multilevel variant is once again superior to Tabu Search. Multilevel excels in solving this problem due to its big size. While reaching the maximum amount of flips, the convergence rate of

Multilevel is much higher than Tabu Search. It can be clearly observed here that the bigger the SAT problem is, the more it is in favour of the Multilevel variant.

Figure 13 illustrate the results of solving a Logistics problem with 4713 literals and 21991 clauses (logistics.d).

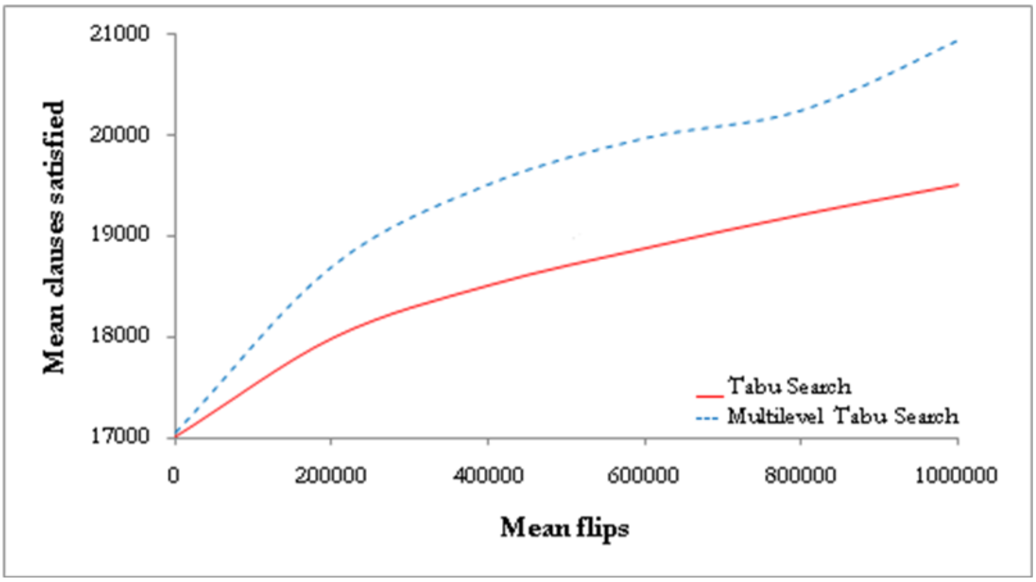

Figure 13: Tabu Search vs. Multilevel Tabu Search solving a Logistics problem with 4713 literals 21991 clauses (logistics.d). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figure 13 illustrates that Multilevel is clearly superior to Tabu Search in terms of convergence efficiency and quality.

4.1.1.3 SAT Competition Beijing

Figures 14 and 15 illustrate the results of solving the following Beijing problems; 21800 literals and 118607 clauses (ewddr2-10-by-5-1) and 22500 literals and 123329 clauses (ewddr2-10-by-5-8).

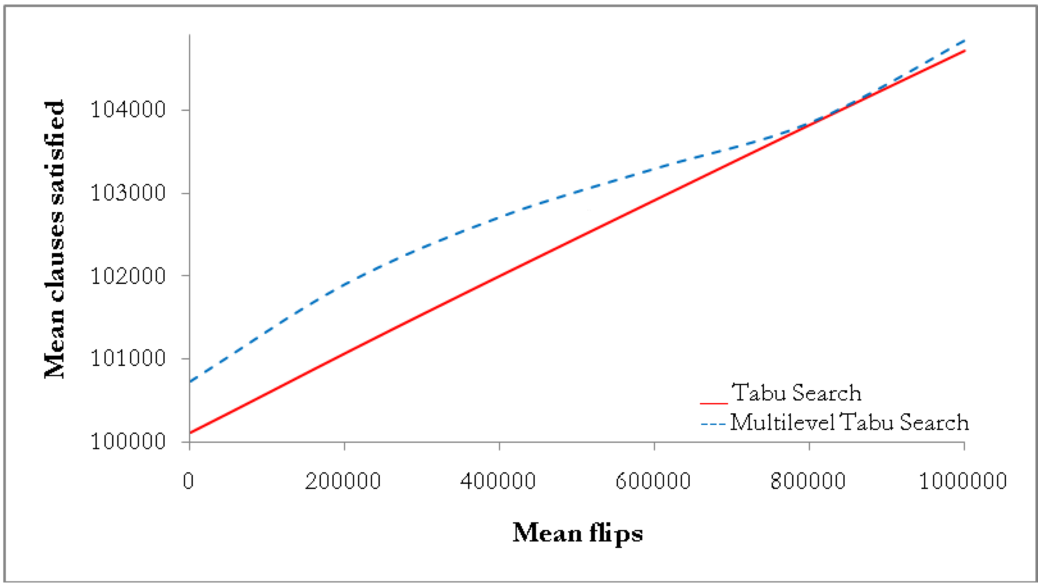

Figure 14: Tabu Search vs. Multilevel Tabu Search solving a Beijing problem with 21800 literals and 118607 clauses (ewddr2-10-by-5-1). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

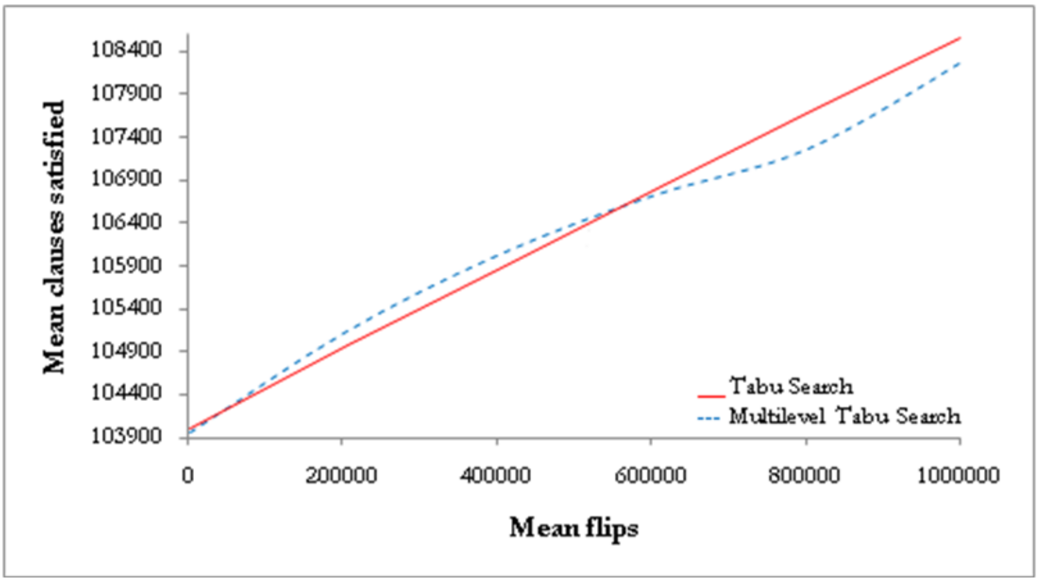

Figure 15: Tabu Search vs. Multilevel Tabu Search solving a Beijing problem with 22500 literals and 123329 clauses (ewddr2-10-by-5-8). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 14 and 15 illustrate that the Multilevel variant however close to Tabu Search, provides a slightly better convergence rate overall. Tabu Search provides a linear convergence while the Multilevel variant provides a more variable convergence here.

4.1.1.4 AIM

Figures 16 and 17 illustrate the results of solving the following AIM problems; 200 literals and 400 clauses (aim-200-2_0-yes1-4) and 200 literals and 680 clauses (aim-200-3_4-yes1-2).

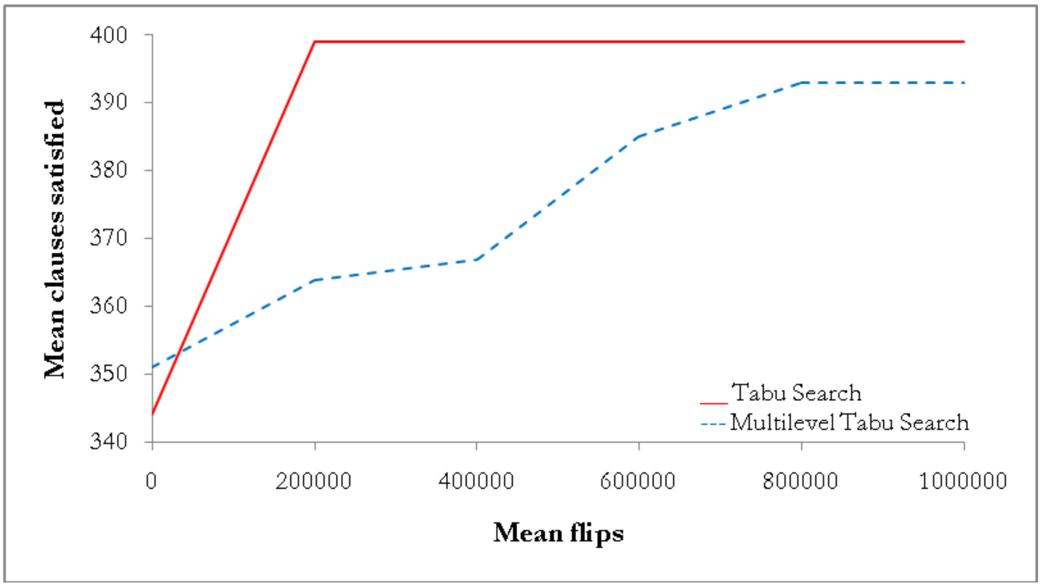

Figure 16: Tabu Search vs. Multilevel Tabu Search solving an AIM problem with 200 literals and 400 clauses (aim-200-2_0-yes1-4). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

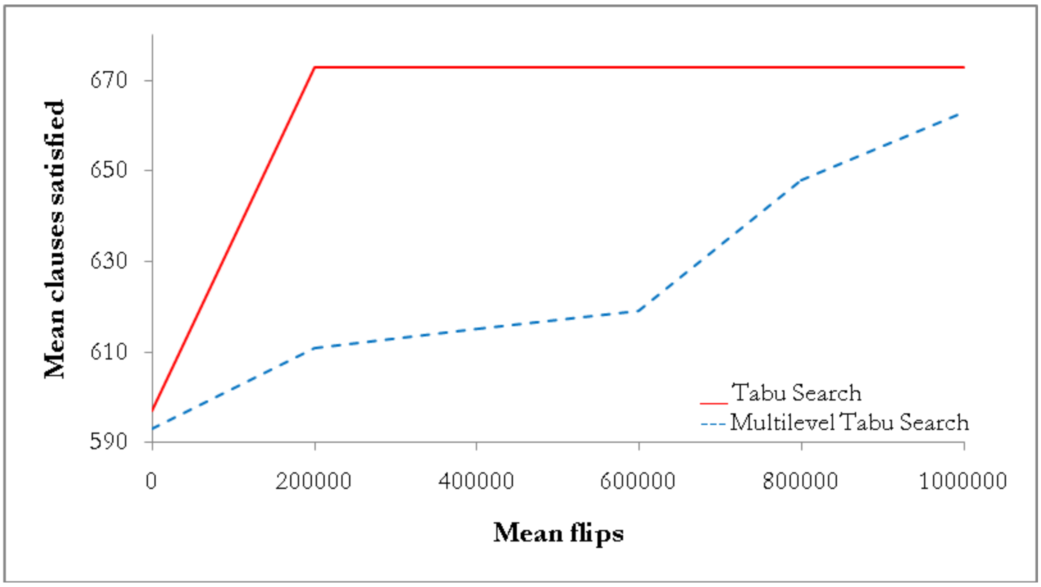

Figure 17: Tabu Search vs. Multilevel Tabu Search solving an AIM problem with 200 literals and 680 clauses (aim-200-3_4-yes1-2). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 16 and 17 illustrate that Tabu Search clearly outperforms the Multilevel variant in terms of convergence. However, the former shows an early stagnation which is not observed in the latter. The apparent convergence favour to Tabu Search in both cases confirms that the Multilevel variant does not work well in relatively small problems, excelling instead in rather large ones.

4.1.1.5 All Interval Series (AIS)

Figures 18 and 19 illustrate the results of solving the following All Interval Series (AIS) problems; 181 literals and 3151 clauses (ais10) and 265 literals and 5666 clauses (ais12).

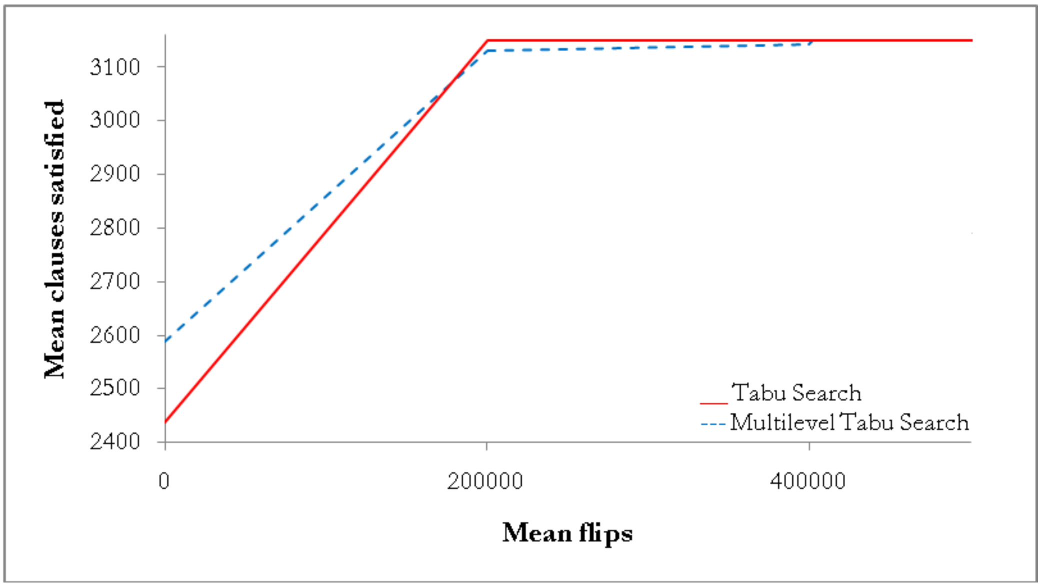

Figure 18: Tabu Search vs. Multilevel Tabu Search solving an AIS problem with 181 literals and 3151 clauses (ais10). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

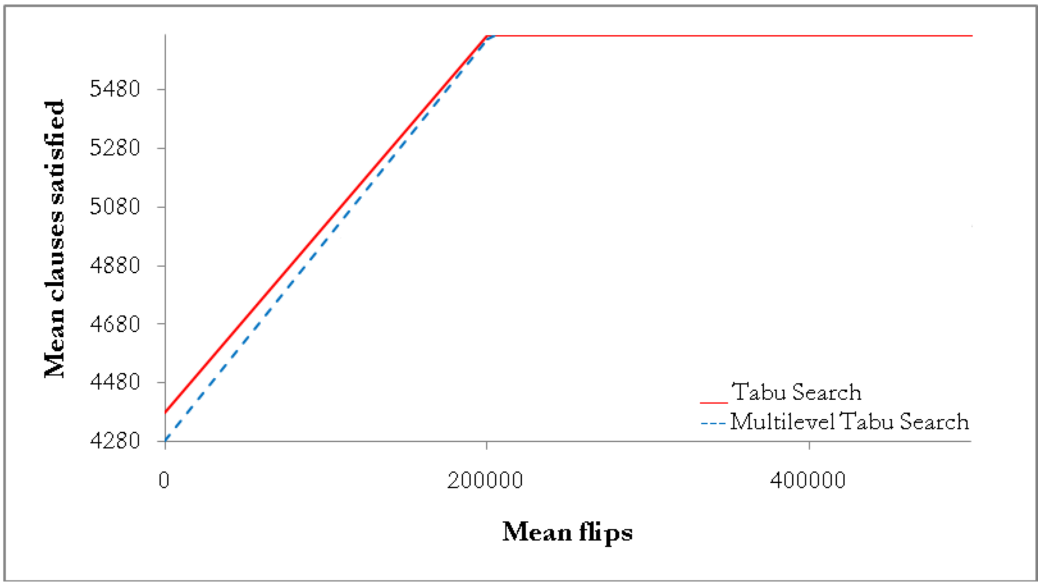

Figure 19: Tabu Search vs. Multilevel Tabu Search solving an AIS problem with 265 literals and 5666 clauses (ais12). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figure 18 illustrates a slight convergence favour to Multilevel until it starts to stagnate at around 200000 flips and continues until it manages to solve the problem at around 400000 flips. Tabu Search starts to stagnate at the same time and continues on without managing to solve the problem.

Figure 19 also illustrates a slight convergence favour, however in this case to Tabu Search. While Multilevel manages to solve the problem at around 200000 flips, Tabu Search starts to stagnate at this point and continues on without managing to solve the problem.

4.1.1.6 Graph Colouring SW

Figures 20 and 21 illustrate the results of solving the following Graph Colouring SW problems; 500 literals and 3100 clauses (sw100-98) and 500 literals and 3100 clauses (sw100-99).

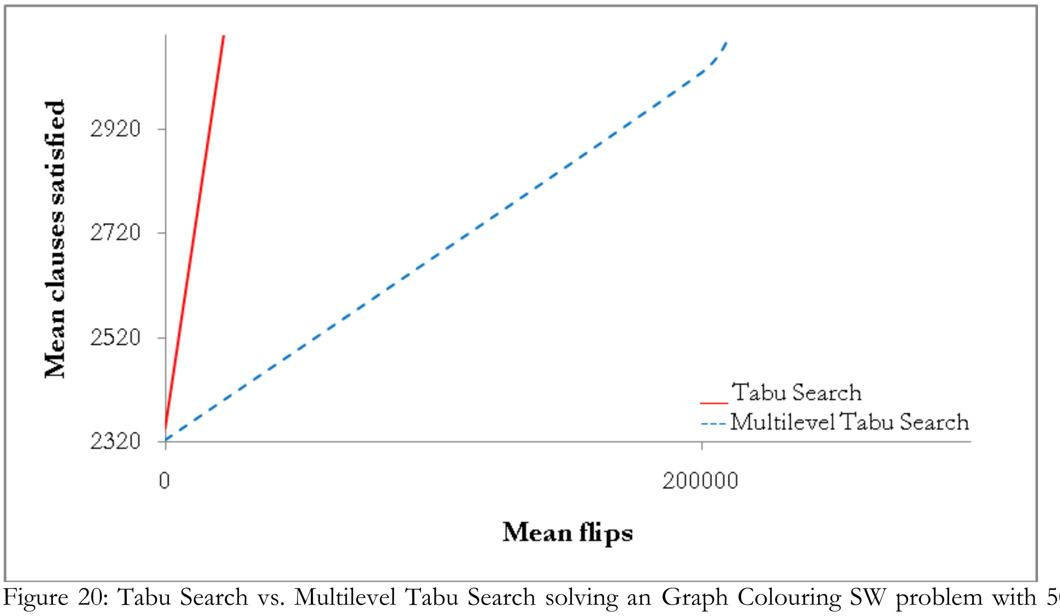

Figure 20: Tabu Search vs. Multilevel Tabu Search solving an Graph Colouring SW problem with 500 literals and 3100 clauses (sw100-98). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

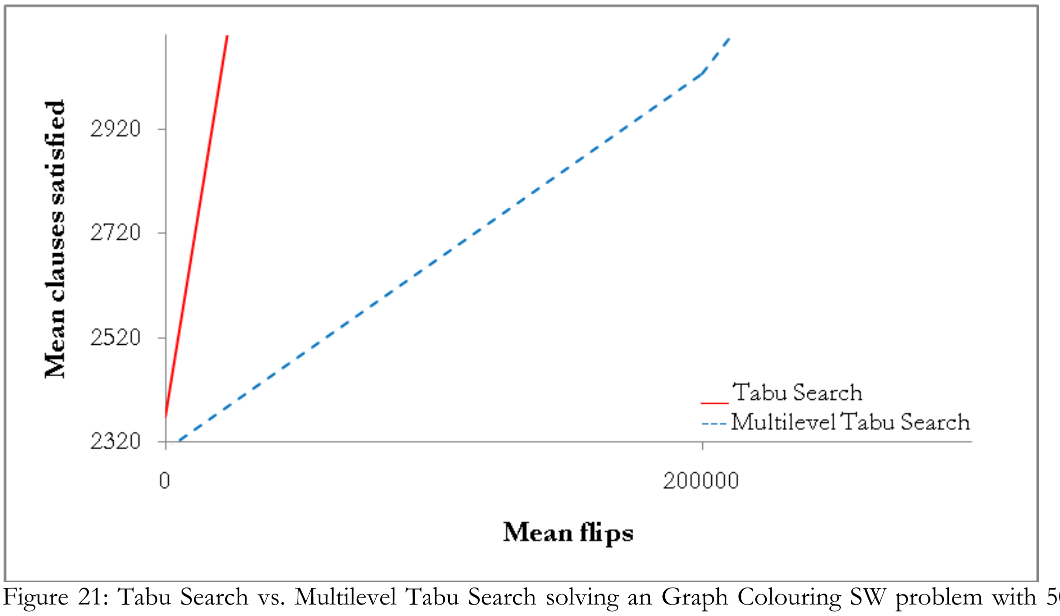

Figure 21: Tabu Search vs. Multilevel Tabu Search solving an Graph Colouring SW problem with 500 literals and 3100 clauses (sw100-99). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 20 and 21 illustrate that Tabu Search clearly beats Multilevel in terms of convergence, solving both problems in well under 200000 flips. Multilevel manages to solve both problems slightly above 200000 flips. The results of both algorithms are quite similar due to their respective sizes.

4.1.1.7 Quasi Groups

Figures 22 and 23 illustrate the results of solving the following Quasi Groups problems; 1331 literals and 49204 clauses (qg6-11) and 1728 literals and 69931 clauses (qg6-12).

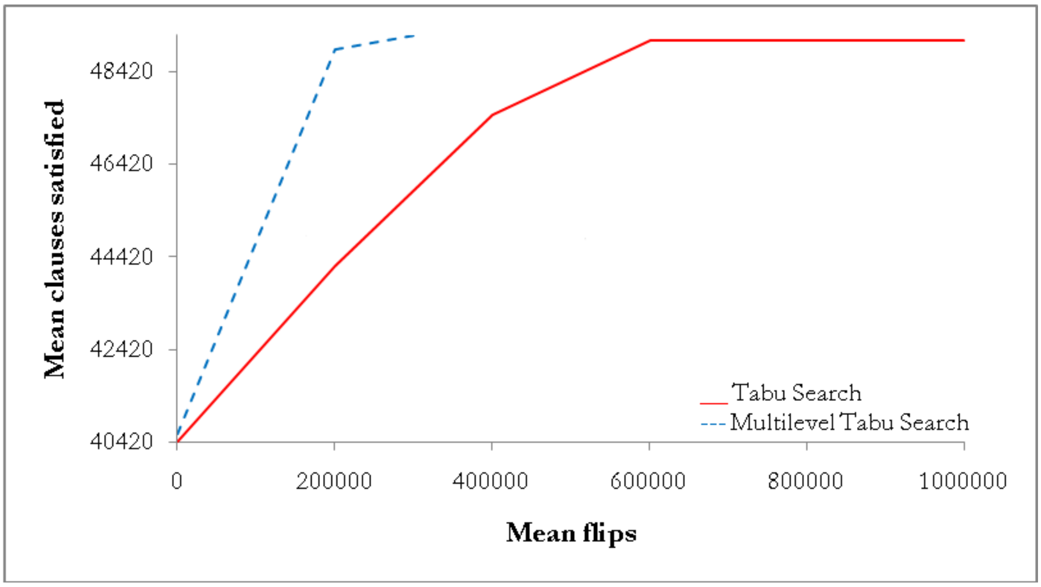

Figure 22: Tabu Search vs. Multilevel Tabu Search solving a Quasi Groups problem with 1331 literals and 49204 clauses (qg6-11). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

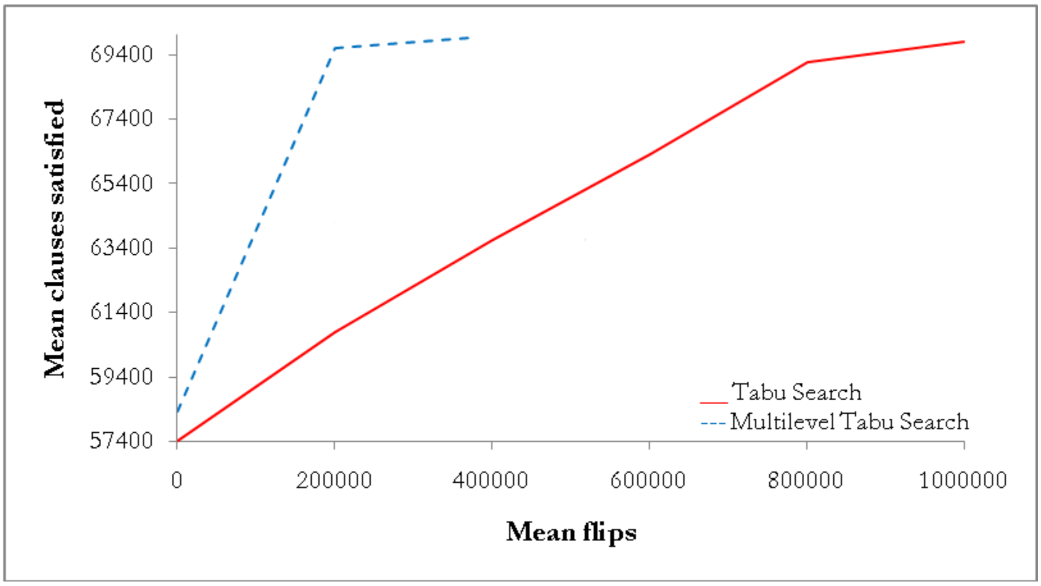

Figure 23: Tabu Search vs. Multilevel Tabu Search solving a Quasi Groups problem with 1728 literals and 69931 clauses (qg6-12). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 22 and 23 illustrate that the Multilevel variant clearly outperforms Tabu Search, both in convergence rate and quality. In figure 22 it is seen that Tabu Search starts to stagnate at around 600000 and continues on without managing to solve the problem, while Multilevel manages to solve the problem in well under 400000 flips. Similarly, figure 23 shows that Tabu Search does not manage to solve the problem while Multilevel does so at around 400000 flips.

4.1.2 Max SAT Problems

4.1.2.1 Industry

Figures 24, 25, 26, 27 and 28 illustrate the results of solving the following Max SAT (Industry) problems; 5484 literals and 13894 clauses (mot_comb2._red-gate-0.dimacs.seq.filtered), 11265 literals and 29520 clauses (mot_comb3._red-gate-0.dimacs.seq.filtered), 44079 literals and 117720 clauses (c6_DD_s3_f1_e1_v1-bug-onevec-gate-0.dimacs.seq.filtered), 84525 literals and 236942 clauses (c2_DD_s3_f1_e2_v1-bug-onevec-gate-0.dimacs.seq.filtered) and 200944 literals and 540984 clauses (c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered).

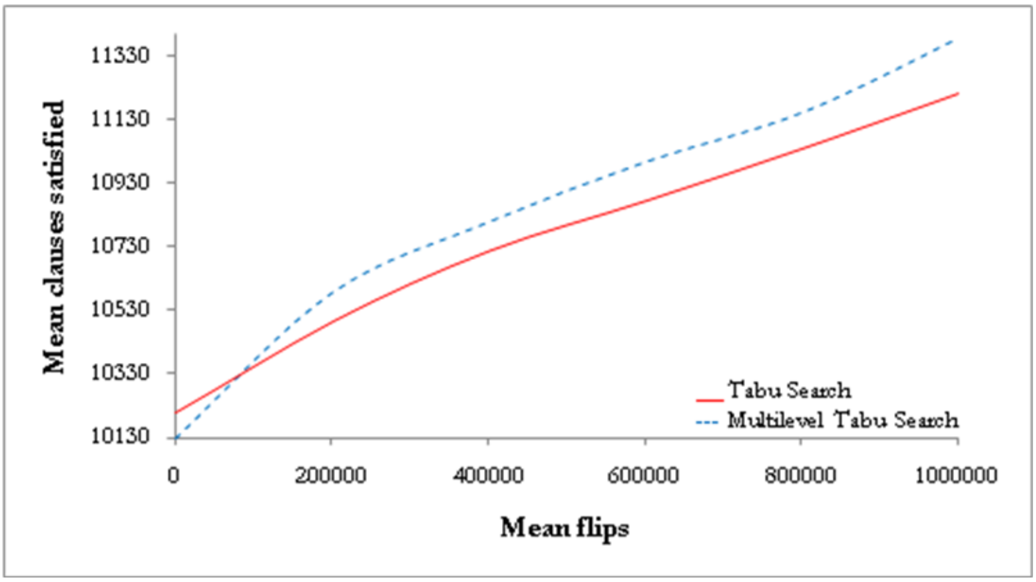

Figure 24: Tabu Search vs. Multilevel Tabu Search solving a Max SAT problem with 5484 literals and 13894 clauses (mot_comb2._red-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

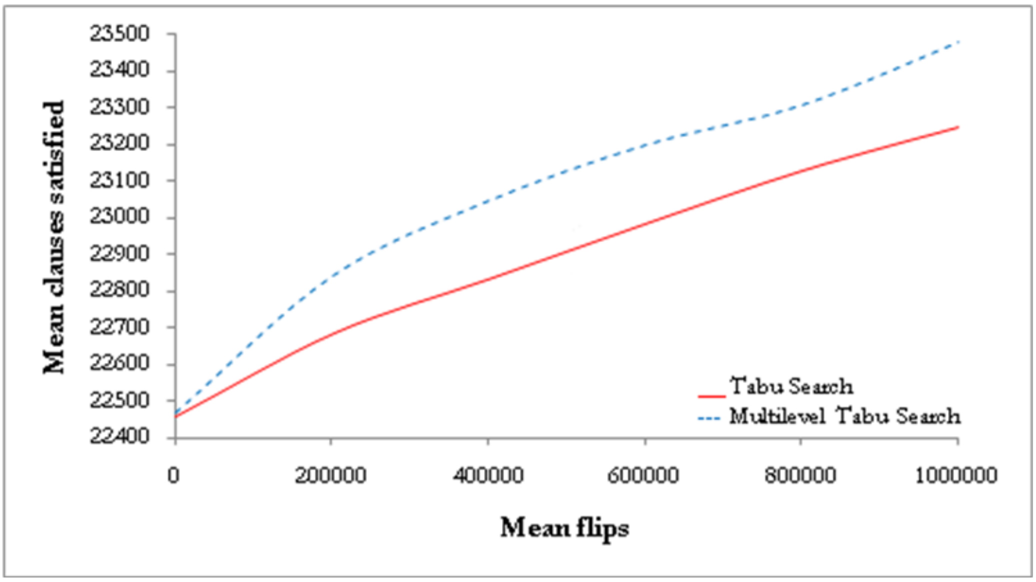

Figure 25: Tabu Search vs. Multilevel Tabu Search solving a Max SAT problem with 11265 literals and 29520 clauses (mot_comb3._red-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

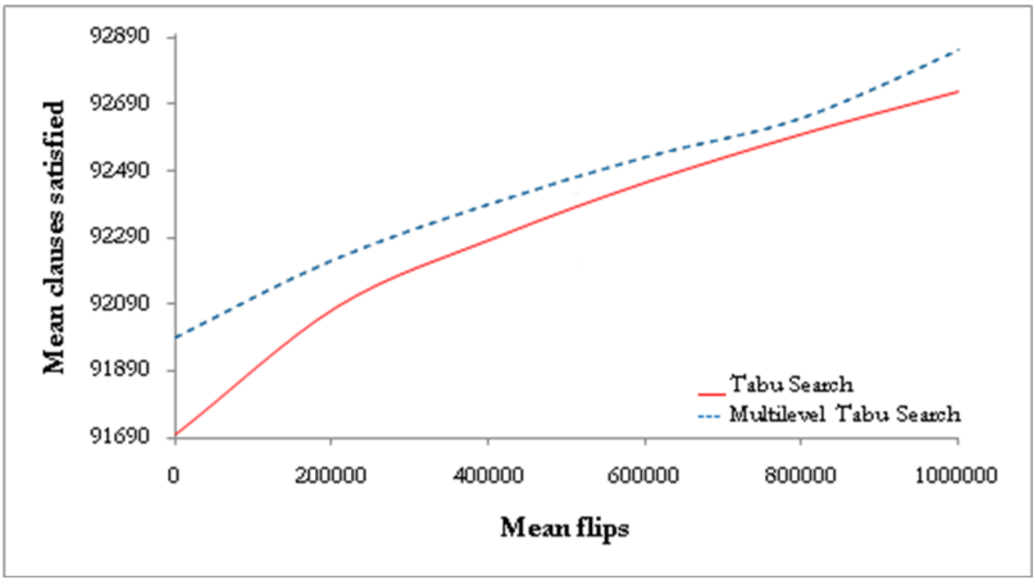

Figure 26: Tabu Search vs. Multilevel Tabu Search solving a Max SAT problem with 44079 literals and 117720 clauses (c6_DD_s3_f1_e1_v1-bug-onevec-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

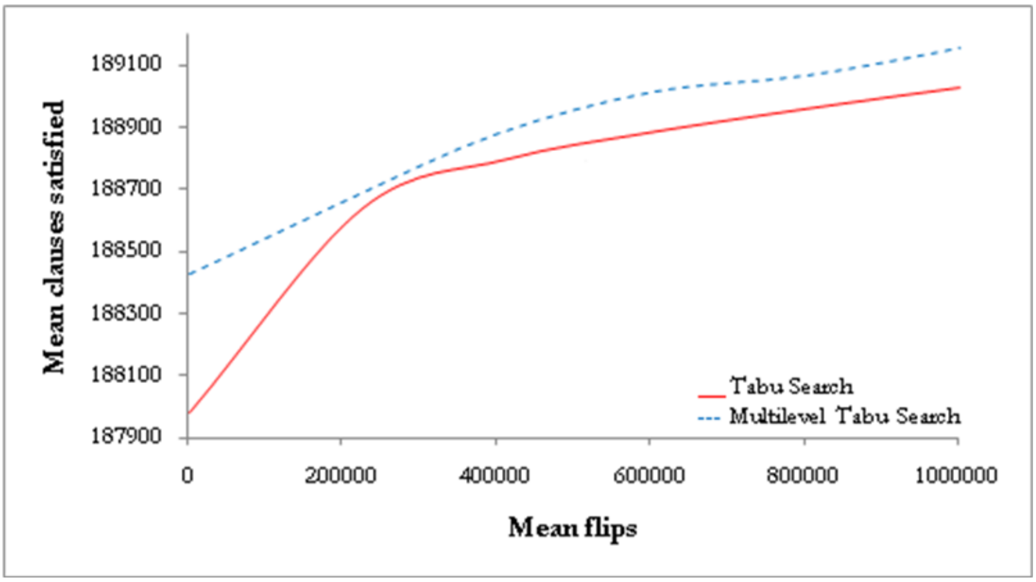

Figure 27: Tabu Search vs. Multilevel Tabu Search solving a Max SAT problem with 84525 literals and 236942 clauses (c2_DD_s3_f1_e2_v1-bug-onevec-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

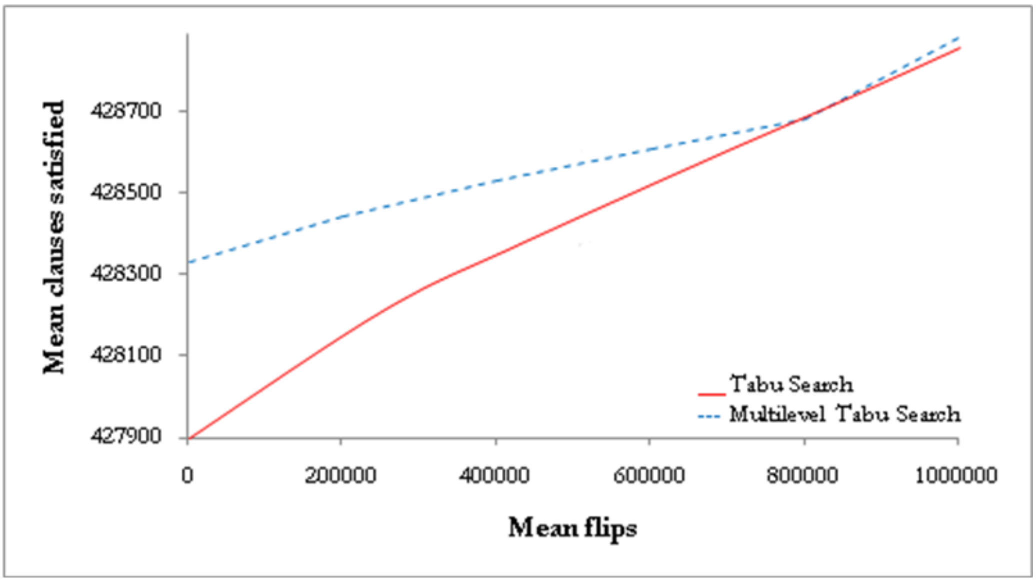

Figure 28: Tabu Search vs. Multilevel Tabu Search solving a Max SAT problem with 200944 literals and 540984 clauses (c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 24, 25, 26 and 27 illustrate that the Multilevel variant's convergence rate is slightly higher than Tabu Search and the latter ends up satisfying less clauses than the former in all cases.

Figure 28 illustrate that the Multilevel variant once again outperforms Tabu Search in terms of convergence. As can be seen in the graph, the algorithms cross at around 800000 flips. From that point, both algorithms continue to converge and once reaching the maximum amount of flips the Multilevel variant manages to satisfy more clauses than Tabu Search.

In all cases, the Multilevel variant has an advantage by having an initial solution which is higher than the initial solution of Tabu Search. The process of randomly assigning truth values to the literals differs for Multilevel (random values are assigned to clusters of literals), this could have had an effect on the initial solution. To that end, we tested a different mechanism; after the clustering process, each literal in a cluster was assigned a random logical value. The clusters would then contain literals that have different values, instead of having a single value. This mechanism did not give better results. Therefore, the manner in which literals are randomly assigned truth values does indeed affect the initial solution of a problem, and - as we have seen - the way this is done in Multilevel gives better results.

Tables 7 and 8 show the mean solved, variance and standard deviation of each problem solved by Tabu Search (TS) and Multilevel Tabu Search (MTS).

Algorithm	**Problem**	**Mean solved (%)**	**Variance**	**Standard deviation**
TS	f600	99.9 %	0.4	0.63
TS	f1000	99.7 %	0.3	0.55
TS	f2000	99.3 %	0.1	0.32
TS	medium	100 %	0	0
TS	huge	100 %	0	0
TS	bw_large.c	92.4 %	0.6	0.77
TS	bw_large.d	81.6 %	0.5	0.71
TS	logistics.d	88.8 %	0.6	0.77
TS	ewddr2-10-by-5-1	88.3 %	0.5	0.71
TS	ewddr2-10-by-5-8	88 %	0.7	0.84
TS	aim-200-2_0-yes1-4	99.8 %	0.3	0.55
TS	aim-200-3_4-yes1-2	99 %	0.4	0.63
TS	ais10	99.9 %	0.6	0.77
TS	ais12	99.9 %	0.4	0.63
TS	sw100-98	100 %	0	0
TS	sw100-99	100 %	0	0
TS	qg6-11	99.8 %	0.2	0.45
TS	qg6-12	99.8 %	0.4	0.63
TS	Max SAT #1*	80.7 %	0.3	0.55
TS	Max SAT #2*	78.8 %	0.5	0.71
TS	Max SAT #3*	78.8 %	0.7	0.84
TS	Max SAT #4*	79.8 %	0.3	0.55
TS	Max SAT #5*	79.3 %	0.2	0.45

Table 7: Mean solved, variance and standard deviation of the problems solved by the Tabu Search (TS) algorithm. *See page 49 footnote.

Algorithm	Problem	Mean solved (%)	Variance	Standard deviation
MTS	f600	98.3 %	0.3	0.55
MTS	f1000	97.8 %	0.4	0.63
MTS	f2000	97.5 %	0.2	0.45
MTS	medium	100 %	0	0
MTS	huge	100 %	0	0
MTS	bw_large.c	100 %	0	0
MTS	bw_large.d	89.6 %	0.4	0.63
MTS	logistics.d	95.3 %	0.5	0.71
MTS	ewddr2-10-by-5-1	88.4 %	0.7	0.84
MTS	ewddr2-10-by-5-8	87.7 %	0.5	0.71
MTS	aim-200-2_0-yes1-4	98.3 %	0.6	0.77
MTS	aim-200-3_4-yes1-2	97.5 %	0.5	0.71
MTS	ais10	100 %	0	0
MTS	ais12	100 %	0	0
MTS	sw100-98	100 %	0	0
MTS	sw100-99	100 %	0	0
MTS	qg6-11	100 %	0	0
MTS	qg6-12	100 %	0	0
MTS	Max SAT #1*	82 %	0.4	0.63
MTS	Max SAT #2*	79.5 %	0.2	0.45
MTS	Max SAT #3*	78.9 %	0.3	0.55
MTS	Max SAT #4*	79.8 %	0.3	0.55
MTS	Max SAT #5*	79.3 %	0.6	0.77

Table 8: Mean solved, variance and standard deviation of the problems solved by the Multilevel Tabu Search (MTS) algorithm. *See page 49 footnote.

Studying the mean solved in tables 7 and 8, it can be observed that TS provides slightly better results than MTS while solving relatively small problems. As the problems grow bigger, however, the reverse effect is observed. This indicates that the Multilevel's strength lies in solving relatively big problems, something which was initially expected. Multilevel provides better results as problems grow bigger which is a good property when it comes to solving large, complex problems. As can be seen in the tables, the variance and standard deviation are quite low in almost all cases. This is quite good as it indicates that the algorithms are overall stable and the results are not widely spread around the mean, but rather close to it.

*Max SAT #1: mot_comb2._red-gate-0.dimacs.seq.filtered
Max SAT #2: mot_comb3._red-gate-0.dimacs.seq.filtered
Max SAT #3: c6_DD_s3_f1_e1_v1-bug-onevec-gate-0.dimacs.seq.filtered
Max SAT #4: c2_DD_s3_f1_e2_v1-bug-onevec-gate-0.dimacs.seq.filtered
Max SAT #5: c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered

4.2 Learning Automata with Tabu Search vs. Multilevel Learning Automata with Tabu Search

The performances of Learning Automata with Tabu Search and Multilevel Learning Automata with Tabu Search are compared. The same set of SAT instances used in section 4.1 were used here (along with the same conditions; each instance was tested 10 times, each with a maximum flip set to 10^6 or in the case of time, 900 seconds limit), in order to perform a comparison between the algorithms. The results of the algorithms are shown in the next sections.

4.2.1 SATLIB Benchmark Problems

<u>4.2.1.1 Random</u>

Figures 29, 30 and 31 illustrate the results of solving the following random problems; 600 literals and 2550 clauses (f600), 1000 literals and 4250 clauses (f1000) and 2000 literals and 8500 clauses (f2000).

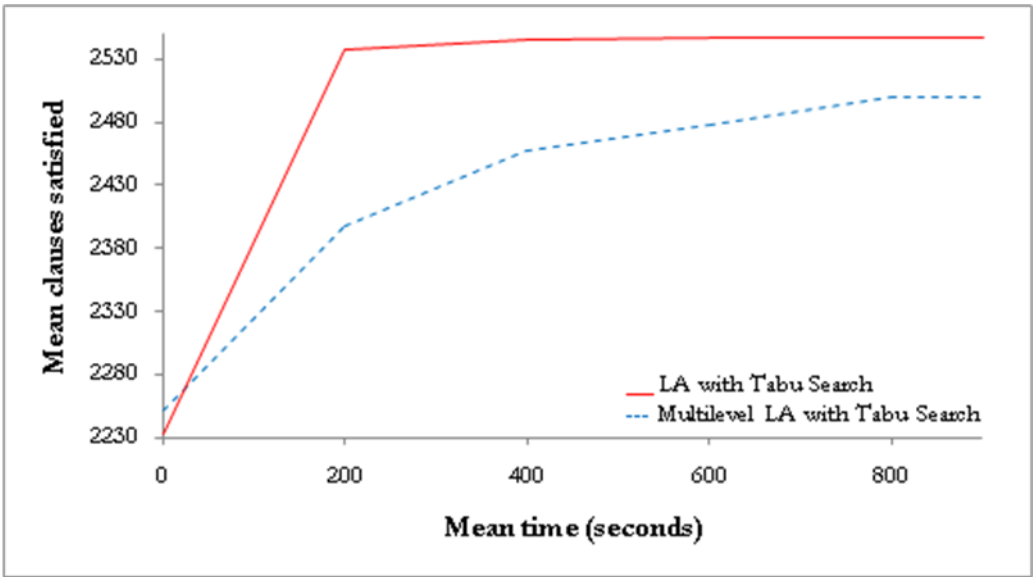

Figure 29: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a 600 literals and 2550 clauses (f600) random problem. Along the horizontal axis the mean time is given and along the vertical axis the mean number of satisfied clauses.

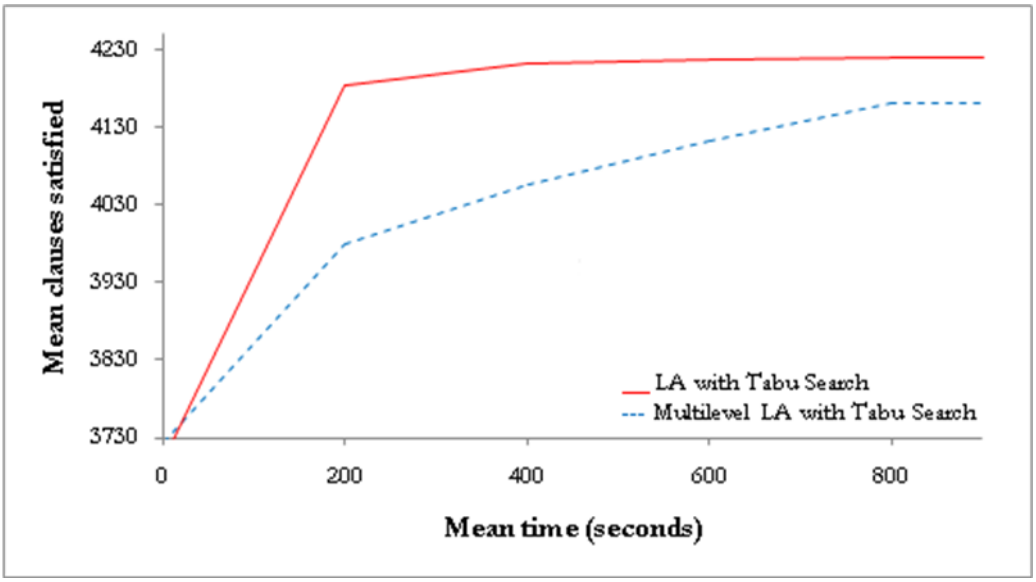

Figure 30: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a 1000 literals and 4250 clauses (f1000) random problem. Along the horizontal axis the mean time is given and along the vertical axis the mean number of satisfied clauses.

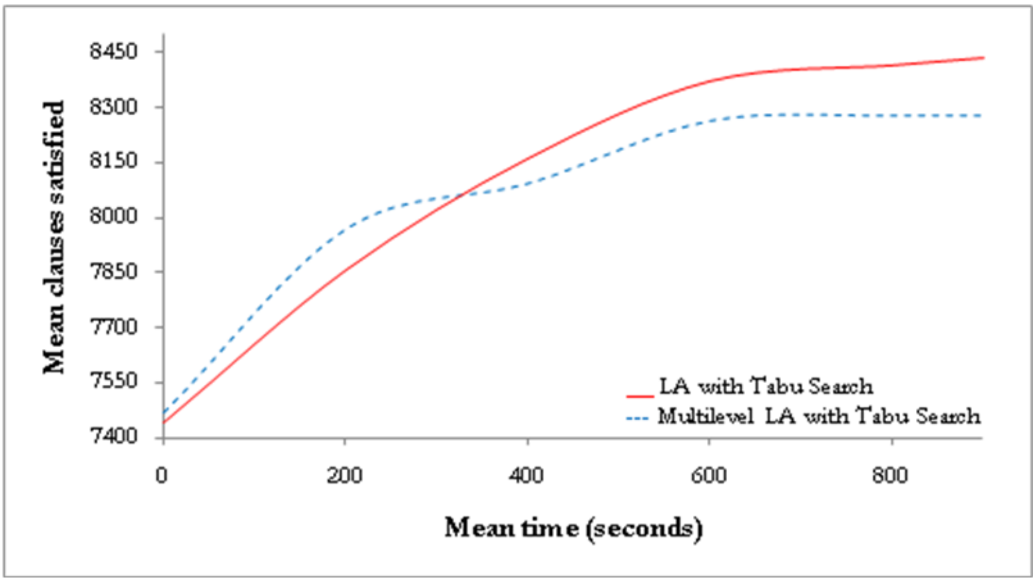

Figure 31: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a 2000 literals and 8500 clauses (f2000) random problem. Along the horizontal axis the mean time is given and along the vertical axis the mean number of satisfied clauses.

Figure 29 illustrates that f600 is a small problem, and thus the Multilevel variant is not as effective as assumed. However, as can be seen from the graph, while Learning Automata with Tabu Search starts to stagnate after around 400 seconds the Multilevel variant continues to converge until around 800 seconds from which it starts to stagnate as well. This occurs because the problem is small and the search space is restricted. As a result, it seemed that single space searching seemed most efficient for this problem.

f1000 is also a small problem and thus the Multilevel variant is also not as effective as assumed, looking at figure 30. As a result, it seemed that single space searching seemed most efficient for this problem.

f2000 is also a small problem. As observed in figure 31, the Multilevel variant's performance is approaching Learning Automata with Tabu Search's as the problems grow bigger. Both algorithms steadily converge (with a slight advantage to the Multilevel variant at the start) and Learning Automata with Tabu Search manages to satisfy more clauses than the Multilevel variant.

4.2.1.2 Planning

Figures 32, 33, 34 and 35 illustrate the results of solving the following Blocks World problems; 116 literals and 953 clauses (medium), 459 literals and 7054 clauses (huge), 3016 literals and 50457 clauses (bw_large.c) and 6325 literals and 131973 clauses (bw_large.d).

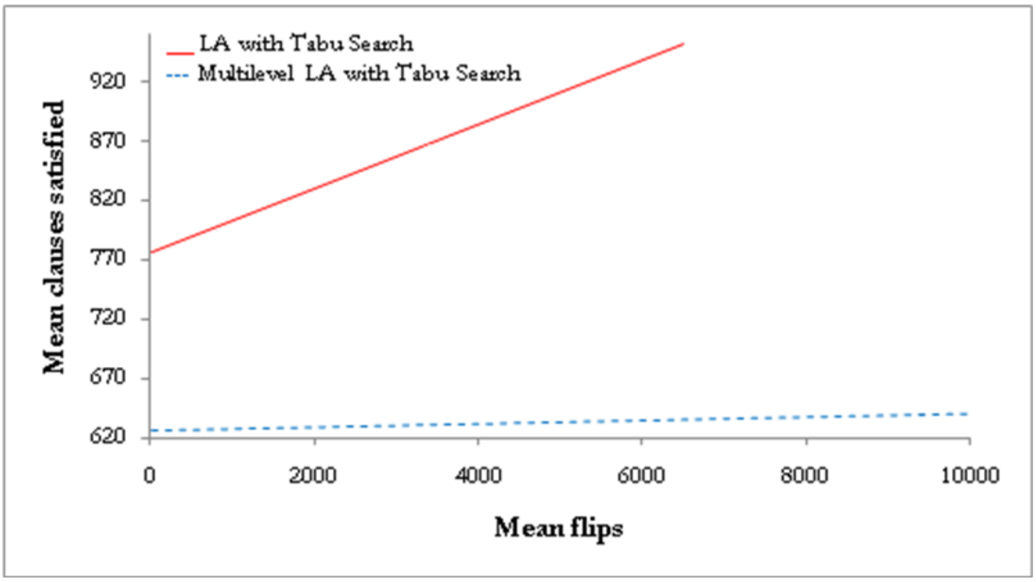

Figure 32: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Blocks World problem with 116 literals and 953 clauses problem (medium). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

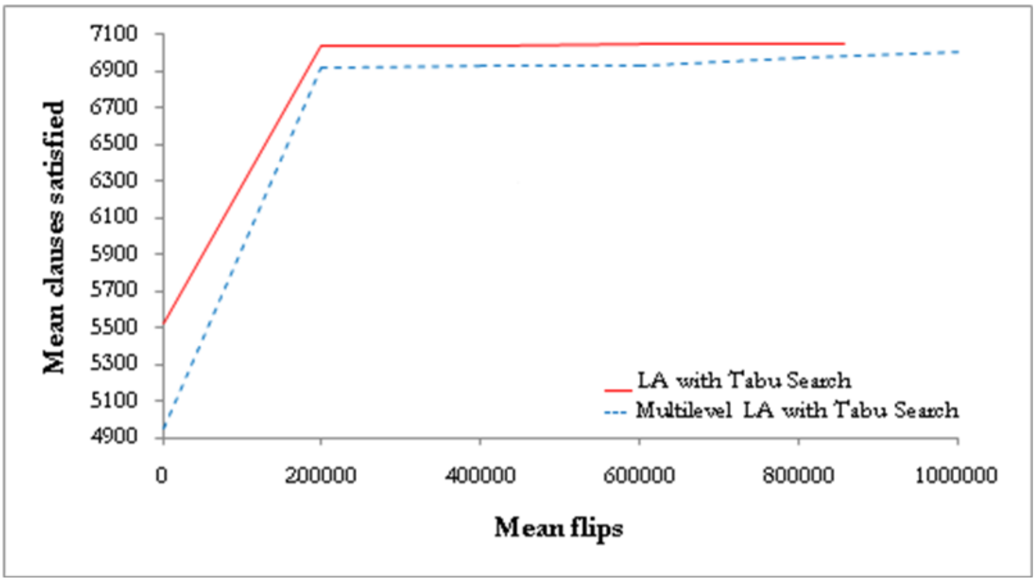

Figure 33: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Blocks World problem with 459 literals and 7054 clauses problem (huge). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

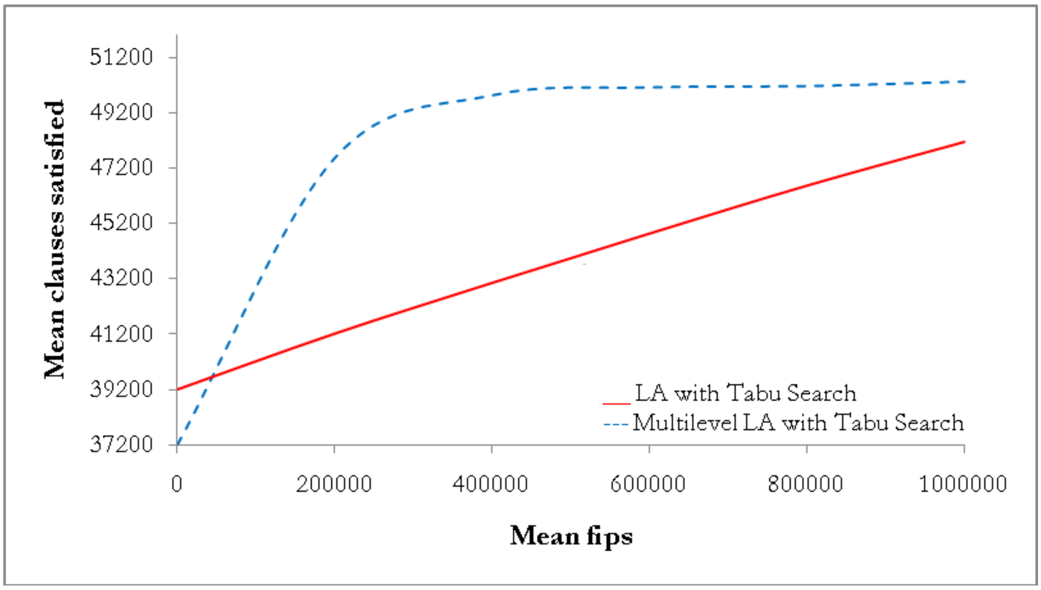

Figure 34: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Blocks World problem with 3016 literals and 50457 clauses problem (bw_large.c). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

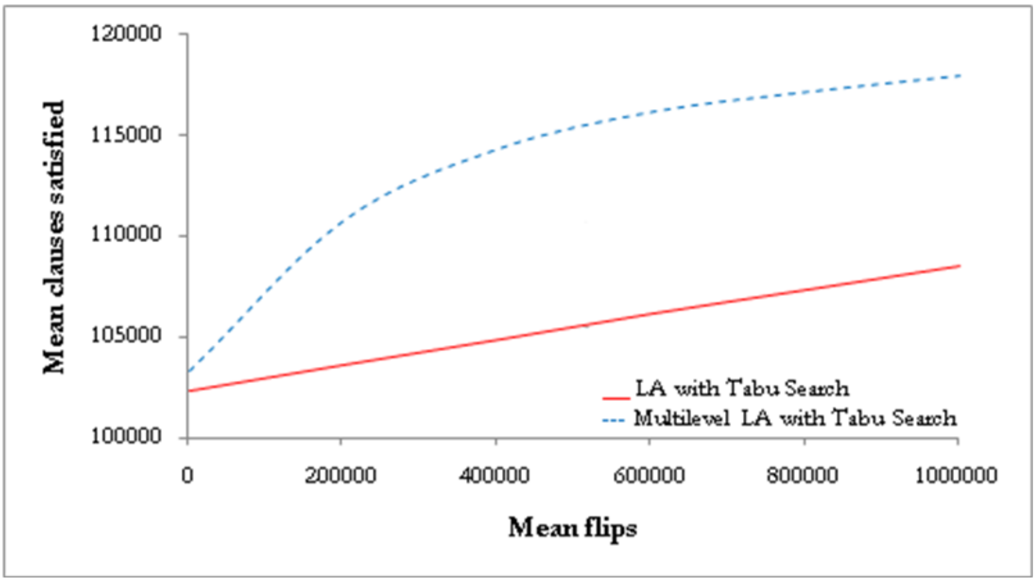

Figure 35: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Blocks World problem with 6325 literals and 131973 clauses (bw_large.d). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figure 32 illustrates a clear difference between the two algorithms, greatly favouring Learning Automata with Tabu Search. This algorithm manages to solve the problem at around 6200 flips, while its Multilevel variant does so at 10000 flips having a relatively poor convergence rate.

Figure 33 illustrates that Learning Automata with Tabu Search managed to solve this problem after around 850000 flips, while the Multilevel variant solved the problem at the maximum amount of flips. In this case, the combination of Learning Automata with Tabu Search beat its Multilevel variant.

As illustrated in figure 34, the Multilevel variant shows a clear dominance to Learning Automata with Tabu Search. While both algorithms do not manage to solve the problem, the convergence rate favours the Multilevel variant.

As illustrated in figure 35, the Multilevel variant is once again superior to Learning Automata with Tabu Search. Multilevel excels in solving this problem due to its fairly big size. While reaching the maximum amount of flips, the convergence rate of Multilevel is much higher than Learning Automata with Tabu Search. It can be clearly observed here that the bigger the SAT problem is, the more it is in favour of the Multilevel variant.

Figure 36 illustrate the results of solving a Logistics problem with 4713 literals and 21991 clauses (logistics.d).

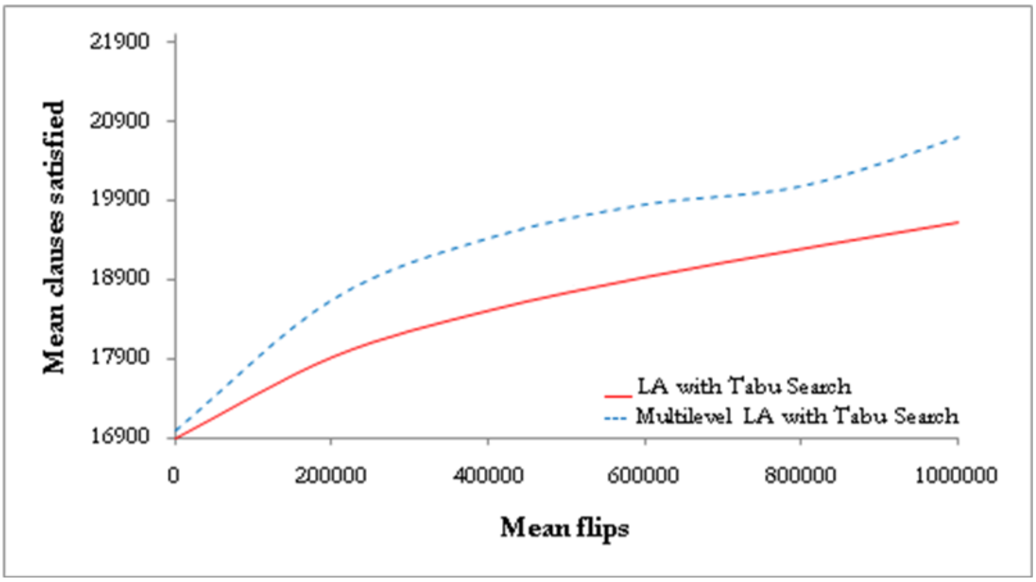

Figure 36: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Logistics problem with 4713 literals 21991 clauses (logistics.d). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figure 36 illustrates that the Multilevel variant is clearly superior to Learning Automata with Tabu Search in terms of convergence efficiency and quality.

4.2.1.3 SAT Competition Beijing

Figures 37 and 38 illustrate the results of solving the following Beijing problems; 21800 literals and 118607 clauses (ewddr2-10-by-5-1) and 22500 literals and 123329 clauses (ewddr2-10-by-5-8).

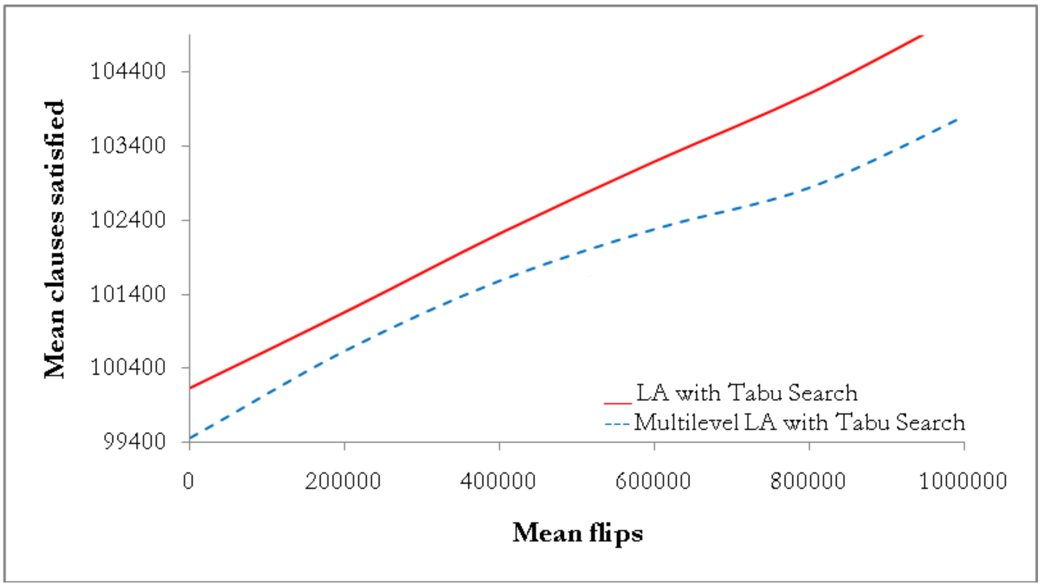

Figure 37: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Beijing problem with 21800 literals and 118607 clauses (ewddr2-10-by-5-1). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

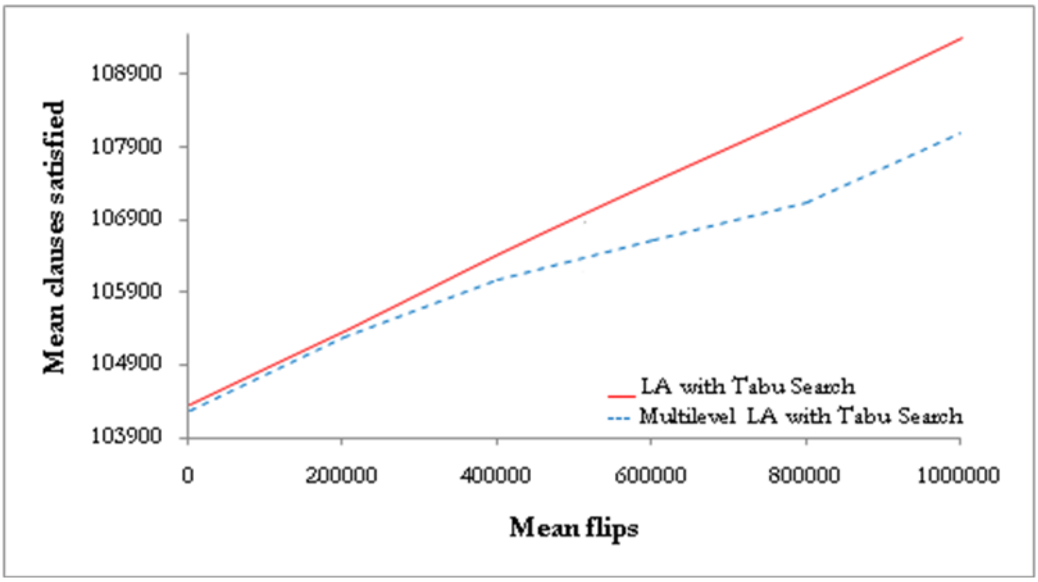

Figure 38: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Beijing problem with 22500 literals and 123329 clauses (ewddr2-10-by-5-8). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 37 and 38 illustrate that Learning Automata with Tabu Search beats its Multilevel variant while solving these fairly big problems. The results here indicate that the combination of Learning Automata and Tabu Search actually provides better results than using a Multilevel variant of the two while solving these fairly big problems.

4.2.1.4 AIM

Figures 39 and 40 illustrate the results of solving the following AIM problems; 200 literals and 400 clauses (aim-200-2_0-yes1-4) and 200 literals and 680 clauses (aim-200-3_4-yes1-2).

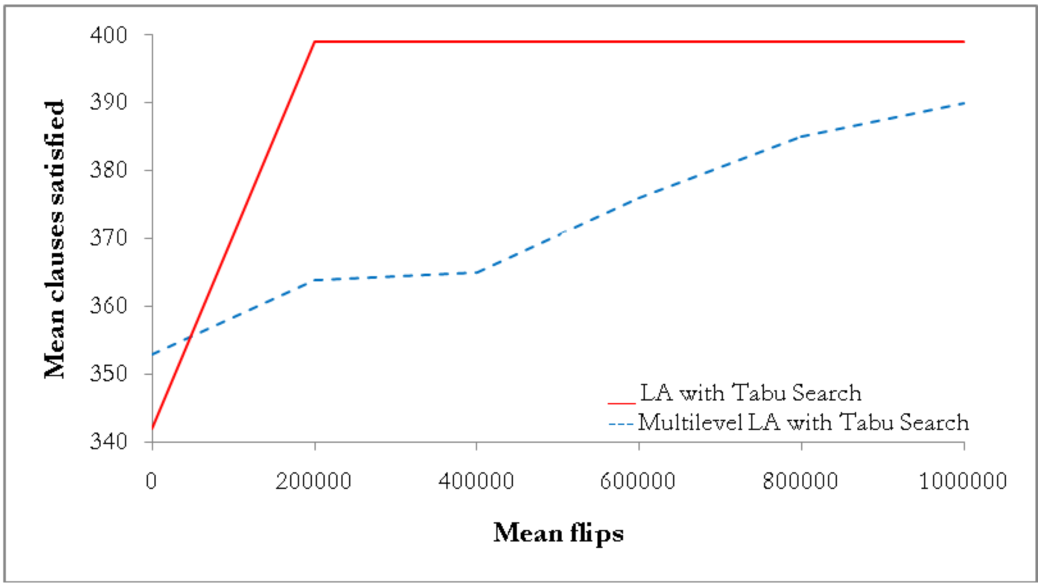

Figure 39: LA with Tabu Search vs. Multilevel LA with Tabu Search solving an AIM problem with 200 literals and 400 clauses (aim-200-2_0-yes1-4). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

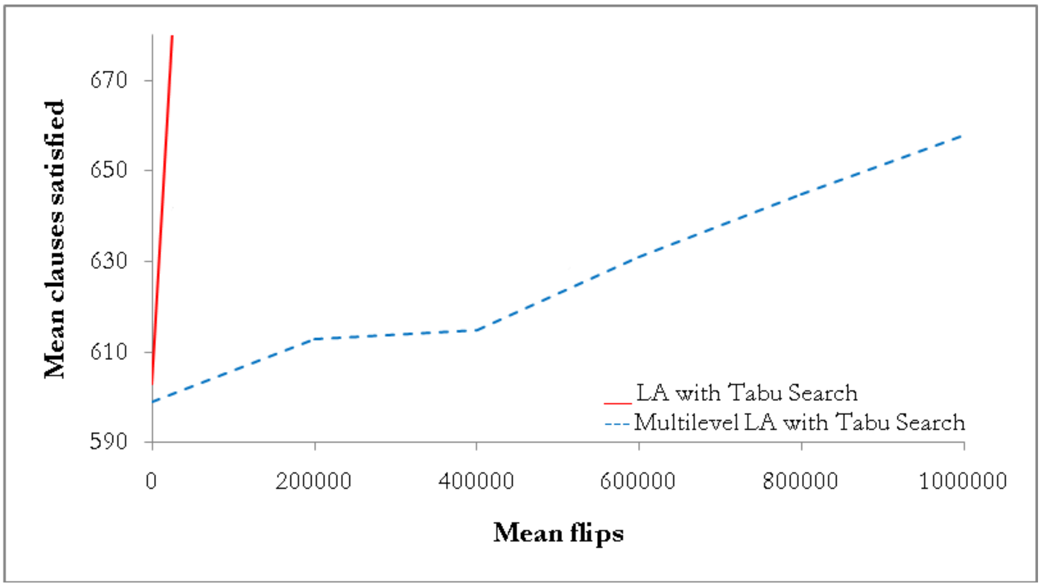

Figure 40: LA with Tabu Search vs. Multilevel LA with Tabu Search solving an AIM problem with 200 literals and 680 clauses (aim-200-3_4-yes1-2). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figure 39 illustrates that Learning Automata with Tabu Search clearly outperforms the Multilevel variant in terms of convergence. However, the former shows an early stagnation which is not observed in the latter. Similarly as in section 4.1.1.4, the convergence favour to Learning Automata with Tabu Search here confirms that the Multilevel variant does not work well in relatively small problems.

Figure 40 illustrates that Learning Automata with Tabu Search once more clearly outperforms its Multilevel variant. In this case having a steep convergence without stagnating.

4.2.1.5 All Interval Series (AIS)

Figures 41 and 42 illustrate the results of solving the following All Interval Series (AIS) problems; 181 literals and 3151 clauses (ais10) and 265 literals and 5666 clauses (ais12).

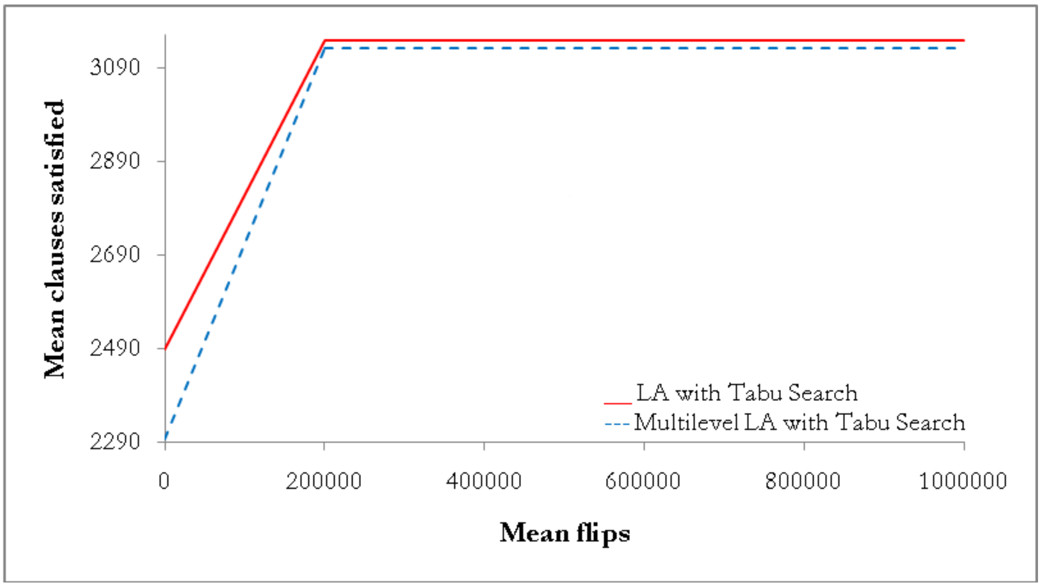

Figure 41: LA with Tabu Search vs. Multilevel LA with Tabu Search solving an AIS problem with 181 literals and 3151 clauses (ais10). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

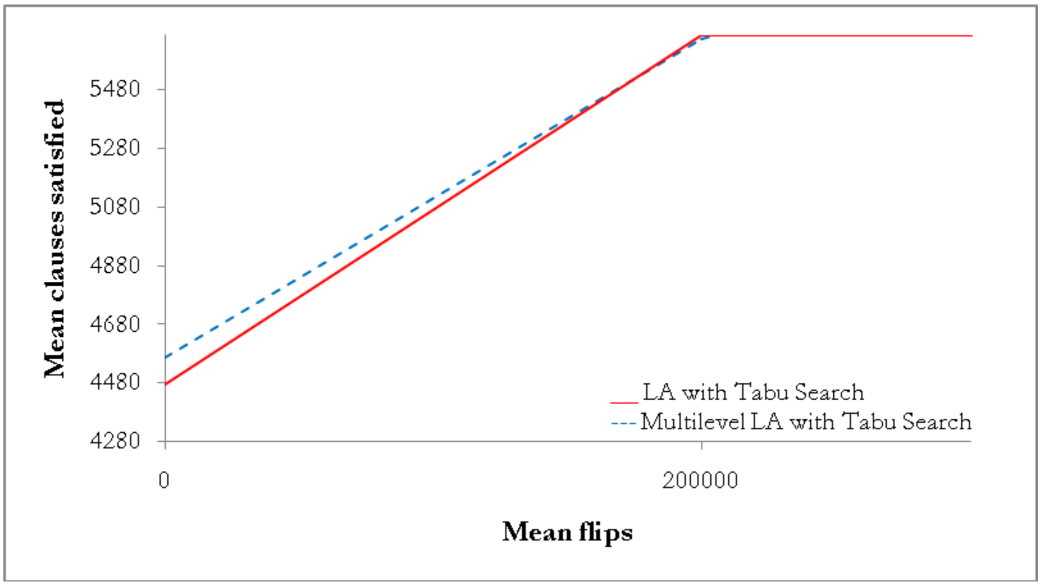

Figure 42: LA with Tabu Search vs. Multilevel LA with Tabu Search solving an AIS problem with 265 literals and 5666 clauses (ais12). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figure 41 illustrates a slight convergence favour to Learning Automata with Tabu Search until it starts to stagnate at around 200000 flips. Similarly, the Multilevel variant starts to stagnate at the same time and continues on without managing to solve the problem.

Figure 42 also illustrates a slight convergence favour, however in this case to the Multilevel variant. While Multilevel manages to solve the problem at around 200000 flips, Learning Automata with Tabu Search starts to stagnate at this point and continues on without managing to solve the problem.

4.2.1.6 Graph Colouring SW

Figures 43 and 44 illustrate the results of solving the following Graph Colouring SW problems; 500 literals and 3100 clauses (sw100-98) and 500 literals and 3100 clauses (sw100-99).

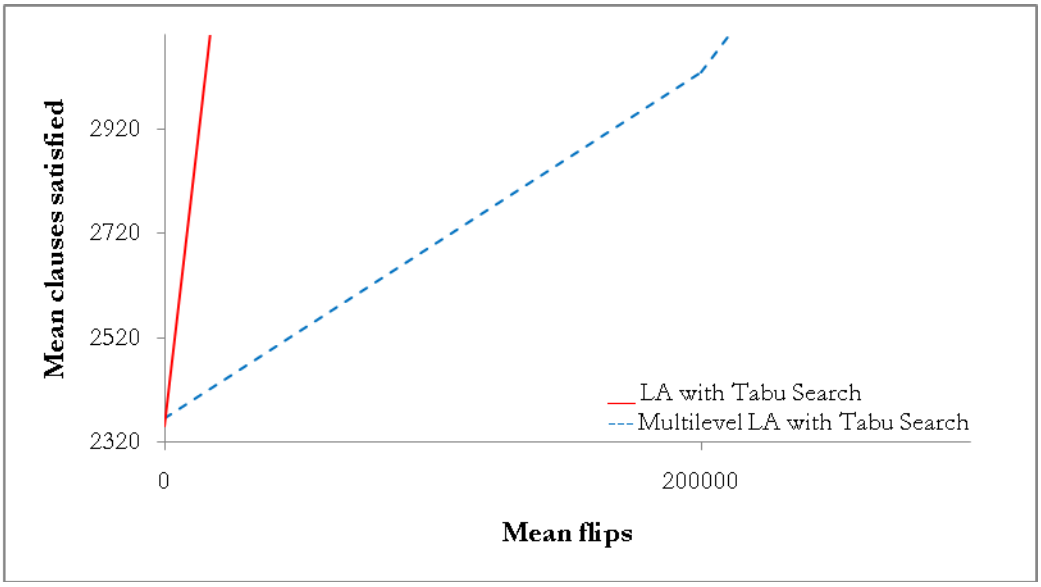

Figure 43: LA with Tabu Search vs. Multilevel LA with Tabu Search solving an Graph Colouring SW problem with 500 literals and 3100 clauses (sw100-98). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

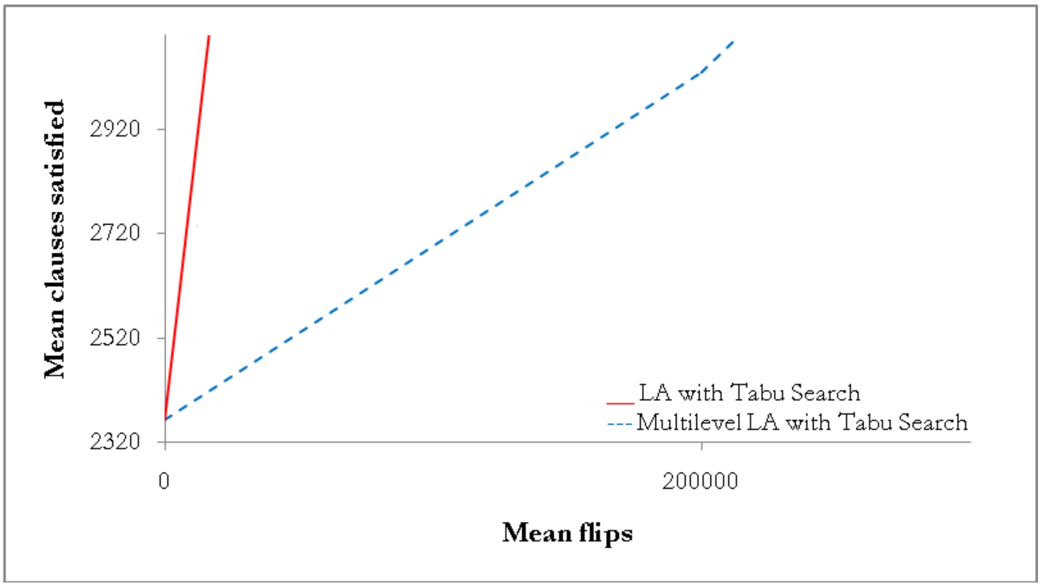

Figure 44: LA with Tabu Search vs. Multilevel LA with Tabu Search solving an Graph Colouring SW problem with 500 literals and 3100 clauses (sw100-99). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 43 and 44 illustrate that Learning Automata with Tabu Search clearly beats Multilevel in terms of convergence, solving both problems in well under 200000 flips. Multilevel manages to solve both problems slightly above 200000 flips. The results of both algorithms are quite similar due to their respective sizes. Interestingly, these results are quite similar to those earlier shown in section 4.1.1.6.

4.2.1.7 Quasi Groups

Figures 45 and 46 illustrate the results of solving the following Quasi Groups problems; 1331 literals and 49204 clauses (qg6-11) and 1728 literals and 69931 clauses (qg6-12).

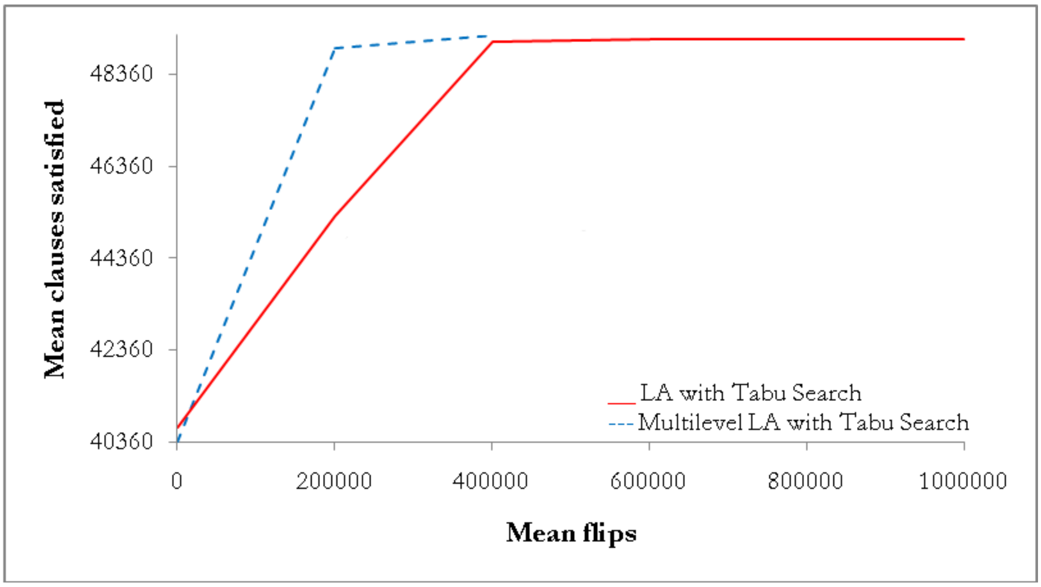

Figure 45: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Quasi Groups problem with 1331 literals and 49204 clauses (qg6-11). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

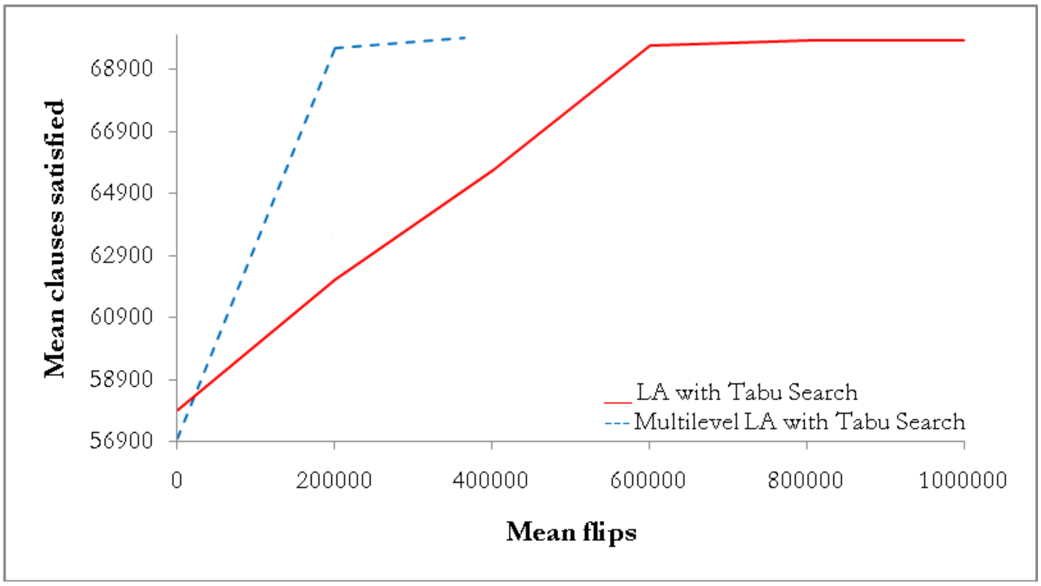

Figure 46: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Quasi Groups problem with 1728 literals and 69931 clauses (qg6-12). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 45 and 46 illustrate that the Multilevel variant clearly outperforms Learning Automata with Tabu Search, both in convergence rate and quality. In figure 45 it is seen that Learning Automata with Tabu Search starts to stagnate at around 600000 and continues on without managing to solve the problem, while Multilevel manages to solve the problem at around 400000 flips. Similarly, figure 46 shows that Learning Automata with Tabu Search starts to stagnate at around 800000 flips and does not manage to solve the problem while Multilevel does in just under 400000 flips.

4.2.2 Max SAT Problems

4.2.2.1 Industry

Figures 47, 48, 49, 50 and 51 illustrate the results of solving the following Max SAT (Industry) problems; 5484 literals and 13894 clauses (mot_comb2._red-gate-0.dimacs.seq.filtered), 11265 literals and 29520 clauses (mot_comb3._red-gate-0.dimacs.seq.filtered), 44079 literals and 117720 clauses (c6_DD_s3_f1_e1_v1-bug-onevec-gate-0.dimacs.seq.filtered), 84525 literals and 236942 clauses (c2_DD_s3_f1_e2_v1-bug-onevec-gate-0.dimacs.seq.filtered) and 200944 literals and 540984 clauses (c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered).

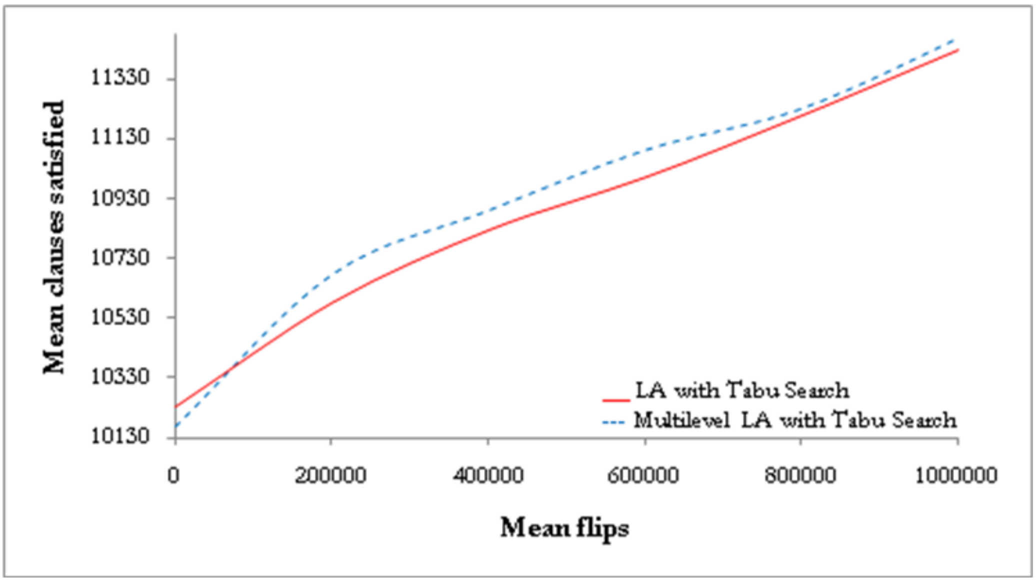

Figure 47: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Max SAT problem with 5484 literals and 13894 clauses (mot_comb2._red-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

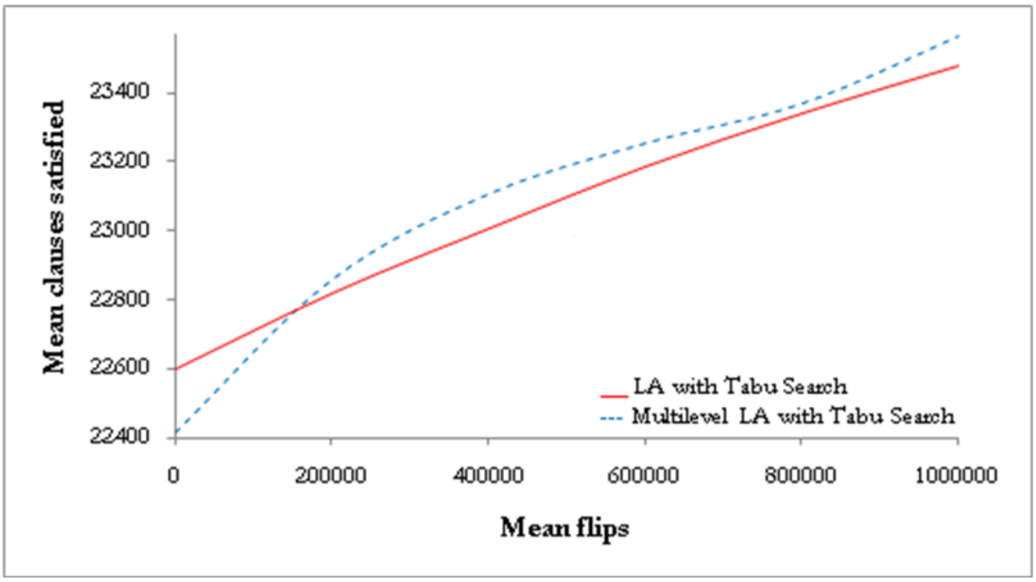

Figure 48: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Max SAT problem with 11265 literals and 29520 clauses (mot_comb3._red-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

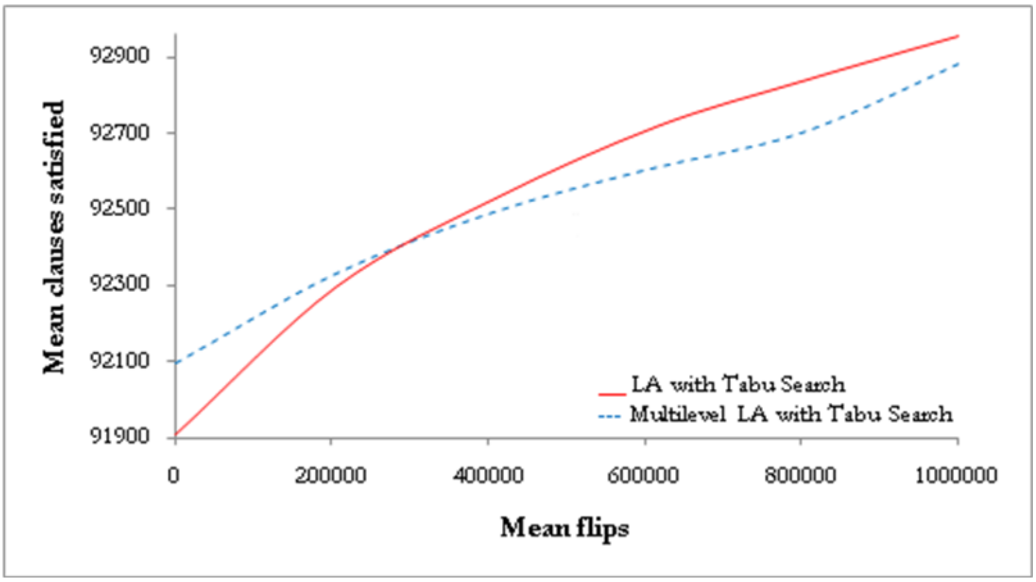

Figure 49: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Max SAT problem with 44079 literals and 117720 clauses (c6_DD_s3_f1_e1_v1-bug-onevec-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

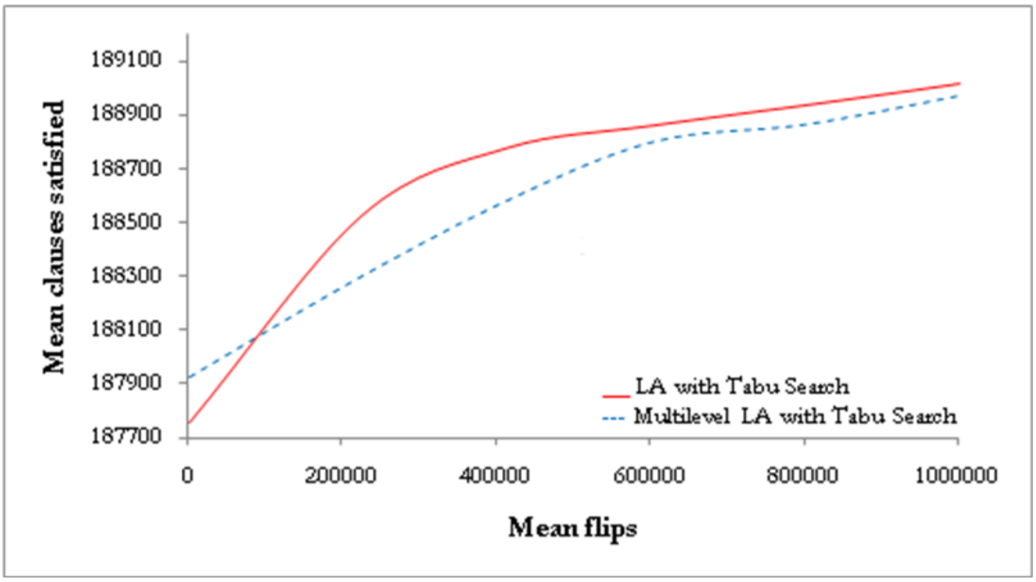

Figure 50: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Max SAT problem with 84525 literals and 236942 clauses (c2_DD_s3_f1_e2_v1-bug-onevec-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

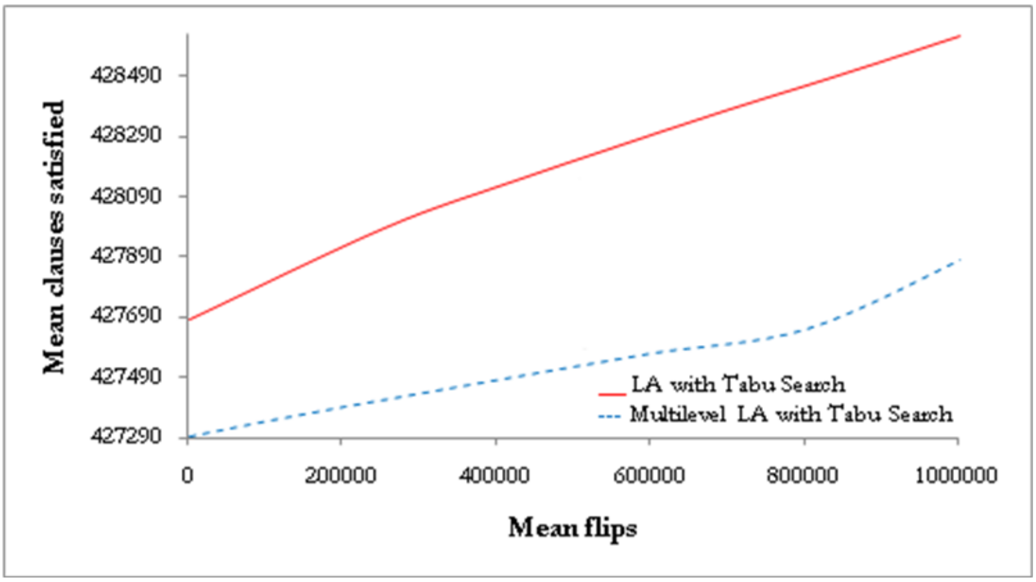

Figure 51: LA with Tabu Search vs. Multilevel LA with Tabu Search solving a Max SAT problem with 200944 literals and 540984 clauses (c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered). Along the horizontal axis the mean number of flips is given and along the vertical axis the mean number of satisfied clauses.

Figures 47 and 48 illustrate that the Multilevel variant's convergence rate is higher than Learning Automata with Tabu Search and the latter satisfies less clauses than the former in those cases in both cases.

Figures 49, 50 and 51 illustrate rather interesting and surprising results. Learning Automata with Tabu Search seem to outperform the Multilevel variant while solving these large problems, both in terms of convergence and quality. This is an indication that Learning Automata with Tabu Search is a solid algorithm that manages to beat Multilevel while solving these problems.

Tables 9 and 10 show the mean solved, variance and standard deviation of each problem solved by Learning Automata with Tabu Search (LATS) and Multilevel Learning Automata with Tabu Search (MLATS).

Algorithm	**Problem**	**Mean solved (%)**	**Variance**	**Standard deviation**
LATS	f600	100 %	0	0
LATS	f1000	99.8 %	0.3	0.55
LATS	f2000	99.5 %	0.4	0.63
LATS	medium	100 %	0	0
LATS	huge	100 %	0	0
LATS	bw_large.c	95.4 %	0.2	0.45
LATS	bw_large.d	82.3 %	0.5	0.71
LATS	logistics.d	89.3 %	0.6	0.77
LATS	ewddr2-10-by-5-1	88.7 %	0.4	0.63
LATS	ewddr2-10-by-5-8	88.7 %	0.3	0.55
LATS	aim-200-2_0-yes1-4	99.8 %	0.7	0.84
LATS	aim-200-3_4-yes1-2	99.3 %	0.5	0.71
LATS	ais10	99.9 %	0.8	0.89
LATS	ais12	99.9 %	0.3	0.55
LATS	sw100-98	100 %	0	0
LATS	sw100-99	100 %	0	0
LATS	qg6-11	99.8 %	0.4	0.63
LATS	qg6-12	99.8 %	0.9	0.95
LATS	Max SAT #1*	82.3 %	0.4	0.63
LATS	Max SAT #2*	79.5 %	0.7	0.84
LATS	Max SAT #3*	80 %	0.7	0.84
LATS	Max SAT #4*	79.8 %	0.5	0.71
LATS	Max SAT #5*	79.2 %	0.8	0.89

Table 9: Mean solved, variance and standard deviation of the problems solved by the Learning Automata with Tabu Search (LATS) algorithm. *See page 64 footnote.

Algorithm	**Problem**	**Mean solved (%)**	**Variance**	**Standard deviation**
MLATS	f600	98 %	0.4	0.63
MLATS	f1000	97.9 %	0.3	0.55
MLATS	f2000	97.4 %	0.6	0.77
MLATS	medium	97.6 %	0.5	0.71
MLATS	huge	99.4 %	0.3	0.55
MLATS	bw_large.c	99.8 %	0.3	0.55
MLATS	bw_large.d	89.4 %	0.2	0.45
MLATS	logistics.d	94.1 %	0.7	0.84
MLATS	ewddr2-10-by-5-1	87.5 %	0.5	0.71
MLATS	ewddr2-10-by-5-8	87.7 %	0.3	0.55
MLATS	aim-200-2_0-yes1-4	97.5 %	0.6	0.77
MLATS	aim-200-3_4-yes1-2	96.8 %	0.4	0.63
MLATS	ais10	99.4 %	0.6	0.77
MLATS	ais12	100 %	0	0
MLATS	sw100-98	100 %	0	0
MLATS	sw100-99	100 %	0	0
MLATS	qg6-11	100 %	0	0
MLATS	qg6-12	100 %	0	0
MLATS	Max SAT #1*	82.5 %	0.4	0.63
MLATS	Max SAT #2*	79.8 %	0.6	0.77
MLATS	Max SAT #3*	78.9 %	0.3	0.55
MLATS	Max SAT #4*	77.4 %	0.7	0.84
MLATS	Max SAT #5*	79.1 %	0.5	0.71

Table 10: Mean solved, variance and standard deviation of the problems solved by the Multilevel Learning Automata with Tabu Search (MLATS) algorithm. *See page 64 footnote.

While studying the mean solved in tables 9 and 10, rather interesting results are observed. As expected, LATS provides slightly better results than MLATS while solving relatively small problems. However, it is also observed that LATS in some cases beats its Multilevel variant while solving relatively big problems. This phenomenon is especially observable in the SAT competition Beijing and Max SAT problems, as illustrated in the tables. Studying the results obtained earlier in tables 7 and 8 in section 4.1 - while analysing TS and MTS - the results obtained here are rather surprising. While the Multilevel variant was expected to beat its counterpart in solving all large problems, it did not here. The single combination of Learning Automata and Tabu Search gave slightly better results than a Multilevel approach while solving relatively big problems. This is not entirely conclusive however, since in other cases the Multilevel variant gave better results. In any case, this is something worth investigating in the future. Once again, the variance and standard deviation are quite low in almost all cases here. As previously mentioned, this is quite good as it indicates that the algorithms are overall stable and the results are not widely spread around the mean, but rather close to it.

*Max SAT #1: mot_comb2._red-gate-0.dimacs.seq.filtered
Max SAT #2: mot_comb3._red-gate-0.dimacs.seq.filtered
Max SAT #3: c6_DD_s3_f1_e1_v1-bug-onevec-gate-0.dimacs.seq.filtered
Max SAT #4: c2_DD_s3_f1_e2_v1-bug-onevec-gate-0.dimacs.seq.filtered
Max SAT #5: c5_DD_s3_f1_e1_v1-bug-gate-0.dimacs.seq.filtered

5 Discussion

Having run the implemented algorithms on SAT problems from SATLIB [21] and Max SAT [29], rather interesting results were obtained. Tabu Search (TS) seemed to be an effective algorithm solving relatively small problems, while the new Multilevel Tabu Search (MTS) algorithm excelled primarily in solving relatively big problems. This observation was expected initially due to the Multilevel technique's unique mechanism of effectively handling big amounts of literals. It was observed that the size of problems was proportional to the performance of MTS. Meaning that as the problems got bigger, the better MTS performed, and vice versa.

Interestingly, this was not the case with the new Learning Automata with Tabu Search (LATS) and Multilevel Learning Automta with Tabu Search (MLATS) algorithms. It seemed that the combination of Learning Automata and Tabu Search gave good results, to the extent of being on level with its Multilevel variant, when it came to solution quality. LATS managed to slightly beat MLATS in some cases while solving relatively big problems. This was a surprising observation that shows the advantage of using Learning Automata, and the indication that this is a good algorithm. When it came to convergence time, the Multilevel technique was a clear winner in all cases. To obtain a general overview of the results of the algorithms, consider table 11 which shows the mean solution quality and convergence time of each algorithm solving the entire set of the SAT instances (23 problems in total).

Algorithm	**Mean solution quality (%)**	**Mean convergence time (seconds)**
Tabu Search (TS)	88.42 %	3136.5 s.
Multilevel Tabu Search (MTS)	93.47 %	1009.29 s.
Learning Automata with Tabu Search (LATS)	93.17 %	2994.72 s.
Multilevel Learning Automata with Tabu Search (MLATS)	93.05 %	1254.03 s.

Table 11: The mean solution quality and convergence time of each algorithm solving the set of SAT instances (23 problems in total).

As can be seen in table 11, the difference in mean solution quality between TS and MTS is quite clear. MTS has a mean solution quality advantage of just above 5 %. This shows that a Multilevel approach is better (than a non-Multilevel) overall. This is not the case when it comes to LATS and MLATS, as seen in the table. Both algorithms have quite similar mean solution quality, with a 0.12 % advantage to LATS. As mentioned earlier, this observation is rather surprising. Comparing TS to LATS, it is clear that LATS is the better algorithm with a mean solution quality advantage of 4.75 %. This is an indication that Learning Automata is a technique that works quite well with Tabu Search, and is definitely worth investigating in the future. The best overall algorithm is MTS, this is an(other) indication that Multilevel is a technique that works quite well with Tabu Search - specifically excelling in solving relatively big SAT problems (as previously observed from the results in section 4.1). Looking at the mean convergence time, it can be seen that MTS is more than three times faster than TS, reaching a better solution than the latter. Similarly, MLATS is almost three times faster than LATS. It is quite clear that Multilevel is a technique that greatly increases the efficiency of the algorithms. Based on these results, using a Multilevel approach to solve SAT is definitely recommendable. In addition, the calculated variance and standard deviation of the algorithms are relatively low in all cases which indicates that the algorithms are quite stable and the results are not widely spread around the mean, but close to it.

6 Conclusion and Further Work

In this work, the problem was to solve the Boolean Satisfiability Problem (SAT) by introducing a clustering technique - Multilevel - and combining the latter with two existing approaches - Tabu Search and Learning Automata. Thereafter disclosing whether this combination provides better results - than using the two mentioned approaches alone - while solving SAT. SAT is a nondeterministic polynomial time (NP) complete problem which is a Boolean expression composed of a specific amount of variables (literals), clauses that contain disjunctions of the literals and conjunctions of the clauses. The literals have logical values TRUE and FALSE and the task is to find a truth assignment that makes the entire expression TRUE (satisfied).

The proposed Multilevel paradigm consists of three phases; clustering, initial solution and refinement. In the first phase, the SAT instance is simplified by dividing the number of literals in several levels - literals are clustered together. The clustering process can either be performed randomly, or deterministically (by clustering neighbouring literals). Once the clustering process is complete and a final desired level is reached, the clusters (of literals) are randomly assigned logical TRUE/FALSE values and an initial solution is calculated in the second phase. In the final phase of the paradigm, any metaheuristic algorithm may be used. In this work, we have used Tabu Search and Learning Automata. A total of four algorithms were implemented; Tabu Search (TS), Multilevel Tabu Search (MTS), Learning Automata with Tabu Search (LATS) and Multilevel Learning Automata with Tabu Search (MLATS) - the last three algorithms being an all-new contribution. Having implemented each algorithm and a Multilevel variant of itself, we were able to conduct a comparison analysis to disclose whether the Multilevel clustering technique provided better results in terms of solution quality and computational efficiency.

The obtained results were interesting. TS seemed to be an effective algorithm solving relatively small problems, while MTS excelled primarily in solving relatively big problems. It was observed that the size of problems was proportional to the performance of MTS. Meaning that the bigger the problems became, the better MTS performed, and vice versa. This phenomenon was due to the Multilevel's unique mechanism of handling big amounts of literals as clusters. This was however not always the case with LATS and MLATS, as the latter was sometimes beaten by the former in solving relatively big problems. This was a surprising observation that showed the great advantage of using Learning Automata. When it came to mean convergence time, using Multilevel was a definite advantage as results showed the latter being up to three times faster than a single level approach (see table 11).

Based on the results obtained, using the Multilevel technique definitely increased the efficiency of the Tabu Search and Learning Automata approaches. By these results, we have proven our hypothesis; combining the Multilevel technique with existing approaches did indeed increase the efficiency of solving SAT. As steps for further work, it is worth mentioning that the singular combination of Learning Automata and Tabu Search showed great promise and is definitely worth investigating.

References

[1] A. Armando and L. Compagna, Abstraction-Driven SAT-based Analysis of Security Protocols, *DIST - Universita degli Studi di Genova,* 2003.

[2] F. Guillaume, The SAT Problem of MU-Calculus over Petri Nets, INRIA, *France.*

[3] J. Esparza. On the decidability of model checking for several mu-calculi and Petri Nets. *In S. Tison, editor, Proceedings of Trees in Algebra and Programming-CAAP '94, 19th International Colloquium 1994, number 787 in Lecture Notes in Computer Science*, pages 115-129, 1994.

[4] http://www.cse.ohio-state.edu/~gurari/theory-bk/theory-bk-fivese3.html, Eitan M. Gurari, *Ohio State University, Department of Computer Science and Engineering,* downloaded 19.11.2010.

[5] O-C. Granmo, Selected Learning Problems in Stochastic Optimization Part III: The Boolean Satisfiability Problem (SAT), *University of Agder IKT507 Lecture,* 2010.

[6] O-C.Granmo and N. Bouhmala, Solving the Satisfiability Problem Using Finite Learning Automata, *International Journal of Computer Science and Applications,* Vol. 4 Issue 3 p. 15-29, 2007.

[7] B. Selman, H. Levesque, and D. Mitchell, A New Method for Solving Hard Satisfiability Problems, *Proc. of AAA'92*, MIT Press, 440-446, 1992.

[8] B. Selman, Henry A. Kautz, and B. Cohen, Noise Strategies for Improving Local Search, *Proc. of AAAI'94*, MIT Press, 337-343, 1994.

[9] D. McAllester, B. Selman, and H. Kautz, Evidence for Invariants in Local Search, *Proc. of AAAI'97*, MIT Press, 321-326, 1997.

[10] M. Davis and H. Putnam, A computing procedure for quantification theory, *Journal of the ACM*, 7:201-215, 1960.

[11] B. Ferris and J. Froehlich, Walk SAT as an Informed Heuristic to DPLL in SAT Solving, *Department of Computer Science, University of Washington, Seattle.*

[12] W. M. Spears, Simulated Annealing for Hard Satisfiability Problems, Technical Report, Naval Research Laboratory, Washington D.C., 1993.

[13] A. E. Eiben and J. K. van der Hauw, Solving 3-SAT with Adaptive Genetic Algorithms, *Proc. of the 4th IEEE Conference on Evolutionary Computation*, IEEE Press, 81-86, 1997.

[14] B. Mazure, L. Saïs and E. Gregoire, Tabu Search for SAT, *AAAI-97 Proceedings*, p. 281 - 285, 1997.

[15] B. Cha and K. Iwama, Performance Tests of Local Search Algorithms Using New Types of Random CNF Formula, *Proc. of IJCAI'95*, Morgan Kaufmann Publishers, 304-309, 1995.

[16] J. Frank, Learning Short-term Clause Weights for GSAT, *Proc. of IJCAI'97*, Morgan Kaufmann Publishers, 384-389, 1997.

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

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

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

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

[21] http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html, SATLIB Benchmark Problems, *The University of British Columbia, Computer Science*, downloaded 11.12.2010.

[22] B. Selman and H. Kautz, Domain-independent extensions t o GSAT: Solving large structured satisfiability problems. In *Proceedings of IJCAI*, 1993.

[23] J . Frank. Learning short-term weights for GSAT. Technical report, http://rainier.cs.ucdavis.edu/frank/decay.ml96.ps, University of California at Davis, October 1996.

[24] J. Frank. Weighting for Godot: Learning heuristics for GSAT. In *Proceedings of the AAAI* , pages 338 - 343, 1996.

[25] P. Hansen and B. Jaumand, Algorithms for the Maximum Satisfiability Problem, *Computing*, 44:279-303, 1990.

[26] I. Gent and T. Walsh, Unsatisfied Variables in Local Search. In *Hybrid Problems, Hybrid Solutions*, IOS Press, 73-85, 1995.

[27] I. Gent and T. Walsh, Towards an Understanding of Hill-Climbing Procedures for SAT, *Proc. of AAAI'93*, MIT Press, 28-33, 1993.

[28] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/benchmarks.htm, Forced Satisfiable CSP and SAT Benchmarks of Model RB, *Beihang University*, downloaded 22.02.2011.

[29] http://www.maxsat.udl.cat/09/index.php?disp=submitted-benchmarks, MAX-SAT 2009, *The Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT-2009),* downloaded 22.02.2011.

[30] Holger H. Hoos and Thomas Stützle, *SATLIB: An Online Resource for Research on SAT.* In: I.P. Gent, H.v.Maaren, T. Walsh, editors, SAT 2000, pp.283-292, IOS Press, 2000. SATLIB is available online at *www.satlib.org*.

Appendices

Appendix A Source Code

Multilevel Paradigm - Clustering and Initial Solution Phases

```
std::cout << "Running Multilevel clustering....." << std::endl << std::endl;

std::cout << "Clustering literals randomly....." << std::endl << std::endl;

const int NUMBER_OF_LITERALS = atoi(numberOfXLiterals.c_str());
const int NUMBER_OF_CLAUSES = atoi(numberOfClauses.c_str());
TimeElapsed te;
double timeElapsedMultilevelClustering = 0;
clock_t end = 0;
std::string satisfaction = "";
int trueClauseCounter = 0;
bool literalValues[2] = {true, false};
int entranceCounter = 0;
int entranceCounter2 = 0;
std::string cluster;
std::vector<int> vectorVariable;
int randomVariableOne = 0;
int oppositeValue = 0;
int levelIndex = 0;
std::vector<std::string> vectorCheckedClauses;

//Iterator vector, to help find elements
std::vector<std::string>::iterator iteratorVector;

//Iterator map, to be able to find stuff
std::map<std::string, bool>::iterator iteratorMap;

//Iterator vector, to help find elements
std::vector<std::string>::iterator iteratorVectorString;

bool firstEntrance = true;

while(firstEntrance || clusterCollection.size() >= atoi(numberOfXLiterals.c_str()) *
0.10)
{
	firstEntrance = false;

	if(entranceCounter == 0)
	{
		entranceCounter++;
		std::cout << "LEVEL " << levelCounter << std::endl << std::endl;

		for(int i = 1; i <= NUMBER_OF_LITERALS; i++)
		{
			vectorVariable.push_back(i);

			//int -> string
			char sizeTemp = (char)i;
			char bufferTemp[sizeof(sizeTemp)/sizeof(char) + 10];
			std::string tempLiteral = itoa(i, bufferTemp, 10);

			//int -> string
			char sizeTemp2 = (char)levelCounter;
```

```
			char bufferTemp2[sizeof(sizeTemp2)/sizeof(char) + 10];
			std::string stringLevelCounter = itoa(levelCounter, bufferTemp2, 10);

			//int -> string
			char size3 = (char)levelIndex;
			char buffer3[sizeof(size3)/sizeof(char) + 10];
			std::string stringLevelIndex = itoa(levelIndex, buffer3, 10);

			//Update the map level with literals
			mapLevelClusters[stringLevelCounter + " " + stringLevelIndex] = 
tempLiteral;

			//Increment level index
			levelIndex++;
		}

		/**Stop timer**/
		end = clock();

		timeElapsedMultilevelClustering = te.GetTimeElapsed(end, begin)/1000;

		//Level 0 complete
		std::cout << "Time used: " << timeElapsedMultilevelClustering << " seconds" 
<< std::endl << std::endl;

		//Reset
		levelIndex = 0;

		levelCounter++;
		std::cout << "LEVEL " << levelCounter << std::endl << std::endl;

		//Cluster and create LEVEL 1 (Initial cluster collection)
		while(vectorVariable.size() != 0)
		{
			if(vectorVariable.size() == 1)
				break;

			//Random shuffle
			std::random_shuffle(vectorVariable.begin(), vectorVariable.end());

			char sizeVariableOne = (char)vectorVariable[0];
			char bufferVariableOne[sizeof(sizeVariableOne)/sizeof(char) + 10];
			cluster += itoa(vectorVariable[0], bufferVariableOne, 10);
			cluster += " ";

			char sizeVariableTwo = (char)vectorVariable[1];
			char bufferVariableTwo[sizeof(sizeVariableTwo)/sizeof(char) + 10];
			cluster += itoa(vectorVariable[1], bufferVariableTwo, 10);
			cluster += " ";

			while(cluster[0] == ' ')
				cluster.erase(cluster.begin());

			while(cluster[cluster.length() - 1] == ' ')
				cluster.erase(cluster.end() - 1);

			initialClusterCollection.push_back(cluster);

			//Erase
			vectorVariable.erase(vectorVariable.begin());
			vectorVariable.erase(vectorVariable.begin());

			//int -> string
```

```
			char sizeTemp2 = (char)levelCounter;
			char bufferTemp2[sizeof(sizeTemp2)/sizeof(char) + 10];
			std::string stringLevelCounter = itoa(levelCounter, bufferTemp2, 10);

			//int -> string
			char size3 = (char)levelIndex;
			char buffer3[sizeof(size3)/sizeof(char) + 10];
			std::string stringLevelIndex = itoa(levelIndex, buffer3, 10);

			//Update the map level with cluster
			mapLevelClusters[stringLevelCounter + " " + stringLevelIndex] = 
cluster;

			//Increment level index
			levelIndex++;

			//Reset
			cluster = "";
		}

		//Push the rest into initialClusterCollection
		if(vectorVariable.size() == 1)
		{
			char sizeVariableOne = (char)vectorVariable[0];
			char bufferVariableOne[sizeof(sizeVariableOne)/sizeof(char) + 10];
			cluster = itoa(vectorVariable[0], bufferVariableOne, 10);

			while(cluster[0] == ' ')
				cluster.erase(cluster.begin());

			while(cluster[cluster.length() - 1] == ' ')
				cluster.erase(cluster.end() - 1);

			initialClusterCollection.push_back(cluster);

			//Erase
			vectorVariable.erase(vectorVariable.begin());

			//int -> string
			char sizeTemp2 = (char)levelCounter;
			char bufferTemp2[sizeof(sizeTemp2)/sizeof(char) + 10];
			std::string stringLevelCounter = itoa(levelCounter, bufferTemp2, 10);

			//int -> string
			char size3 = (char)levelIndex;
			char buffer3[sizeof(size3)/sizeof(char) + 10];
			std::string stringLevelIndex = itoa(levelIndex, buffer3, 10);

			//Update the map level with cluster
			mapLevelClusters[stringLevelCounter + " " + stringLevelIndex] = 
cluster;

			//Increment level index
			levelIndex++;
		}

		/**Stop timer**/
		end = clock();

		timeElapsedMultilevelClustering = te.GetTimeElapsed(end, begin)/1000;

		//Level 1 complete
		std::cout << "Time used: " << timeElapsedMultilevelClustering << " seconds"
```

```
<< std::endl << std::endl;
	}

	//Clear cluster
	cluster = "";

	std::string trueClusters = "";
	std::string falseClusters = "";
	std::string substring = " ";

	int clusterCounter = 0;

	//Reset level index
	levelIndex = 0;

	//int -> string
	char sizeTemp = (char)levelCounter;
	char bufferTemp[sizeof(sizeTemp)/sizeof(char) + 10];
	std::string stringLevelCounter = itoa(levelCounter, bufferTemp, 10);

	if(entranceCounter2 == 0)
	{
		entranceCounter2++;

		levelCounter++;
		std::cout << "LEVEL " << levelCounter << std::endl << std::endl;

		//int -> string
		char sizeTemp = (char)levelCounter;
		char bufferTemp[sizeof(sizeTemp)/sizeof(char) + 10];
		std::string stringLevelCounter = itoa(levelCounter, bufferTemp, 10);

		//Cluster and create the next level
		while(initialClusterCollection.size() != 0)
		{
			if(initialClusterCollection.size() == 1)
				break;

			//Random shuffle
			std::random_shuffle(initialClusterCollection.begin(),
initialClusterCollection.end());

			trueClusters += initialClusterCollection[0];
			trueClusters += " ";
			trueClusters += initialClusterCollection[1];
			trueClusters += " ";

			while(trueClusters[0] == ' ')
				trueClusters.erase(trueClusters.begin());

			while(trueClusters[trueClusters.length() - 1] == ' ')
				trueClusters.erase(trueClusters.end() - 1);

			clusterCollection.push_back(trueClusters);

			//Erase
			initialClusterCollection.erase(initialClusterCollection.begin());
			initialClusterCollection.erase(initialClusterCollection.begin());

			//int -> string
			char sizeTemp = (char)levelIndex;
			char bufferTemp[sizeof(sizeTemp)/sizeof(char) + 10];
			std::string stringLevelIndex = itoa(levelIndex, bufferTemp, 10);
```

```
			//Update the map level with cluster
			mapLevelClusters[stringLevelCounter + " " + stringLevelIndex] = 
trueClusters;

			//Increase level index
			levelIndex++;

			trueClusters = "";
		}

		//Push the rest into cluster collection
		if(initialClusterCollection.size() == 1)
		{
			trueClusters = initialClusterCollection[0];

			while(trueClusters[0] == ' ')
				trueClusters.erase(trueClusters.begin());

			while(trueClusters[trueClusters.length() - 1] == ' ')
				trueClusters.erase(trueClusters.end() - 1);

			clusterCollection.push_back(trueClusters);

			//Erase
			initialClusterCollection.erase(initialClusterCollection.begin());

			//int -> string
			char sizeTemp = (char)levelIndex;
			char bufferTemp[sizeof(sizeTemp)/sizeof(char) + 10];
			std::string stringLevelIndex = itoa(levelIndex, bufferTemp, 10);

			//Update the map level with TRUE cluster
			mapLevelClusters[stringLevelCounter + " " + stringLevelIndex] = 
trueClusters;

			//Increase level index
			levelIndex++;

			clusterCounter = 0;
			trueClusters = "";
		}

		//Reset cluster counter
		clusterCounter = 0;

		/**Stop timer**/
		end = clock();

		timeElapsedMultilevelClustering = te.GetTimeElapsed(end, begin)/1000;

		//Level 2 complete
		std::cout << "Time used: " << timeElapsedMultilevelClustering << " seconds" 
<< std::endl << std::endl;
	}

	levelCounter++;
	std::cout << "LEVEL " << levelCounter << std::endl << std::endl;

	//Reset level index
	levelIndex = 0;

	//int -> string
```

```
	char sizeLevelCounter = (char)levelCounter;
	char bufferLevelCounter[sizeof(sizeLevelCounter)/sizeof(char) + 10];
	std::string stringUpdatedLevelCounter = itoa(levelCounter, bufferTemp, 10);

	//Cluster and create the next level
	while(clusterCollection.size() != 0)
	{
		if(clusterCollection.size() == 1)
			break;

		//Random shuffle
		std::random_shuffle(clusterCollection.begin(), clusterCollection.end());

		trueClusters += clusterCollection[0];
		trueClusters += " ";
		trueClusters += clusterCollection[1];
		trueClusters += " ";

		substring += clusterCollection[0];
		substring += " ";
		substring += clusterCollection[1];
		substring += " ";

		while(trueClusters[0] == ' ')
			trueClusters.erase(trueClusters.begin());

		while(trueClusters[trueClusters.length() - 1] == ' ')
			trueClusters.erase(trueClusters.end() - 1);

		//Erase
		clusterCollection.erase(clusterCollection.begin());
		clusterCollection.erase(clusterCollection.begin());

		//int -> string
		char sizeTemp = (char)levelIndex;
		char bufferTemp[sizeof(sizeTemp)/sizeof(char) + 10];
		std::string stringLevelIndex = itoa(levelIndex, bufferTemp, 10);

		//Update the map level with TRUE cluster
		mapLevelClusters[stringUpdatedLevelCounter + " " + stringLevelIndex] =
trueClusters;

		//Increase level index
		levelIndex++;

		substring += "|";
		trueClusters = "";
	}

	if(clusterCollection.size() == 1)
	{
		trueClusters = clusterCollection[0];

		substring += clusterCollection[0];

		while(trueClusters[0] == ' ')
			trueClusters.erase(trueClusters.begin());

		while(trueClusters[trueClusters.length() - 1] == ' ')
			trueClusters.erase(trueClusters.end() - 1);

		//Erase
		clusterCollection.erase(clusterCollection.begin());
```

```
            //int -> string
            char sizeTemp = (char)levelIndex;
            char bufferTemp[sizeof(sizeTemp)/sizeof(char) + 10];
            std::string stringLevelIndex = itoa(levelIndex, bufferTemp, 10);

            //Update the map level with TRUE cluster
            mapLevelClusters[stringUpdatedLevelCounter + " " + stringLevelIndex] =
trueClusters;

            //Increase level index
            levelIndex++;

            substring += "|";
            trueClusters = "";
        }

        //Temp cluster
        std::string tempCluster = "";

        //Reset cluster collection
        clusterCollection.clear();

        //Update cluster collection
        for(int i = 0; i < substring.length(); i++)
        {
            if(substring[i] != '|')
                tempCluster += substring[i];

            else if(substring[i] == '|')
            {
                while(tempCluster[0] == ' ')
                    tempCluster.erase(tempCluster.begin());

                while(tempCluster[tempCluster.length() - 1] == ' ')
                    tempCluster.erase(tempCluster.end() - 1);

                clusterCollection.push_back(tempCluster);
                tempCluster = "";
            }
        }


        //Clear substring
        substring = "";

        //Clear tempCluster
        tempCluster = "";

        //Reset cluster counter
        clusterCounter = 0;

        /**Stop timer**/
        end = clock();

        timeElapsedMultilevelClustering = te.GetTimeElapsed(end, begin)/1000;

        //Level X complete
        std::cout << "Time used: " << timeElapsedMultilevelClustering << " seconds" <<
std::endl << std::endl;

        //We have reached the final level
        if(clusterCollection.size() <= atoi(numberOfXLiterals.c_str()) * 0.10)
```

```
	{
		/**Tabu Search Initialization Phase**/

		std::cout << "Assigning the clusters TRUE/FALSE values randomly....." << 
std::endl << std::endl;

		std::string tempCluster = "";
		std::string tempLiteral = "";
		int randomIndex = 0;
		int levelIndex = 0;		//Cluster index
		int lowerLevelIndex = 0; //Literal index

		//Assign the clusters random TRUE/FALSE values as well as the literals 
within the clusters
		for(int j = 0; j < clusterCollection.size(); j++)
		{
			randomIndex = rand () % 2;
			tempCluster = clusterCollection[j];

			while(tempCluster[0] == ' ')
				tempCluster.erase(tempCluster.begin());

			while(tempCluster[tempCluster.length() - 1] == ' ')
				tempCluster.erase(tempCluster.end() - 1);

			mapLiteralValues[tempCluster] = literalValues[randomIndex];

			//If FALSE, all literals inside must be FALSE
			if(mapLiteralValues[tempCluster] == false)
			{
				for(int k = 0; k < tempCluster.length(); k++)
				{
					if(tempCluster[k] != ' ')
						tempLiteral += tempCluster[k];

					if(tempCluster[k] == ' ' || k == tempCluster.length() - 
1)
					{
						//FALSE literal
						mapLiteralValues[tempLiteral] = 
literalValues[randomIndex];

						//Opposite value will be TRUE
						int temp = atoi(tempLiteral.c_str()) * -1;
						char sizeTempOpposite = (char)temp;
						char 
bufferTempOpposite[sizeof(sizeTempOpposite)/sizeof(char) + 10];
						std::string oppositeTempLiteral = itoa(temp, 
bufferTempOpposite, 10);

						mapLiteralValues[oppositeTempLiteral] = 
literalValues[randomIndex - 1];
						tempLiteral = "";
					}
				}
			}

			//If TRUE, all literals inside must be TRUE
			else if(mapLiteralValues[tempCluster] == true)
			{
				for(int k = 0; k < tempCluster.length(); k++)
				{
					if(tempCluster[k] != ' ')
```

```
                                        tempLiteral += tempCluster[k];

                                if(tempCluster[k] == ' ' || k == tempCluster.length() -
1)
                                {
                                        //TRUE literal
                                        mapLiteralValues[tempLiteral] =
literalValues[randomIndex];

                                        //Opposite value will be FALSE
                                        int temp = atoi(tempLiteral.c_str()) * -1;
                                        char sizeTempOpposite = (char)temp;
                                        char
bufferTempOpposite[sizeof(sizeTempOpposite)/sizeof(char) + 10];
                                        std::string oppositeTempLiteral = itoa(temp,
bufferTempOpposite, 10);

                                        mapLiteralValues[oppositeTempLiteral] =
literalValues[randomIndex + 1];
                                        tempLiteral = "";
                                }
                        }
                }
        }

    /**Learning Automata Initialization Phase**

    //Assign the clusters random state values as well as the literals within the
clusters
    for(int j = 0; j < clusterCollection.size(); j++)
    {
    randomIndex = rand () % 2;
    tempCluster = clusterCollection[j];

    while(tempCluster[0] == ' ')
            tempCluster.erase(tempCluster.begin());

    while(tempCluster[tempCluster.length() - 1] == ' ')
            tempCluster.erase(tempCluster.end() - 1);

    mapClustersStates[tempCluster] = states[randomIndex];

    if(states[randomIndex] == -1)
            mapLiteralValues[tempCluster] = false;
    else
            mapLiteralValues[tempCluster] = true;

    //If FALSE, all literals inside must be FALSE
    if(mapLiteralValues[tempCluster] == false)
    {
            for(int k = 0; k < tempCluster.length(); k++)
            {
                    if(tempCluster[k] != ' ')
                            tempLiteral += tempCluster[k];

                    if(tempCluster[k] == ' ' || k == tempCluster.length() - 1)
                    {
                            //FALSE literal
                            mapClustersStates[tempLiteral] = -1;
                            mapLiteralValues[tempLiteral] = false;

                            //Opposite value will be TRUE
                            int temp = atoi(tempLiteral.c_str()) * -1;
```

```
                            char sizeTempOpposite = (char)temp;
                            char bufferTempOpposite[sizeof(sizeTempOpposite)/sizeof(char)
+ 10];
                            std::string oppositeTempLiteral = itoa(temp,
bufferTempOpposite, 10);

                            mapClustersStates[oppositeTempLiteral] = 1;
                            mapLiteralValues[oppositeTempLiteral] = true;
                            tempLiteral = "";
                     }
              }
       }

       //If TRUE, all literals inside must be TRUE
       else if(mapLiteralValues[tempCluster] == true)
       {
              for(int k = 0; k < tempCluster.length(); k++)
              {
                     if(tempCluster[k] != ' ')
                            tempLiteral += tempCluster[k];

                     if(tempCluster[k] == ' ' || k == tempCluster.length() - 1)
                     {
                            //TRUE literal
                            mapClustersStates[tempLiteral] = 1;
                            mapLiteralValues[tempLiteral] = true;

                            //Opposite value will be FALSE
                            int temp = atoi(tempLiteral.c_str()) * -1;
                            char sizeTempOpposite = (char)temp;
                            char bufferTempOpposite[sizeof(sizeTempOpposite)/sizeof(char)
+ 10];
                            std::string oppositeTempLiteral = itoa(temp,
bufferTempOpposite, 10);

                            mapClustersStates[oppositeTempLiteral] = -1;
                            mapLiteralValues[oppositeTempLiteral] = false;
                            tempLiteral = "";
                     }
              }
       }
       }**/
              //Assign TRUE/FALSE values to the SAT formula
              std::cout << "Assigning TRUE/FALSE values to the clauses of the SAT
formula....." << std::endl << std::endl;

              int trueCounter = 0;
              std::string stringClause = "";

              for(int j = 0; j < vectorStringNumbers.size(); j++)
              {
                     if(vectorStringNumbers[j] != "|")
                     {
                            stringClause += vectorStringNumbers[j];
                            stringClause += " ";

                            if(mapLiteralValues[vectorStringNumbers[j]] == true)
                                   trueCounter++;
                     }

                     else if(vectorStringNumbers[j] == "|")
                     {
                            //Erase the space at the end of string
```

```
                            stringClause.erase(stringClause.end() - 1);

                            iteratorVectorString = std::find(vectorCheckedClauses.begin(),
vectorCheckedClauses.end(), stringClause);

                            if(iteratorVectorString == vectorCheckedClauses.end())
                            {
                                   vectorCheckedClauses.push_back(stringClause);

                                   if(trueCounter >= 1)
                                   {
                                          //Set the clause to TRUE
                                          mapClauseValues.insert(std::pair<std::string,
bool>(stringClause, true));
                                          trueClauseCounter++;
                                   }

                                   else if(trueCounter == 0)
                                   {
                                          //Set the clause to FALSE
                                          mapClauseValues.insert(std::pair<std::string,
bool>(stringClause, false));
                                   }
                            }

                            //Reset
                            stringClause = "";
                            trueCounter = 0;
                     }
              }
       }
}

bestSoFar = trueClauseCounter;

//Reset
trueClauseCounter = 0;
vectorCheckedClauses.clear();

foutput << "Literals: " << numberOfXLiterals << "  Clauses: " << NUMBER_OF_CLAUSES <<
"\n\n";
std::cout << "Number of TRUE clauses (initial solution): " << std::endl << std::endl <<
bestSoFar << std::endl << std::endl;
foutput << "Level     Satisfied clauses     Time     Flips\n\n";
foutput << levelCounter << "                         " << bestSoFar << "
" << 0 << "                        " << 0 << "\n";

if(bestSoFar == NUMBER_OF_CLAUSES)
{
       satisfaction = "SATISFIED";
       std::cout << "SAT with " << numberOfXLiterals << " literals and " <<
NUMBER_OF_CLAUSES << " clauses is " << satisfaction << " at LEVEL " << levelCounter <<
"." << std::endl << std::endl;
}
else
{
       satisfaction = "NOT SATISFIED";
       std::cout << "SAT with " << numberOfXLiterals << " literals and " <<
NUMBER_OF_CLAUSES << " clauses is " << satisfaction << " at LEVEL " << levelCounter <<
"." << std::endl << std::endl;
}

//Finished!
```

```
if(satisfaction == "SATISFIED")
{
	system("pause");
	system("exit");
}
```

Tabu Search All Versions

The code can be downloaded online at:

https://ikt590-sat-tabu-search.googlecode.com/svn/trunk

Multilevel Tabu Search

The code can be downloaded online at:

https://ikt590-sat-multilevel-tabu-search.googlecode.com/svn/trunk

Learning Automata with Tabu Search

The code can be downloaded online at:

https://ikt590-sat-learning-automata.googlecode.com/svn/trunk/

Multilevel Learning Automata with Tabu Search

The code can be downloaded online at:

https://ikt590-sat-multilevel-learning-automata.googlecode.com/svn/trunk/

You can also refer to Appendix C for the source code on CD attached with the thesis report.

Appendix B Experimental Results Data

Tabu Search solving bw_large.d (BlocksWorld)

Problem: bw_large.d
Literals: 6325 Clauses: 131973

Mean solved: 81.6 % Variance: 0.5 Standard deviation: 0.71

Mean satisfied clauses	Mean time	Mean flips
102984	0	0
102984	0	10
102984	0	100
102984	0	1000
103015	0	5014
103015	11.763	10000
103045	11.763	10032
103075	24.062	15033
103104	37.196	20057
103132	50.413	25045
103160	63.785	30049
103188	76.984	35026
103216	90.147	40033
103243	103.348	45022
103270	116.473	50000
103297	129.837	54968
103324	142.918	59982
103351	156.133	64941
103378	169.185	69912
103405	182.248	74890
103432	195.588	79847
103458	208.613	84824
103484	221.698	89787
103510	234.744	94772
103536	247.845	99744
103536	261.167	100000
103562	261.167	104702
103588	274.213	109661
103614	287.263	114642
103639	300.347	119611
103664	313.382	124577
103689	326.661	129505
103714	339.59	134450
103739	352.571	139357
103764	365.461	144295
103789	378.431	149238
103814	391.713	154186
103839	404.754	159116
103864	417.769	164073
103889	430.793	168996
103914	443.745	173942
103939	456.968	178876
103964	469.931	183815
103989	482.908	188757
104014	495.901	193646

104039	508.749	198587
104063	521.959	203513
104087	534.904	208455
104111	547.892	213375
104135	561.02	218308
104159	574.112	223250
104183	587.335	228172
104209	600.271	233042
104233	613.065	237956
104257	625.965	242847
104281	638.806	247745
104305	651.831	252651
104329	664.834	257533
104353	677.705	262447
104377	690.645	267341
104401	703.526	272249
104425	716.541	277131
104449	729.528	282036
104473	742.413	286949
104496	755.358	291827
104519	768.178	296775
104542	781.168	301642
104565	794.272	306532
104588	807.153	311425
104611	820.011	316265
104634	832.736	321143
104657	845.576	326028
104680	858.632	330908
104703	871.443	335803
104726	884.293	340671
104749	897.077	345569
104772	909.927	350421
104795	922.89	355275
104818	935.628	360160
104841	948.472	365048
104864	961.31	369927
104887	974.123	374805
104910	987.154	379665
104933	999.913	384546
104956	1012.76	389399
104979	1025.53	394247
105002	1038.27	399132
105025	1051.29	403979
105048	1064.05	408828
105071	1076.78	413699
105094	1089.75	418532
105117	1102.61	423376
105140	1115.48	428191
105163	1128.25	433049
105186	1141	437861
105209	1153.65	442695
105232	1166.36	447526
105255	1179.11	452332
105277	1191.99	457167
105299	1204.95	462004
105321	1218.29	466843

105343	1232.41	471690
105365	1246.91	476537
105387	1261.43	481373
105409	1275.88	486201
105431	1290.31	491055
105453	1304.81	495876
105475	1319.34	500707
105497	1333.77	505520
105519	1348.17	510345
105541	1362.52	515168
105563	1376.77	519964
105585	1391.24	524783
105607	1405.32	529582
105629	1419.68	534412
105651	1434.14	539229
105673	1448.65	544019
105695	1463.02	548827
105717	1477.4	553641
105739	1491.82	558435
105761	1506.05	563232
105783	1520.87	568033
105805	1535.03	572810
105827	1549.11	577597
105849	1563.14	582389
105871	1577.22	587185
105893	1591.39	591974
105915	1605.12	596785
105937	1618.86	601575
105959	1632.62	606360
105981	1646.69	611123
106003	1660.72	615915
106025	1674.76	620678
106047	1688.89	625430
106069	1701.42	630188
106091	1715.63	634949
106113	1729.1	639699
106135	1741.97	644441
106156	1754.49	649194
106177	1766.97	653953
106198	1779.63	658748
106219	1792.24	663547
106240	1804.75	668288
106261	1817.17	673044
106282	1829.59	677818
106303	1842.04	682528
106324	1854.57	687283
106345	1866.98	692029
106366	1879.38	696794
106387	1891.8	701546
106408	1904.23	706302
106429	1917.04	711040
106450	1929.38	715776
106471	1941.76	720551
106492	1954.25	725293
106513	1966.63	730036
106534	1979.2	734775

106555	1993.03	739510
106576	2007.04	744262
106597	2021.07	748975
106618	2035.09	753700
106639	2049.17	758443
106660	2063.25	763206
106681	2075.91	767956
106702	2088.38	772706
106723	2100.81	777441
106744	2113.41	782210
106765	2125.85	786927
106786	2138.21	791647
106807	2152.2	796405
106828	2164.82	801143
106849	2177.93	805862
106870	2190.88	810552
106891	2203.21	815247
106912	2215.63	819945
106933	2228.1	824639
106954	2240.62	829374
106975	2253.38	834092
106996	2265.82	838819
107018	2278.32	843537
107039	2290.79	848229
107060	2303.1	852927
107081	2315.61	857629
107102	2328.44	862292
107123	2341.87	866977
107144	2354.74	871693
107165	2371.02	876389
107186	2387.97	881097
107207	2402.03	885776
107228	2414.4	890466
107249	2426.97	895146
107270	2439.54	899828
107291	2452.14	904517
107312	2464.43	909224
107332	2477.2	913906
107352	2489.79	918559
107372	2502.15	923236
107392	2514.67	927927
107412	2527.49	932589
107432	2539.96	937252
107452	2554.46	941916
107472	2569.68	946612
107493	2584.97	951302
107513	2600.07	955964
107533	2615.14	960636
107553	2630.1	965324
107573	2645.3	969998
107593	2660.34	974650
107613	2675.29	979306
107633	2690.23	983991
107653	2705.29	988674
107673	2720.29	993327
107693	2735.23	997977

107693	2750.16	1e+006
107713	2750.16	1e+006

Total time elapsed: 2832.61 seconds

Multilevel Tabu Search solving bw_large.d (BlocksWorld)

Problem: bw_large.d
Literals: 6325 Clauses: 131973

Mean solved: 89.6 % Variance: 0.4 Standard deviation: 0.63

Level	Mean satisfied clauses	Mean time	Mean flips
4	102379	0	0
4	102379	0	16
4	102379	0	112
4	102379	0	1000
4	102691	0	6325
4	102691	3.82	10005
4	102998	3.82	12634
4	103301	7.437	18943
4	103596	11.118	25252
4	103890	14.78	31561
4	104181	18.433	37870
4	104467	22.084	44179
4	104750	25.75	50488
4	105031	29.456	56797
4	105311	33.193	63106
4	105590	36.849	69415
4	105864	40.519	75724
4	106137	44.199	82033
4	106405	47.88	88342
4	106671	51.558	94651
4	106671	55.243	100000
4	106935	55.243	100960
4	107198	58.92	107269
4	107457	62.593	113578
4	107713	66.252	119887
4	107969	69.915	126196
4	108223	73.579	132505
4	108476	77.233	138814
4	108729	80.91	145123
4	108977	84.567	151432
4	109225	88.237	157741
4	109471	91.891	164050
4	109716	95.602	170359
4	109959	99.33	176668
4	110203	102.961	182977
4	110443	106.586	189286
4	110682	110.225	195595
4	110682	113.884	200000
4	110918	113.884	200000
3	110918	116.458	16
3	110918	116.458	104
3	110918	116.458	1007
3	111052	116.458	6325
3	111052	121.235	10004
3	111186	121.235	12642
3	111317	125.629	18959
3	111447	130.022	25276

3	111576	134.446	31593
3	111704	138.905	37910
3	111831	143.146	44228
3	111956	147.384	50545
3	112081	151.64	56862
3	112205	156.071	63179
3	112328	160.55	69496
3	112451	165.04	75813
3	112573	169.458	82130
3	112695	173.862	88447
3	112817	178.261	94764
3	112817	182.651	100002
3	112938	182.651	101081
3	113059	187.036	107398
3	113179	191.42	113715
3	113299	195.818	120032
3	113418	200.201	126349
3	113537	204.582	132666
3	113653	208.972	138983
3	113769	213.365	145300
3	113885	217.756	151617
3	114000	222.157	157934
3	114115	226.554	164251
3	114228	231.051	170568
3	114341	235.471	176885
3	114453	239.865	183202
3	114568	244.303	189519
3	114680	248.702	195836
3	114680	253.16	200003
3	114792	253.16	200003
2	114792	256.141	12
2	114792	256.141	100
2	114792	256.141	1000
2	114859	256.141	6325
2	114859	263.437	10001
2	114925	263.437	12646
2	114989	270.495	18967
2	115053	277.287	25288
2	115116	284.137	31609
2	115178	290.955	37930
2	115241	297.884	44251
2	115303	304.716	50572
2	115364	311.548	56893
2	115424	318.375	63214
2	115484	325.209	69535
2	115544	332.04	75856
2	115604	338.869	82177
2	115664	345.709	88498
2	115724	352.541	94819
2	115724	359.399	100002
2	115783	359.399	101140
2	115841	366.378	107461
2	115899	373.204	113782
2	115957	380.019	120103
2	116015	386.821	126424
2	116073	393.964	132745

2	116130	400.894	139066
2	116187	407.706	145387
2	116244	414.536	151708
2	116301	421.37	158029
2	116359	428.259	164350
2	116416	435.146	170671
2	116473	441.962	176992
2	116529	448.792	183313
2	116585	455.604	189634
2	116641	462.441	195955
2	116641	469.279	200003
2	116697	469.279	200003
1	116697	473.652	10
1	116697	473.652	100
1	116697	473.652	1000
1	116731	473.652	6325
1	116731	485.744	10000
1	116764	485.744	12648
1	116797	497.994	18971
1	116830	510.092	25294
1	116863	522.197	31617
1	116895	534.295	37940
1	116927	546.395	44263
1	116959	558.488	50586
1	116991	570.752	56909
1	117023	583.08	63232
1	117054	595.355	69555
1	117085	607.549	75878
1	117116	619.701	82201
1	117147	631.934	88524
1	117178	644.029	94847
1	117178	656.112	100000
1	117209	656.112	101170
1	117240	668.199	107493
1	117271	680.293	113816
1	117302	692.416	120139
1	117333	704.62	126462
1	117364	716.703	132785
1	117395	728.79	139108
1	117426	740.87	145431
1	117457	752.965	151754
1	117487	765.21	158077
1	117517	777.296	164400
1	117547	789.414	170723
1	117577	801.577	177046
1	117607	813.734	183369
1	117637	825.935	189692
1	117667	838.156	196015
1	117667	849.2	200000
1	117697	849.2	200000
0	117697	855.865	10
0	117697	855.865	100
0	117697	855.865	1000
0	117697	855.865	3987
0	117697	946.399	7970
0	117697	960.195	10000

0	117697	960.195	11961
0	117697	973.952	15917
0	117697	987.516	19871
0	117697	1001.16	23824
0	117697	1014.79	27783
0	117697	1028.78	31716
0	117697	1042.59	35650
0	117697	1056.4	39576
0	117697	1070.18	43460
0	117697	1083.79	47366
0	117697	1097.59	51287
0	117697	1111.26	55176
0	117697	1124.8	59058
0	117697	1138.3	62876
0	117697	1151.71	66746
0	117697	1165.55	70599
0	117711	1179.19	74446
0	117728	1193.23	78230
0	117744	1206.8	82041
0	117760	1220.53	85815
0	117776	1234.33	89569
0	117792	1248.06	93357
0	117808	1261.74	97140
0	117808	1275.22	100000
0	117824	1275.22	100932
0	117840	1288.83	104693
0	117856	1302.52	108457
0	117872	1316.09	112190
0	117888	1329.55	115934
0	117904	1343.03	119658
0	117920	1356.48	123364
0	117936	1369.91	127059
0	117952	1383.23	130728
0	117968	1396.43	134413
0	117984	1409.67	138082
0	118000	1422.9	141722
0	118016	1436.1	145373
0	118032	1449.22	149007
0	118048	1462.27	152635
0	118064	1475.36	156267
0	118080	1488.56	159867
0	118096	1501.63	163465
0	118112	1514.65	167023
0	118128	1527.52	170576
0	118144	1540.38	174131
0	118160	1553.25	177700
0	118176	1566.31	181240
0	118192	1579.11	184766
0	118208	1591.86	188311
0	118224	1604.76	191856
0	118240	1617.61	195331
0	118256	1630.33	198846
0	118256	1643.03	200000
0	118272	1643.03	200000

Total time elapsed: 1727.69 seconds

Learning Automata with Tabu Search solving mot_comb3._red-gate-0.dimacs.seq.filtered (MaxSAT Industry)

Problem: mot_comb3._red-gate-0.dimacs.seq.filtered
Literals: 11265 Clauses: 29520

Mean solved: 79.5 % Variance: 0.7 Standard deviation: 0.84

Mean satisfied clauses	Mean time	Mean flips
22600	0	0
22603	0	1
22603	0	10
22603	0	100
22603	0	1000
22614	0	5381
22614	17.331	10000
22622	17.331	10792
22629	35.92	16214
22636	55.26	21622
22643	74.457	27007
22643	93.322	27008
22650	93.322	32413
22657	112.543	37856
22659	131.454	37857
22666	131.454	43246
22668	150.648	43247
22675	150.648	48620
22681	169.706	53996
22682	188.464	53997
22688	188.464	59374
22694	207.231	64783
22700	226.098	70157
22700	244.839	70158
22706	244.839	75568
22706	263.736	75569
22711	263.736	80931
22716	282.462	86299
22721	301.264	91672
22723	320.065	91673
22728	320.065	97040
22728	338.896	100000
22733	338.896	102400
22738	357.676	107785
22743	376.673	113153
22743	396.15	113154
22748	396.15	118520
22753	415.145	123861
22753	434.033	123862
22758	434.033	129224
22763	453.047	134554
22768	471.976	139929
22773	491.074	145315
22778	510.226	150634
22780	528.93	150635
22785	528.93	155970

22790	549.136	161293
22795	568.245	166663
22796	587.872	166664
22800	587.872	172013
22804	606.927	177350
22808	625.855	182689
22808	644.995	182690
22812	644.995	188013
22816	664.046	193364
22820	682.602	198713
22824	701.858	204039
22828	720.703	209429
22832	739.464	214764
22836	758.507	220035
22837	777.251	220036
22841	777.251	225366
22845	796.27	230657
22849	814.944	235996
22852	833.77	235997
22856	833.77	241339
22860	852.593	246606
22864	871.174	251942
22865	889.973	251943
22869	889.973	257247
22873	908.641	262549
22875	927.261	262550
22879	927.261	267818
22881	945.892	267819
22885	945.892	273111
22889	964.807	278422
22893	983.769	283735
22893	1002.69	283736
22897	1002.69	289050
22897	1021.43	289051
22901	1021.43	294396
22905	1040.27	294397
22909	1040.27	299647
22913	1058.77	304964
22917	1077.54	310258
22921	1096.21	315606
22923	1115.02	315607
22927	1115.02	320889
22927	1133.61	320890
22930	1133.61	326188
22934	1152.24	326189
22938	1152.24	331446
22942	1170.73	336706
22942	1189.23	336707
22947	1189.23	341987
22949	1207.87	341988
22953	1207.87	347249
22957	1226.39	352535
22961	1245.09	357802
22965	1263.68	363065
22968	1282.38	363066
22972	1282.38	368306

22976	1300.9	368307
22980	1300.9	373570
22982	1319.48	373571
22986	1319.48	378792
22990	1338.01	384065
22990	1356.6	384066
22994	1356.6	389268
22997	1375.65	389269
23001	1375.65	394501
23004	1393.07	394502
23008	1393.07	399729
23012	1408.98	404978
23012	1425.13	404979
23016	1425.13	410205
23018	1439.04	410206
23022	1439.04	415459
23025	1452.47	415460
23029	1452.47	420663
23029	1465.6	420664
23033	1465.6	425872
23037	1478.5	431088
23041	1491.72	436260
23043	1504.56	436261
23047	1504.56	441477
23047	1517.41	441478
23051	1517.41	446702
23053	1530.28	446703
23057	1530.28	451911
23061	1543.08	457125
23064	1555.89	457126
23068	1555.89	462342
23072	1568.71	467507
23076	1581.41	472716
23080	1594.22	477911
23082	1605.94	477912
23086	1605.94	483104
23090	1613.21	488291
23090	1620.38	488292
23094	1620.38	493456
23098	1627.58	498609
23102	1634.76	503846
23106	1642.07	509008
23106	1649.27	509009
23110	1649.27	514223
23114	1656.56	519444
23114	1663.84	519445
23118	1663.84	524572
23122	1673.4	529712
23126	1686.53	534845
23130	1700.07	540042
23132	1713.45	540043
23136	1713.45	545218
23140	1726.24	550384
23143	1739.92	550385
23147	1739.92	555535
23151	1753.08	560664

23151	1765.85	560665
23155	1765.85	565808
23155	1778.37	565809
23159	1778.37	570976
23159	1791.15	570977
23163	1791.15	576126
23165	1803.84	576127
23169	1803.84	581287
23173	1816.62	586404
23174	1829.41	586405
23178	1829.41	591569
23182	1842.17	596722
23185	1854.75	596723
23189	1854.75	601879
23193	1867.29	607046
23193	1879.84	607047
23197	1879.84	612177
23198	1892.3	612178
23202	1892.3	617327
23202	1904.81	617328
23205	1904.81	622438
23208	1917.23	622439
23212	1917.23	627589
23216	1929.75	632691
23219	1942.15	632692
23223	1942.15	637803
23227	1954.56	642911
23231	1966.97	648042
23235	1979.42	653184
23235	1991.93	653185
23238	1991.93	658280
23242	2004.31	663403
23246	2016.76	668481
23249	2029.11	668482
23253	2029.11	673593
23257	2041.53	678713
23257	2053.97	678714
23261	2053.97	683805
23264	2066.34	683806
23268	2066.34	688882
23272	2078.67	693981
23276	2091.05	699079
23278	2103.43	699080
23281	2103.43	704203
23284	2115.88	704204
23287	2115.88	709284
23290	2128.23	714354
23291	2140.56	714355
23294	2140.56	719410
23297	2152.85	724493
23297	2165.22	724494
23300	2165.22	729577
23303	2177.57	734680
23303	2189.99	734681
23306	2189.99	739764
23309	2202.35	744806

23311	2214.62	744807
23314	2214.62	749913
23314	2227.05	749914
23315	2227.05	754989
23318	2239.38	760055
23321	2251.7	765118
23324	2264	770178
23327	2276.28	775235
23330	2288.57	780300
23330	2300.86	780301
23333	2300.86	785364
23336	2313.16	790406
23339	2325.42	795443
23339	2337.67	795444
23342	2337.67	800519
23345	2350.02	805529
23346	2362.19	805530
23349	2362.19	810588
23352	2374.48	810589
23355	2374.48	815627
23357	2386.73	815628
23360	2386.73	820681
23363	2399.02	825708
23363	2411.24	825709
23366	2411.24	830743
23369	2423.49	835805
23372	2435.79	840825
23373	2448.04	840826
23377	2448.04	845893
23377	2460.37	845894
23378	2460.37	850926
23381	2472.6	855960
23384	2484.82	861018
23387	2497.1	866028
23388	2509.27	866029
23392	2509.27	871040
23394	2521.44	871041
23397	2521.44	876057
23397	2533.64	876058
23399	2533.64	881066
23402	2545.82	886081
23405	2558	891090
23405	2570.17	891091
23408	2570.17	896097
23411	2582.35	901073
23412	2594.44	901074
23415	2594.44	906109
23415	2606.69	906110
23418	2606.69	911136
23419	2618.89	911137
23422	2618.89	916162
23424	2631.1	916163
23427	2631.1	921190
23427	2643.33	921191
23430	2643.33	926170
23433	2655.45	931172

23433	2667.6	931173
23436	2667.6	936194
23437	2679.8	936195
23441	2679.8	941226
23444	2692.01	946223
23446	2704.18	946224
23449	2704.18	951199
23449	2716.26	951200
23452	2716.26	956183
23455	2728.37	961151
23455	2740.45	961152
23458	2740.45	966118
23458	2752.51	966119
23460	2752.51	971100
23463	2764.61	976084
23463	2776.72	976085
23466	2776.72	981090
23468	2788.87	981091
23471	2788.87	986103
23474	2801.3	991070
23477	2814.28	996042
23477	2826.81	1e+006
23480	2826.81	1e+006

Total time elapsed: 2844.21 seconds

Multilevel Learning Automata with Tabu Search solving mot_comb3._red-gate-0.dimacs.seq.filtered (MaxSAT Industry)

Problem: mot_comb3._red-gate-0.dimacs.seq.filtered
Literals: 11265 Clauses: 29520

Mean solved: 79.8 % Variance: 0.6 Standard deviation: 0.77

Level	Mean satisfied clauses	Mean time	Mean flips
4	22418	0	0
4	22418	0	1
4	22421	0	2
4	22421	0	3
4	22421	0	19
4	22421	0	115
4	22421	0	1011
4	22421	0	10004
4	22451	0	11268
4	22477	3.153	22533
4	22479	6.039	22534
4	22479	6.039	22535
4	22503	6.039	33800
4	22503	8.966	33801
4	22503	8.966	33802
4	22504	8.966	33803
4	22508	8.966	33804
4	22508	8.966	33805
4	22531	8.966	45070
4	22531	12.088	45071
4	22531	12.088	45072
4	22531	12.088	45073
4	22532	12.088	45074
4	22533	12.088	45075
4	22533	12.088	45076
4	22535	12.088	45077
4	22557	12.088	56342
4	22560	15.188	56343
4	22560	15.188	56344
4	22560	15.188	56345
4	22560	15.188	56346
4	22561	15.188	56347
4	22566	15.188	56348
4	22588	15.188	67613
4	22588	18.264	67614
4	22588	18.264	67615
4	22588	18.264	67616
4	22590	18.264	67617
4	22590	18.264	67618
4	22611	18.264	78883
4	22612	21.29	78884
4	22614	21.29	78885
4	22616	21.29	78886
4	22616	21.29	78887
4	22619	21.29	78888
4	22622	21.29	78889

4	22622	21.29	78890
4	22623	21.29	78891
4	22626	21.29	78892
4	22629	21.29	78893
4	22629	21.29	78894
4	22629	21.29	78895
4	22646	21.29	90160
4	22646	24.422	90161
4	22649	24.422	90162
4	22649	24.422	90163
4	22651	24.422	90164
4	22654	24.422	90165
4	22654	24.422	100011
4	22674	24.422	101430
4	22676	27.41	101431
4	22677	27.41	101432
4	22679	27.41	101433
4	22680	27.41	101434
4	22682	27.41	101435
4	22684	27.41	101436
4	22686	27.41	101437
4	22689	27.41	101438
4	22709	27.41	112703
4	22710	30.397	112704
4	22713	30.397	112705
4	22715	30.397	112706
4	22715	30.397	112707
4	22734	30.397	123972
4	22737	33.346	123973
4	22756	33.346	135238
4	22757	36.37	135239
4	22759	36.37	135240
4	22760	36.37	135241
4	22760	36.37	135242
4	22776	36.37	146507
4	22780	39.414	146508
4	22780	39.414	146509
4	22782	39.414	146510
4	22800	39.414	157775
4	22803	42.445	157776
4	22821	42.445	169041
4	22839	45.424	180306
4	22839	48.379	180307
4	22839	48.379	180308
4	22839	48.379	180309
4	22839	48.379	180310
4	22839	48.379	180311
4	22839	48.379	180312
4	22844	48.379	191577
4	22844	51.34	191578
4	22844	51.34	191579
4	22844	51.34	200012
4	22860	51.34	200012
3	22860	53.546	16
3	22860	53.546	104
3	22860	53.546	1000

3	22860	53.546	10005
3	22874	53.546	11265
3	22888	58.838	22530
3	22891	64.258	22531
3	22905	64.258	33796
3	22907	69.636	33797
3	22907	69.636	33798
3	22910	69.636	33799
3	22910	69.636	33800
3	22910	69.636	33801
3	22911	69.636	33802
3	22925	69.636	45067
3	22938	75.207	56332
3	22938	80.608	56333
3	22940	80.608	56334
3	22953	80.608	67599
3	22953	86.009	67600
3	22953	86.009	67601
3	22953	86.009	67602
3	22954	86.009	67603
3	22955	86.009	67604
3	22968	86.009	78869
3	22970	91.351	78870
3	22971	91.351	78871
3	22971	91.351	78872
3	22972	91.351	78873
3	22984	91.351	90138
3	22986	96.69	90139
3	22986	96.69	100004
3	22998	96.69	101404
3	23000	101.991	101405
3	23003	101.991	101406
3	23015	101.991	112671
3	23015	107.313	112672
3	23027	107.313	123937
3	23038	112.632	135202
3	23049	118.077	146467
3	23060	123.598	157732
3	23071	128.996	168997
3	23073	134.466	168998
3	23073	134.466	168999
3	23084	134.466	180264
3	23086	139.973	180265
3	23087	139.973	180266
3	23098	139.973	191531
3	23098	145.302	200003
3	23108	145.302	200003
2	23109	149.298	1
2	23109	149.298	2
2	23111	149.298	3
2	23111	149.298	11
2	23111	149.298	103
2	23111	149.298	1003
2	23111	149.298	10000
2	23120	149.298	11268
2	23129	159.515	22533

2	23138	169.687	33798
2	23146	179.867	45063
2	23146	190.045	45064
2	23154	190.045	56329
2	23162	200.243	67594
2	23170	210.417	78859
2	23170	220.595	78860
2	23176	220.595	90125
2	23176	230.808	90126
2	23176	230.808	100000
2	23184	230.808	101391
2	23192	241	112656
2	23192	251.167	112657
2	23199	251.167	123922
2	23201	261.383	123923
2	23209	261.383	135188
2	23217	271.624	146453
2	23217	281.852	146454
2	23222	281.852	157719
2	23222	292.047	157720
2	23222	292.047	157721
2	23229	292.047	168986
2	23237	302.26	180251
2	23245	312.511	191516
2	23247	322.743	191517
2	23247	322.743	200002
2	23255	322.743	200002
1	23255	330.459	1
1	23255	330.459	11
1	23255	330.459	101
1	23255	330.459	1001
1	23255	330.459	10000
1	23263	330.459	11266
1	23270	350.653	22531
1	23270	371.469	22532
1	23277	371.469	33797
1	23279	391.792	33798
1	23286	391.792	45063
1	23286	412.036	45064
1	23292	412.036	56329
1	23298	432.356	67594
1	23298	452.901	67595
1	23301	452.901	78860
1	23307	473.083	90125
1	23307	493.396	100000
1	23313	493.396	101390
1	23319	514.549	112655
1	23325	535.461	123920
1	23331	556.472	135185
1	23337	577.053	146450
1	23340	597.367	146451
1	23346	597.367	157716
1	23352	617.872	168981
1	23358	638.425	180246
1	23364	658.319	191511
1	23364	679.015	191512

1	23364	679.015	200001
1	23370	679.015	200001
0	23373	694.405	10
0	23373	694.405	100
0	23373	694.405	1000
0	23379	694.405	4976
0	23385	717.815	9953
0	23385	735.523	10000
0	23390	735.523	14957
0	23390	753.474	14958
0	23395	753.474	19958
0	23400	771.218	24990
0	23405	788.978	30016
0	23410	806.7	34990
0	23415	824.212	39972
0	23417	841.782	39973
0	23422	841.782	44978
0	23422	859.431	44979
0	23425	859.431	49941
0	23430	876.915	54916
0	23435	894.391	59911
0	23437	911.956	59912
0	23442	911.956	64883
0	23447	929.634	69862
0	23452	947.551	74814
0	23454	965.091	74815
0	23459	965.091	79822
0	23464	982.873	84759
0	23465	1000.28	84760
0	23469	1000.28	89706
0	23473	1017.69	94674
0	23474	1035.2	94675
0	23478	1035.2	99643
0	23478	1052.74	100000
0	23482	1052.74	104606
0	23483	1070.24	104607
0	23487	1070.24	109523
0	23491	1087.54	114471
0	23495	1104.95	119399
0	23497	1122.27	119400
0	23501	1122.27	124323
0	23501	1139.59	124324
0	23505	1139.59	129237
0	23509	1156.87	134184
0	23510	1174.3	134185
0	23514	1174.3	139075
0	23514	1191.53	139076
0	23518	1191.53	143988
0	23522	1208.89	148894
0	23526	1226.24	153817
0	23530	1243.6	158690
0	23534	1260.94	163586
0	23538	1278.23	168475
0	23538	1295.48	168476
0	23541	1295.48	173369
0	23545	1312.85	178278

0	23549	1330.16	183172
0	23553	1348.08	188074
0	23553	1364.88	188075
0	23556	1364.88	192962
0	23558	1379.79	192963
0	23562	1379.79	197847
0	23562	1394.73	200000
0	23566	1394.73	200000

Total time elapsed: 1411.15 seconds

You can also refer to Appendix C for the entire experimental results on CD attached with the thesis report.

Appendix C Source Code, Documentation and Experimental Results CD

Refer to the CD attached with the thesis report.

Appendix D Paper Publication

The work conducted in this thesis and some of the experimental results have been included in the following paper:

N. Bouhmala, O-C. Granmo, Sirar Salih, Yujie Song: *A Tabu Search Algorithm Combined with Learning Automata for the Satisfiability Problem.*

As of June 15, 2011, the paper is to be submitted as a chapter in book.