Tabu Search Parameterization and Implementation in a Constraint Programming Library

Ilija Jovanoski *, Ivan Chorbev *, Dragan Mihajlov *, Ivica Dimitrovski *
* Faculty of Electrical Engineering and Information Technology / Institute for Computer Science and Engineering, Skopje, R. of Macedonia, e-mail: ilijaj@gmail.com, {ivan, dragan, ivicad} @feit.ukim.edu.mk

***Abstract*—Tabu Search is a metaheuristic that guides a local heuristic search procedure. The goal is to efficiently explore the search space of a problem by using memory structures to prevent the search from being stuck in local optima. In this paper, we give an overview of a C# based, multi-threaded, tabu search implementation within a Constraint Solving Engine. The engine is designed for modeling and solving problems that can be defined as Constraint Satisfaction Problems. Here we elaborate some Tabu Search adaptation issues and some experimental results obtained via parameterization of tabu tenure, number of iterations, and number of threads. The problem used to evaluate the implementation was the Traveling Salesman Problem.**

***Keywords*—tabu search, constraint programming, solving engine.**

I. Introduction

This paper presents one part of a broad research effort of combining constraint modeling and application of modern and innovative hybrid, metaheuristic optimization algorithms. One of the aims of the research is the concept of universality. The concept assumes that every Constraint Satisfaction Problem (CSP) can be modeled with the constraints and classes provided in our Constraint Solving Engine (CSE) [17]. Modeled in this way, the problem can be solved by applying some of the optimization techniques and algorithms available in the package. Different algorithms (Simulated Annealing, Generalized Arc Consistency – Conflict Back Jumping (CAC-CBJ), etc) have previously been included in the package, with flexibility of adding more. Tabu Search has proven effective in many applications, therefore its implementation, or implementation of its ideas in other hybrid metaheuristics, seemed like the obvious next step.

In this paper our main interest is exploration of our tabu search implementation in C#.NET. Efficiency and performance measurements were performed and documented when different parameter changes were applied against the model. Also, the adaptation of TS to fit in the previously developed Constraint Solving Engine is an issue.

Section 2 gives an overview of Tabu Search. In Section 3 of this paper, current achievements and variations of Tabu search implementations are summarized and reviewed. Some of the past and ongoing projects and research papers on this subject are presented.

Section 4 describes our implementation of TS in C#, gives a parameterization overview of tabu tenure (tabu list length), number of iterations, and number of threads in multithreaded vs. single threaded implementation. In Section 5, an evaluation summary of the approach is presented with associated figures, tables and charts. The multi-threaded TS implementation test results are given in section 6, while section 7 gives the conclusion.

II. Tabu Search

As a member of the stochastic and heuristic algorithms group, Tabu search (TS) [1, 2] originates from the need to avoid cycling and being trapped into local minima in greedy local search strategies. The TS-based algorithms will continue the search even when a locally optimal solution was found. TS is a process of subsequent moves from one local optimum to another thus exploring larger part of the solution space.

The fundamental scheme behind TS method is allowing climbing moves when no improving neighboring solution exists, i.e. a move is allowed even if a new solution from the neighborhood of the current solution is of inferior quality compared to the current one. Because this notion can lead to cyclic behavior of the search, in TS some moves become forbidden or "tabu" from time to time.

Formally, TS starts from an initial solution and it then iterates visiting current solution's neighborhood. At each step of the procedure, a certain subset of the neighboring solutions of the current solution is considered, and the move that improves most the objective function value is chosen. The newly found solution is not necessary better than the existing one: in case of no improving moves, TS chooses the move that least degrades the objective function. With the intention to eliminate an immediate returning to the solution just visited, the reverse move must be forbidden. For this purpose, a memory structure for storing the moves - a tabu list, is introduced. The list keeps information on the last m moves, which have been performed during the search process. This ensures that the algorithm would not go back to a solution reached in the previous m steps. However, such straightforward prohibition may sometimes reduce the efficiency of the search. Furthermore, it is sometimes desirable to return to a previously visited solution in order to search in a different promising direction. To accomplish this goal, an aspiration criterion is introduced to allow the tabu status to be overridden under certain circumstances. Usually, a move from a solution s to solution s′ is permitted if s′ is better than s* (s* being the best solution found so far).

Tabu search stops when a termination criterion is satisfied, which is generally a fixed number of iterations.

More detailed description on fundamentals of TS, its modifications and applications can be found in [3, 4].

III. Related Work

In this section, we present some approaches where TS was implemented and/or used as standalone or combined with other methodologies to facilitate the resolution of different classes of well-known problems.

Tabu search has been extensively used since its introduction by Glover et al. [1,2] for successful resolution of wide specter of problems such as: the Job Shop Problem [5], SAT and MAX-SAT [6], the Traveling Salesman Problem, Graph-Structured Case Retrieval, the Railway Scheduling Problem[9], Glass Cutting / Bin Packing, the Built-In Self-Test and others.

In [5] the issue of taboo list size for solving the job shop-scheduling problem with fuzzy reasoning is discussed. The initially given size of the tabu list is modified by the use of fuzzy logic system. This tabu list size is then used in tabu search process for solving the job shop scheduling problem. The impact of many other factors such as aspiration level and problem instance data on the tabu search process is not considered. However, it is concluded that these factors may be very important as well.

TSAT, a basic tabu search algorithm for SAT, has been proposed in [6] and compared with Random Walk Strategy GSAT procedure [7]. TSAT makes a systematic use of a tabu list and proves very competitive in the resolution of hard random K-SAT instances. The optimal length of the tabu lists, found experimentally for these random problems, is linear with respect to the number of variables.

Similar to this approach is another one tackling with MAX-SAT. The paper reference in [8] introduces a stochastic local search algorithm for MAX SAT called Iterated Robust Tabu Search (IRoTS). This algorithm combines two methods for solving a variety of other hard combinatorial optimization problems, Iterated Local Search (ILS) and Robust Tabu Search (RoTS). The empirical analysis of IRoTS on a range of MAX-SAT instances shows that in many cases IRoTS outperforms GLS, one of the best-performing MAX-SAT algorithms, and ILSYI, an earlier and simpler Iterated Local Search algorithm. However, there are also observed cases in which the performance of IRoTS did not reach that of GLS.

A Tabu Search Algorithm for the Railway Scheduling Problem was presented in [9]. The general scheduling problem was modeled by means of an alternative graph. It is a mathematical model, allowing a detailed representation of many scheduling problems. This approach was successfully applied to several real world environments, such as the production of stainless steel and traffic management of railway networks.

Puchinger et al. has used tabu search as supplementary method in a glass cutting problem [10] and two-dimensional bin packing [11]. In the glass-cutting problem, tabu search was used to assign the elements to the glass sheets. Later cutting patterns for individual sheets are obtained by different inner heuristics. In bin packing tabu search was used to assign the items to bins.

Unlike local search, which stops when no improved new solution is found, TS continues the search for the best solution in the neighborhood, even if the next step is worse than the current solution. To take advantage of the described diversification, Ubar et al. has successfully applied TS for on-chip test solutions, usually referred to as built-in self-test (BIST) [12]. Tabu search method was used to search for a global minimum in a search space consisting of many local minima, thus allowing to select the optimal switching moment from pseudorandom test generation mode to the stored deterministic patterns mode.

The Traveling Salesman Problem has been used as an input to many solution-space searches. An improved and optimized tabu search framework (ITS) was tested on the traveling salesman problem in [13]. Their main idea is to try to achieve more efficiency by coupling the standard tabu search with the proper reconstruction (diversification) mechanism. The results obtained from the experiments with the TSP instances, especially the smaller TSP instances, confirm that the performance of traditional TS is improved considerably if the pure TS acts as an effective local improvement procedure within the ITS paradigm. Some of the following directions of improvement are proposed: using the reactive tabu search [14] as a probably more efficient local improvement procedure; implementing more elaborated diversification operators within ITS; trying new efficient "cold restart" techniques; maintaining "the history" of the locally optimal solutions; incorporating the limited runs of ITS into other meta-heuristics, for example, genetic algorithms.

In our previous research [16], we presented a new algorithm for combinatorial optimization based on the basic TS scheme named Adaptive Tabu Search (A-TS). The A-TS introduces a new, complex function for evaluation of moves. The new evaluation function incorporates both the aspiration criteria and the long-term memory. A-TS also introduces a new decision making mechanism, providing means for avoiding possible infinite loops. The performance of A-TS was measured by applying it to the Quadratic Assignment Problem. We compared the experimental results to published results from other authors. The data shows that A-TS performs favorably against other established techniques.

We have previously used ideas and methods from TS, especially memory, within implementations of hybrid SA [18].

All of the presented works are of interest for our study, particularly [5, 8, 13], where different approaches on implementation and optimization on the tabu search are given. In our work, we have implemented tabu search algorithm in C#. We have obtained some experimental result on the performance of this implementation by applying different parameter - changing schemes, associated figures, tables and charts.

IV. Tabu Search C#.Net Implementation

Our implementation of tabu search had to fit in a previously developed framework of generic constraints for problem modeling and a set of heuristic algorithms (Simulated Annealing, GAC-CBJ, Greedy Search). The engine provides interfaces for using pre-developed neighborhood functions, pre-constructed problem models

stated using the engine's generic constraints etc. Therefore, the implementation is as follows:

Every entity in the TS algorithm has its own class:

- *CTabuSearchBase:CAlgTabuSearch* represents the base class for CMultiThreadedTabuSearch and CSingleThreadedTabuSearch
- *CSingleThreadedTabuSearch*, an actual single - threaded implementation of TS which blocks until the given number of iterations have been completed.
- *CMultiThreadedTabuSearch*, multi - threaded implementation that uses CNeighborhoodHelper(s) to delegate the work of processing of all generated moves
- *CNeighborhoodHelper*(s) use CSingleThreadedTabuSearch to process its share of all generated moves.
- *CAaspirationCriteria* defines when to consider a move even if it is a tabu i.e. to override the tabu status of a move
- *CObjectiveFunction* has methods for evaluation of a possible solution
- *CTabuList*, a list to keep track of forbidden moves
- *CMove* defines a move in the solution space
- *CTabuSolution* defines a solution of a given problem treated with TS

There are two modes of running the implementation: single-threaded and multi-threaded. Multi-threaded mode uses single-threaded methods in its multiple threads. Either way, the essential method is performOneIteration() where the following takes place:

Get the list of moves via CMoveManager
Get best move (CSingleThreadedTabuSearch.getBestMove)
    Iterate through all moves
    Get best move based on
        obj. function, tabu list, aspiration criteria
Enroll the move into tabu list
Register the new value (is it better or worse)
Operate the move to the current solution

The algorithm stops after a given number of iterations. The implemented aspiration criteria was best-ever, meaning that a tabu move should be allowed if it results in a value better than the current best solution.

The parameterization of the TS implementation was performed to the following values:

- tabu tenure (tabu list length),
- number of iterations,
- number of threads, and
- number of customers,

given that the evaluation was done with the Traveling Salesman Problem (TSP) instances. The solution value of a TSP instance is taken into account in the tables and graphs throughout this paper. Solution values are taken from the objective function (CObjectiveFunction) whose task is to find the distance between customers/cities visited by the salesman. The lesser the solution value is, the better the solution. CMoveManager.getAllMoves() generates moves that move each customer forward and back up to five spaces, so to be able to explore the solution space.

V. Evaluation And Experimental Results

In the following paragraphs, some tables and graphs from the experimental results are presented. The results are presented in terms of number of iterations and solution cost, instead of seconds, due to hardware dependence. The test runs were made using the following Personal Computer hardware system configuration:

- AMD Athlon 64-bit 3000+ Processor
- 1.81 GHz working frequency
- 512 MB RAM

Initially we observed the improvement of the solution value of TSP by increasing the number of iterations. As expected with higher number of iterations, the solution is getting better. However, the trend of improvement stops when saturation is reached. In Fig.1, the case of 1000 customers with tabu tenure of 20 is shown. We can see that after 1000 iterations there are no solution improvements.

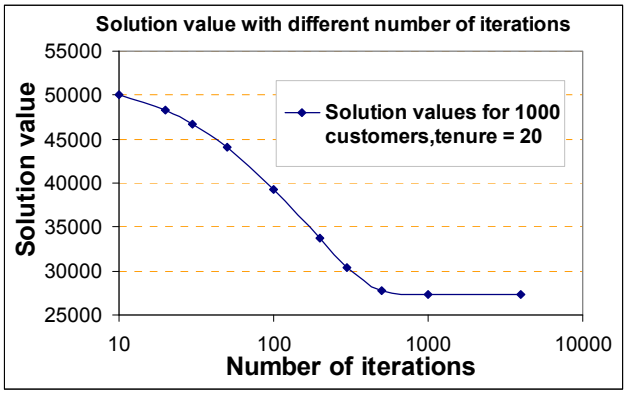

Fig. 1. Solution improvement with increased number of iterations.

Similarly, the case of 1000 customers, but with tabu tenure of 800 is shown in Fig.2. The change in the tabu tenure causes the solution to gain better values even at higher number of iterations. However, with such tenure, the results at lower number of iterations is worse than in the case of shorter tabu tenure, due to the fact that the tenure itself if longer than the number of iterations. There, the algorithm is a simple random search instead of a heuristic Tabu Search.

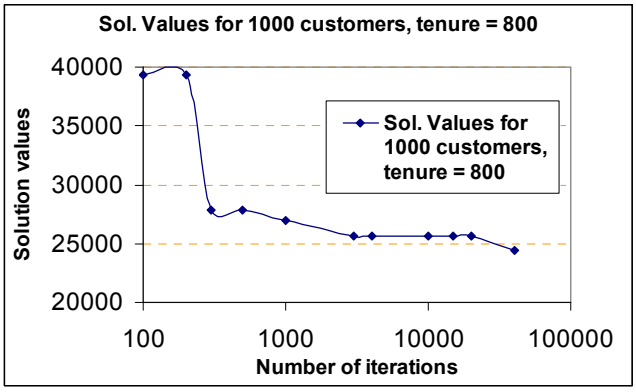

Fig. 2. Solution improvement with increased number of iterations, different tabu tenure.

Fig.3, Fig.4 and Fig.5 give graphs for settings with 50, 100 and 500 customers respectively. The given graphs show the effect of how change in tabu tenure can improve the solution value. It is evident that tabu tenure values close in value with the number of customers n give better solutions. If the tenure value goes too high above or to low bellow n, the solution quality decreases.

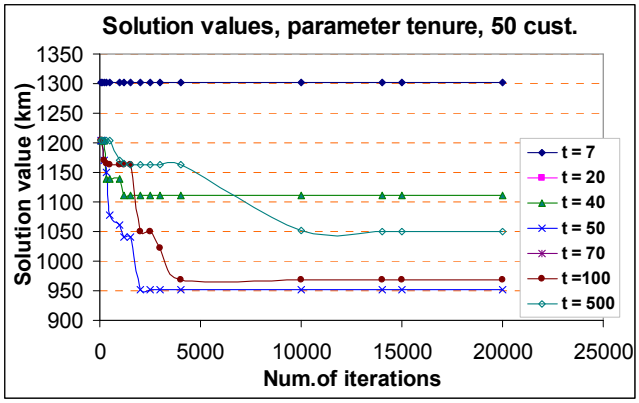

Fig. 3. Tabu tenure parameterized for 50 customers.

The previous conclusion was demonstrated with several values for the number of customers combined with different tabu tenure values. The best solution values were always obtained when there was linear dependency between the number of customers and the tabu tenure value. The dependency is shown in Table 1. and Fig.6.

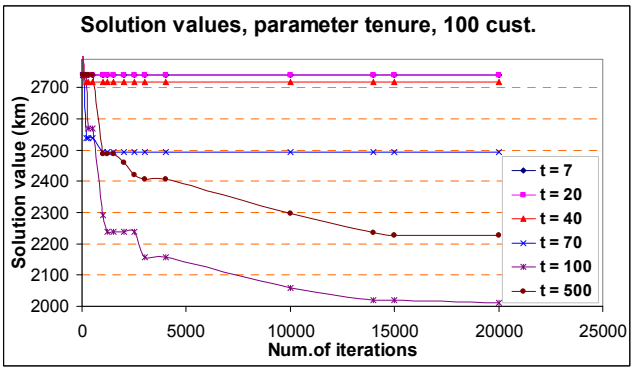

Fig. 4. Tabu tenure parameterized for 100 customers.

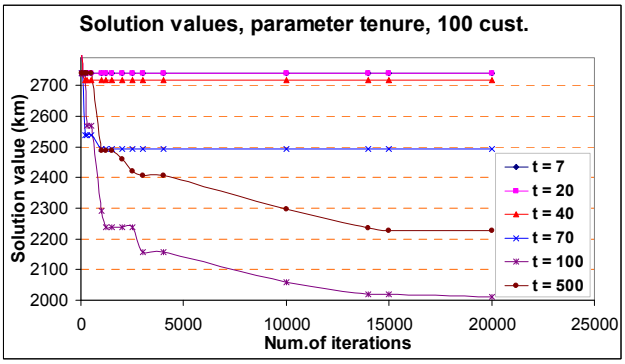

Fig. 5. Tabu tenure parameterized for 500 customers.

TABLE I.
NUMBER OF CUSTOMERS AND CORRESPONDING TENURE AT BEST SOLUTION VALUE

Number of customers	Tabu tenure	Best solution value
50	50	952.13
100	100	2010.64
200	200	4467.77
300	300	7427.16
400	400	10016.32
500	500	12456.98
1000	1000	24079.06

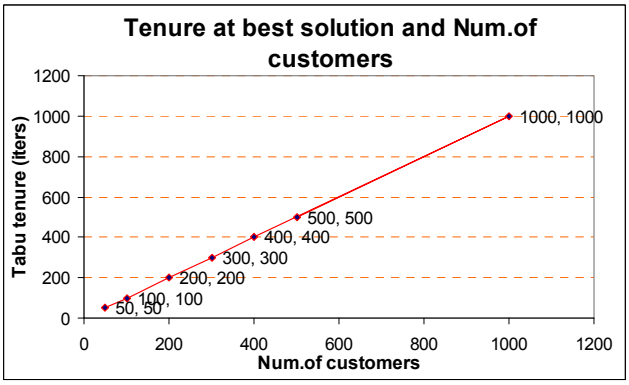

Fig. 6. Tabu tenure at best solution for different settings.

Another observational goal was to find the number of improving moves made during TS execution. Also, the number of occasions when the aspiration criteria was used to override tabu status of a given move is a topic of interest. The charted values are given on Fig.7.

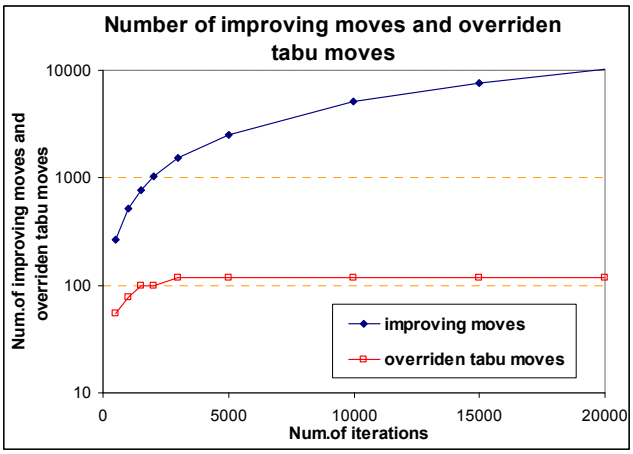

Fig. 7. Number of improving moves and number of uses of aspiration criteria for overriding tabu status, for 50 customers.

In the case of 50 customers (Fig.7), although the number of improving moves increases with higher number of iterations, the number of occasions when a tabu status was being overridden goes into saturation after 5000 iterations. There is similar behavior in the case of 500 customers as well, with values given in Fig.8.

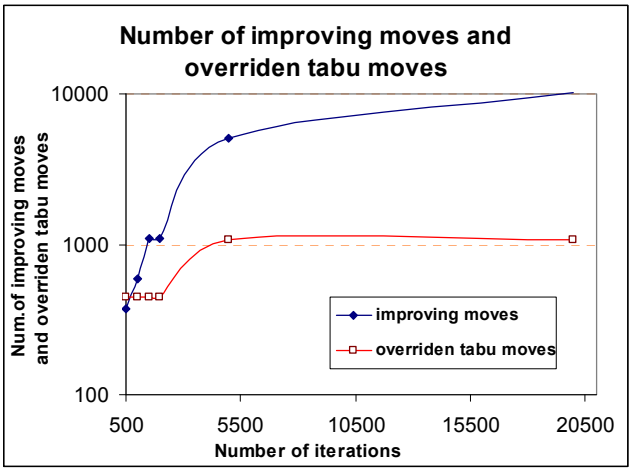

Fig. 8. Number of improving moves and number of uses of aspiration criteria for overriding tabu status, for 500 customers.

Previously mentioned linearity between tabu tenure and the number of customers is appearing again in the highest value of overridden tabu moves vs. improving moves ratio. The conclusion is evident in Table 2, where tabu tenure, number of improving moves and overridden tabu moves are shown. More obviously, dependencies can be seen in Fig. 9, where the ratio (overridden tabu moves / improving moves) is at peak when tabu tenure is equal to the number of customers.

TABLE II.
IMPROVING AND TABU-OVERRIDEN MOVES DEPENDING ON TABU TENURE, 50 CUSTOMERS, 20000 ITERATIONS

Tabu tenure	Improving moves (IMP)	Overridden moves (OVR)	OVR/IMP	%
10	10017	19	0.00190	0.19
50	10143	118	0.01164	1.16
200	10022	53	0.00529	0.53
500	10022	53	0.00529	0.53

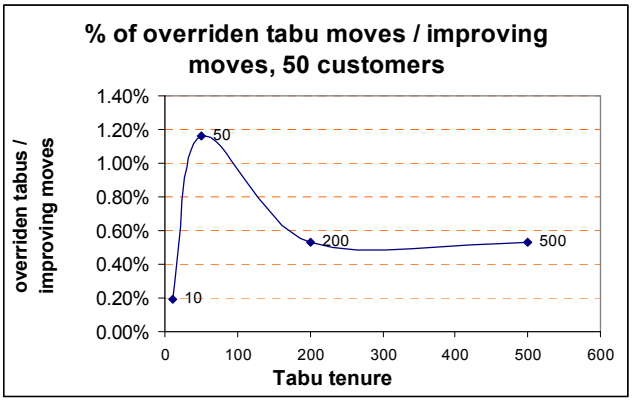

Fig. 9. Overridden tabu moves / Improving moves for 50 customers, 20000 iterations.

VI. MULTI-THREADED EXPERIMENTAL RESULTS

The multi-threaded approach used in the implementation was also analyzed, and somewhat counter intuitive results were achieved. The basic multithreaded mode consist of two threads, one for the main program flow and one for the TS iterations. When the number of threads is increased, every additional thread is considered as a helper thread. The goal is to ease the amount of work in finding best move when a bulk of moves is generated. In Fig. 10 we can see that in our implementation the algorithm runs fastest with only one thread for the TS iterations.

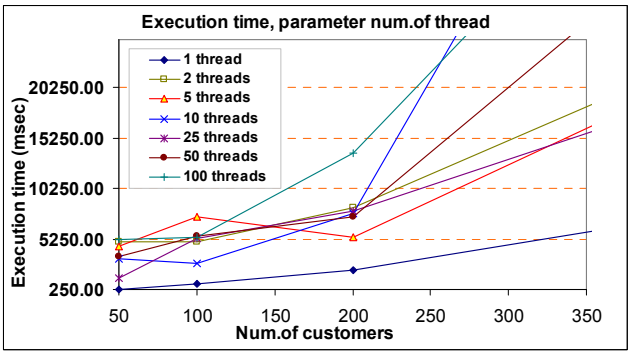

Fig. 10. Execution time with different number of threads.

The increased number of threads poses a synchronization overload to the implementation. More time is spent due to threads waiting each other in moves evaluation so to be able to find the best move in each iteration.

VII. CONSLUSION

In this paper, we have described an implementation and evaluation of the Tabu Search algorithm. The implementation was developed in the C# programming language as a multi-threaded application. We have conducted series of measurements with different parameters when running Tabu Search while solving the Traveling Salesman Problem. With the parameterization of the number of customers, the tabu list length and the number of running threads, we observed that the algorithm achieves best results when the tabu tenure is close to the number of variables.

Although the number of improving moves increases with iterations, the number of occasions when a move's tabu status was overridden achieves saturation after certain number of iterations. Finally, we discovered a limitation in our multi-threaded implementation. Increased number of threads causes time-costly synchronizations to slow down the implementation.

In order to further widen the efficiency of this implementation and make it applicable on real-world problems, we intent to review and compare it with different existing tabu search optimization schemes. The first step is to implement the ideas and benefits of the C++ based implementation, Adaptive Tabu Search that we developed and described in [16]. Feedback mechanisms implemented there will increase accuracy and speed, in the same time maintaining the universality of our C# implementation and constraint modeling. This will help us improve as well as modify any shortcomings in our model. That is the optimal way to apply the ideas of the Adaptive Tabu Search to various constraint modeled, real life, problems.

REFERENCES

[1] F. Glover. Tabu search: part I. ORSA Journal on Computing, 1989, Vol.1, 190 - 206.

[2] F. Glover. Tabu search: part II. ORSA Journal on Computing, 1990, Vol.2, 4 - 32.

[3] F. Glover, M. Laguna. Tabu Search, Kluwer, Dordrecht, 1997.

[4] A. Hertz, E. Taillard, D. de Werra. Tabu search. In E.Aarts, J.K.Lenstra (eds.), Local Search in Combinatorial Optimization, Wiley, Chichester, 1997, 121 - 136.

[5] P.C. Yen and Guohua wan, Solving the Job Shop Problem using Taboo Search with Fuzzy Reasoning, Benjamin Department of Industrial Engineering and Engineering Management, The Hong Kong University of Science and Technology, Hong Kong

[6] Bertrand Mazure, Lakhdar Sais, Eric Gregoire, Tabu Search for SAT, CRIL Universite d'Artois, Lens Cedex, France

[7] Selman, B; Kautz, H; and Cohen, B; Local Search Strategies for Satisfiability Testing, Proceedings of DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability, 1993.

[8] Kevin Smyth1, Holger H. Hoos1 and Thomas Stutzle, Iterated Robust Tabu Search for MAX-SAT, Department of Computer Science, University of British Columbia, Vancouver, Canada; Fachbereich Informatik, Technische Universitat, Darmstadt, Germany

[9] Dario Pacciarelli, Marco Pranzo, A Tabu Search Algortihm for the Railway Scheduling Problem, Dipartimento di Informatica e Automazione, Universita di Roma Tre, Roma, Italy, Proceedings of 4th Metaheuristics International Conference, 2001

[10] Jakob Puchinger, Gunther R. Raidl, and Gabriele Koller, Solving a Real-World Glass Cutting Problem

[11] Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria

[12] Jakob Puchinger and Gunther R. Raidl, Models and Algorithms for Three-Stage Two-Dimensional Bin Packing, Institute of Computer Graphics and Algorithms Vienna University of Technology, Vienna, Austria, Elsevier Science, November 2005

[13] Raimund Ubar, Helena Kruus Zebo Peng, Gert Jervan, Using Tabu Search Method for Optimizing the Cost of Hybrid BIST, Tallinn Technical University, Estonia, Linköping University, Sweden

[14] Alfonsas Misevičius, Using Iterated Tabu Search For The Traveling Salesman Problem, Kaunas University of Technology, Department of Practical Informatics, Kaunas, Lithuania, ISSN 1392 – 124x Informacinės Technologijos Ir Valdymas, 2004

[15] Roberto Battiti, Giampietro Tecchiolli, The Reactive Tabu Search, Dipartimento di Matematica and Istituto Nazionale di Fisica Nucleare, Universita di Trento; Istituto per la Ricerca Scientica e Tecnologica, Trento, Italy, ORSA Journal on Computing Vol. 6, N. 2 (1994), p. 126-140.

[16] Elena Ikonomovska, Ivan Chorbev, Dejan Gjorgjevik, Dragan Mihajlov; The Adaptive Tabu Search and Its Application to the Quadratic Assignment Problem, Proceedings of the 9th International Multiconference Intelligent Systems 2006, Ljubljana, Slovenia, p 27-30

[17] Jolevski I., Loskovska S., Chorbev I., D.Mihajlov. An Overview of a Constraint Solving Engine with Multiple Optimization Algorithms, ITI2005 Cavtat, Croatia, 2005

[18] Jolevski I., Loskovska S., Chorbev I., Mihajlov D., Murgovski N., Constraints Modeling of the High School Scheduling Problem, Eurocon 2005, Belgrade, Serbia and Montenegro, 2005, pages 748-751