Application of Self Controlling Software Approach to Reactive Tabu Search

Nilgun Fescioglu-Unver
TOBB Economics and Technology University
Sogutozu Cad., Ankara, Turkey
nfunver@etu.edu.tr

Mieczyslaw M. Kokar
Northeastern University
360 Huntington Ave., Boston, Massachusetts, USA
mkokar@ece.neu.edu

Abstract

In this paper the principle of self adaptation is applied to achieve a self controlling software. The software considered in this case is a heuristic search algorithm: the reactive tabu search. In reactive search algorithms, the behavior of the algorithm is evaluated and modified during the search. To improve self adaptation, two new strategies for reactive tabu search are introduced. The first strategy uses a control theoretic approach, treats the algorithm as a plant to be controlled and modifies the algorithm parameters to control the intensification of the search. The second strategy adjusts several parameters according to the feedback coming from the search to achieve diversification during the search. These strategies adjust the parameters of the tabu search and form the Self Controlling Tabu Search (SC-Tabu) algorithm. The performance of the algorithm is tested on different problem types of the Quadratic Assignment Problem (QAP). The results show that the algorithm adapts successfully to achieve good performance on problems with different structures.

1 Introduction

Self adaptive software is software that changes itself at runtime to achieve better performance. Heuristic search algorithms are software which try to find good solutions to optimization problems within reasonable time limits. Reactive heuristic search algorithms modify algorithm parameters and/or strategies during the search to improve the search quality. In this paper we apply the principles of self adaptive software to reactive tabu search and implement and test the Self Controlling Tabu Search (SC-Tabu) algorithm on the Quadratic Assignment Problem (QAP).

Most of the real life combinatorial optimization problems cannot be solved with exact algorithms within reasonable time limits. Heuristic algorithms are used to solve these problems but the performance of these algorithms depends on the values of their parameters. One set of parameters which give good results on one problem type may perform poorly on another type. Researchers work on developing methodologies for selecting these parameters [1, 19, 36]. Parameter tuning and reactive search are two research areas which deal with finding the best parameter values for heuristic search algorithms. Experimental design techniques, statistical analysis methods [36], as well as their combination with local search [1] and machine learning methods [19, 6] are often used for parameter tuning. Reactive search methods change the parameters during the search through a feedback mechanism which uses information about the history of the search [3]. One of the most well known reactive search algorithms is Reactive Tabu Search [4]. Other reactive algorithms include Guided Local Search [33] and application of reaction methods to genetic algorithms [10] as well as other heuristics [18].

In this paper we use the *self controlling software* paradigm proposed in Kokar, Eracar and Baclawski [11, 22, 23] and use a control theoretic approach to adjust the parameters of the controlled plant - the tabu search algorithm. The rest of the paper is organized as follows. In Section 2 several self adaptive software approaches are summarized. The Self Controlling Tabu Search algorithm and its implementation on the QAP is introduced in Section 4. In Section 5 the experimental results with the problem domain and comparisons are presented. Finally, Section 6 discusses the results and future research directions.

978-0-7695-3404-6/08 $25.00 © 2008 IEEE
DOI 10.1109/SASO.2008.16

2 Self Adaptive Software Approach

The concept of self modifying software has been known for some time now. Research on *active software*, *self adapting software* [24, 27], *software cybernetics* [20], *self controlling software* [11] and *autonomic computing* [21] represents multiple ways of achieving self adaptive behavior in software.

Laddaga [24, 25] separated the work on self-adaptive software into three groups: coding as a dynamic planning system, coding as a control system and coding as a self aware system. In the coding as a dynamic planning system approach, a system plans its actions and re-plans when the effectiveness of the plan decreases. In the coding as a control system approach, the software is a system with monitoring and control units. The reconfiguration of the system is managed by the evaluation, measurement and control units. In the coding as a self aware system approach evaluation, revision and reconfiguration is part of the running software. The application has knowledge of its operation, evaluates itself, reconfigures and adapts to changes. Robertson and Laddaga propose to achieve the self-adaptive behavior through the use of reflective layers which interpret the environment. In their architecture the program knows itself and can alter itself in order to respond to changes in the real world. [28].

Lately, several researchers consider designing self adaptive software using approaches such as *Checkland's Soft System Methodology* [8, 32], *Beer's Viable System Model* [5, 32, 17] and hidden markov models [34]. Herring use the Viable System Model as the basis for the *Viable Software Architecture* [17]. The Viable System Model of Beer [5] is based on insights from neurophysiology, control theory and cybernetics, and investigates the organizational structure of viable organizations. The Viable Software Architecture applies this model to software architecture. Another model recently used in self adaptive software architecture is the Soft Systems Methodology (SSM). Soft Systems Methodology (SSM) seeks to utilize the basic principles of systems thinking to analyze complex situations where the problem definition is not clear [8]. SSM is used for capturing the requirements of a self adaptive software [32]. In addition to these approaches Wang employ the hidden markov model to model the software environment and achieve self-adaptation at runtime [34].

In engineering, *control theory* (cf. [12]) is used for controlling dynamic systems. Kokar, Eracar and Baclawski identified software as a candidate for a control plant whose efficiency can increase by dynamic adjustments [11, 22, 23]. This approach is called the *self-controlling software approach*. The use of control theory for controlling software has the benefits of using the concepts that have been developed for years by the control community. Recently, control theory was used in the design of web servers as a means of increasing the service quality [37, 35, 16, 9].

The control theory based self-controlling software model ([11, 22, 23]) maps the concepts of control theory to software engineering. In this model:

- Software is treated as the controlled plant.
- The behavior of the plant and environment is modeled as a dynamic system.
- Inputs to the plant are classified as *control inputs* and *disturbances*. Control inputs control the behavior of the plant according to the control goal, while disturbances change the plant's behavior unpredictably.
- A *controller* subsystem changes the value of control inputs to the plant.
- A *quality of service* (QoS) subsystem provides feedback to control the plant.

In this paper we develop two reaction strategies for reactive tabu search. The first strategy follows the approach described in [22] to control the intensification of the search. The second strategy is a diversification mechanism which adapts algorithm parameters according to the feedback coming from the history of the search.

3 Tabu Search and Quadratic Assignment Problem

Tabu search (TS) is an iterative heuristic algorithm [13, 14, 15]. TS uses the history of search (*memory*) to prevent cycling back to previously visited solutions. In each iteration, a transformation operator - the *move* is used to generate the *neighborhood* of the current solution. The moves operated are stored in the *tabu list*. A move is *tabu* if it reverses one of the transformations on the tabu list. The tabu list size ts indicates the number of iterations a move will be considered as tabu. If the *aspiration condition* is met, a move is not prohibited. One commonly used aspiration condition is: if a move results in a solution which has a better objective value than the best solution, it is not tabu.

In order to improve the performance, additional strategies for intensification and diversification of the search are used in different versions of tabu search algorithm. The intensification strategies direct tabu search to search for solutions similar to each other whereas diversification drives the search to unexplored areas in the solution space [2].

We implemented the SC-Tabu algorithm for the NP-hard Quadratic Assignment Problem (QAP) [29]. The application areas QAP has been used includes image processing,

network design, airport gate assignment, factory/office layout design, printed circuit board design. We used the benchmark problems from the Quadratic Assignment Problem Library (QAPLIB) in our experiments [7].

QAP is an assignment problem where n units are assigned to n locations. There is a flow of supplies $f(i,j)$ between each unit and there is a distance $d(i,j)$ between each location. The number of units - n determines the size of the problem. The goal is to find the assignment $s^* \in S(n)$ which minimizes the total of the sum of the products of flow and distance between the units. $S(n)$ is the set of all permutations of $\{1, \cdots, n\}$ and $s(i)$ gives the location of unit i in the permutation $s \in S(n)$. Equation 1 shows the objective function for this minimization problem.

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f(i,j) \cdot d(s^*(i), s^*(j)) = \min_{s} \sum_{i=1}^{n}\sum_{j=1}^{n} f(i,j) \cdot d(s(i), s(j)) \quad (1)$$

There are several types of Quadratic Assignment Problems. These problems differ from each other according to the distribution of their flow and distance data. Taillard [31] showed that different heuristic algorithms show different performance on different problem types. The problem types can be summarized as follows:

1. *Random uniform instances*: Distance and flow data are randomly generated from a uniform distribution.
2. *Random flows on grids*: Flows are randomly generated. Distance between units is the Manhattan distances between grid points of an $n_1 \times n_2$ grid.
3. *Real life instances*: The facility layout, circuit design and typewriter design problems are examples to this type.
4. *Real life like instances*: Distribution of the distances and flows are not uniform. They resemble the real life problems but are bigger in size.

The real life problems available in the QAPLIB [7] are small in size hence can not measure the performance of heuristic algorithms effectively. In this research we implemented the Self Controlling Tabu Search and experimented on the random uniform instances and the real life like instances.

There are several implementations of tabu search algorithm on QAP. One type of move that is mostly used in these algorithms is exchanging the location of two units [30, 4]. After creating an initial solution s, a neighboring solution is obtained by exchanging the location of two units.

$$s'(k) = s(k)\ \forall k \neq i,j \quad s'(j) = s(i) \quad s'(i) = s(j) \quad (2)$$

Information about the moves applied to the solutions are stored in the tabu list. In the SC-Tabu implementation, the tabu list structure of Taillard [30] is used. In this structure the tabu list is a two-dimensional array, tl, indexed by units and locations. The last iteration when unit i was moved to location j is entered to $tl(i,j)$. The tabu list size ts refers to the number iterations a move will be considered tabu. A move is tabu if the following condition is satisfied:

$$tl(i, s(j)) > (currentIteration - ts)\ \&\ tl(j, s(i)) > (currentIteration - ts) \quad (3)$$

If the solution resulting form a tabu move has a better objective function value than the best solution found so far, the move is permitted.

4 Self Control of Tabu Search

Iterative search algorithms move through the space of solutions by selecting a different solution in each iteration. They use different intensification and diversification strategies to direct the search. Intensification refers to focusing the search efforts on a region within the solution space. Diversification, refers to driving the search to different regions. Finding a balance between intensification and diversification is important for the success of the search algorithms. This balance is determined by some parameters of the algorithm. Using the same parameter values for different problems disregard the specific problem's needs. Even while solving a specific problem the need for intensification and diversification change during the search depending on the region the search is currently in. The dynamically changing requirements bring the need of a search algorithm which can modify its own behavior when needed. In this research we controlled the intensification and diversification of the search using self adaptation.

4.1 Controlling Intensification

The control theory based self-controlling software model ([11, 22, 23]) is used for controlling the intensification of the search. In this model the software is treated as system whose parameters can be dynamically changed using a feedback controller. A feedback control system is composed of the following elements. The *Target system* (also referred to as *Plant*) is the system to be controlled. In this case it is the tabu search. The goal is to control the intensification of the search. The *Controlled output* is a variable of

the target system that is controlled. The *Reference input* is the desired value of the measured output. The *Controller* is an equation which determines the value of the control input, given the reference input and the controlled output.

4.1.1 Control Input and Output

In order to design the control system, we first identified the controlled variable and control input. The goal is to control the intensification/diversification of the search and tabu list size ts is already recognized by researchers as a critical parameter for the balance of intensification and diversification [3]. Large ts values increase the number of prohibited moves and encourage diversification while small ts values promote intensification. Hence we selected the tabu list size ts as the control input.

The controlled output was determined in the next step. When the controlled software is a search algorithm the first output that comes to mind is the value of the objective function. However this value gives us no information about whether the system is intensifying or diversifying. Hence it was not used as the controlled variable. The next candidate for controlled output was the repetition length, l_r. It is the length of the interval between repetitions of the same solution. The existence of repetitions was considered as an evidence for need of diversification also in Reactive Tabu Search [4]. The repetition length gives information about not only if there exists a repetition but also how frequently repeating solutions are found. Hence we selected the repetition length l_r as the controlled output.

We performed some experiments with the tabu search algorithm to support our decision on the control input and output. In the experiments, we changed the value of tabu list size ts according to a schedule and observed the repetition length l_r. Figure 1 shows the measurements from the experiment we performed on the Tai35a problem of the Quadratic Assignment Problem set [7]. In this experiment, the run was 5,000 iterations long and the value of ts changed every 200 iterations following a roughly triangular waveform. The repetition length l_r was measured in a sliding window of size 100. The window size of 100 was chosen arbitrarily in this experiment. Use of a sliding window enables taking the recent history into account. Thus, if a solution is found 30 iterations ago, the repetition length l_r is recorded as 30. If the solution has not repeated within the last 100 iterations, the l_r is recorded as 100.

Figure 1 demonstrate the impact of the change in tabu list size ts (the light straight lines) on repetition length, l_r, (the dark marks). It is seen that when ts decreases the repetition length decreases and increases when ts increase. The decrease of the repetition length to very small values means that the search got trapped around a local optima. As ts values increase, l_r values increase and reaches to 100 which

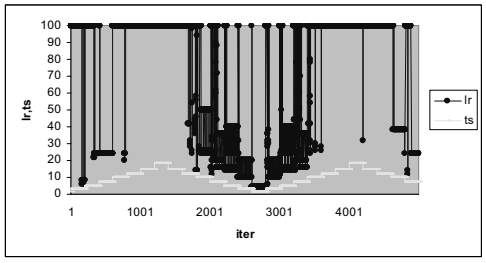


Figure 1. Impact of changes in tabu list size ts on repetition length

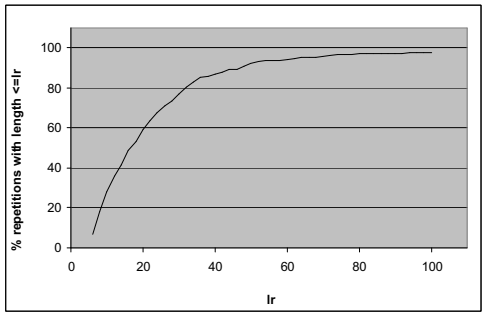


Figure 2. Cumulative distribution of repetition length, l_r

means the solutions found has not been met within the last 100 iterations. Having repetitions shows the search is intensifying on a region, whereas having no repetitions for a while shows that search is moving away from solutions found within the last 100 iterations.

The next step after this experiment was to determine the window size for measuring l_r. To answer this question an experiment was conducted. In this experiment tabu search was run on various QAP problems for 20,000 iterations. The ts was changed between 3 and 21 according to a step function and the repetition length was recorded. The results are shown in figure 2. The figure shows the cumulative distribution of the l_r values recorded during the experiments and it is seen that most of the existing repetitions occur within the last 100 iterations. Hence the size of the sliding window to measure the repetition length was selected as 100.

4.2 Intensification Control

The elements of the feedback control system can be mapped to the tabu search control system as follows: *Target System:* Tabu Search Algorithm; *Control Input:* Tabu List Size, ts; *Measured Output:* Repetition Length, l_r; *Reference Input:* The desired repetition length, l_r^{ref}; *Control*

Error: The difference between the desired repetition length and the measured output, $e = l_r^{ref} - l_r$.

To control this system we decided to use an *integral controller* [16]. Integral control changes the control input proportional to the integral of the past errors. Instantaneous error variations are smoothed out this way and history of the search is taken into account while determining the control input.

The integral control law for our system is

$$e(k) = l_r^{ref} - l_r(k-1) \tag{4}$$

$$ts(k) = ts(k-1) + K_i\, e(k) \tag{5}$$

In equations 4 and 5, k is the iteration number and $ts(0) = 1$. The reference value l_r^{ref} should be selected such that an intensified search is conducted while the repetitions are limited. Obtaining low values of l_r means the search is repeatedly finding the same solutions hence wasting the search time. Continuing with few repetitions can be possible by selecting a high l_r^{ref} but in that case intensification will not be achieved. To avoid going into both extremes, we selected the reference point as $l_r^{ref} = 50$.

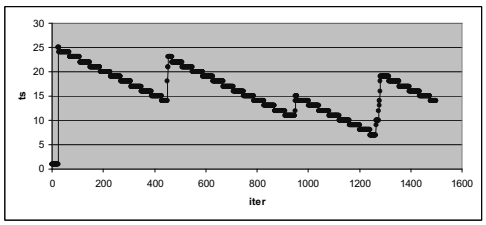
After selecting the reference value, the integral control parameter K_i is determined. Since low l_r values lead into wasting many iterations with repetitions, it is desirable to eliminate repetitions with small l_r values quickly and move toward small values slowly. To achieve this behavior the control parameter K_i is determined through an adaptation loop. This asymmetric behavior is achieved through use of equation 6. The parameters a and b adjust the speed of increase and decrease of the control input ts. The parameters c and d set the turning point for increase/decrease in ts. In this case the turning point is the reference repetition length l_r^{ref}, using the values $c = -1357.47$ and $d = -1370.91$ sets the turning point as $l_r^{ref} = 50$. After off-line tuning experiments the values for a and b are determined as $a = 0.001, b = 0.04$.

$$K_i(k) = \alpha(k) \cdot l_r^{ref}/l_r \tag{6}$$

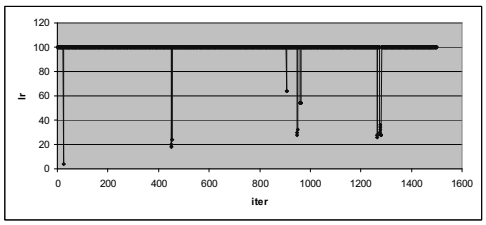
where

$$\alpha(k) = a + b/(1 + e^{d\cdot(l_r^{ref}/l_r(k))-c}) \tag{7}$$

Figure 3 demonstrates how the measured output l_r change according to the changes in ts when this controller is used. The data is taken from a single run of tabu search on Tai30a problem. Initially ts is set to 1, ts stays at this value until a repetition occurs. The first repetition has an l_r of 4, which means the solution was found 4 iterations ago. As this is a very small value, the increase in ts is high and in the next iteration $ts = 24$. In the next phase ts continuously decreases until a new repetition occurs. A repetition length value below the reference causes ts to increase again. This pattern repeats throughout the intensification period.



(a) Tabu list size vs. iterations



(b) Repetition length vs. iterations

Figure 3. Adaptive control behavior

4.3 Supplementary Diversification

The control loop described in 4.2 adapt the ts such that an intensified search is conducted while repetitions are limited. However intensification alone is not sufficient for a successful search. The search should also be directed to unexplored areas of the search space, in other words 'diversify'. To obtain diversification we added another feedback mechanism which gets information from the history of the search and decides on when and how to diversify. This mechanism is based on observing if the search is using most of the available moves and changing the tabu list size such that the moves not used for a while are enforced to be used. The diversification mechanism is composed of two components, the *Observer* which is responsible for observing the use of the moves and the *Diversifier* which modifies the ts as instructed by the Observer.

Figure 4 demonstrates the structure of the Self Controlling Tabu Search. The search starts with the intensification period. During this period, the basic feedback loop feeds the l_r^{ref} and the control error e into the Controller. The adaptation loop uses the control output l_r to adapt the control parameter K_i to be used in the Controller. Controller computes the ts value and sends it to the tabu search algorithm.

Meanwhile the Observer monitors if the *stagnation period* exceeds the given stagnation threshold. The stagnation period is defined as the period of iterations without any improvement with respect to the current best solution [26]. The value of the stagnation threshold is selected equal to number of possible moves, p_o. When the stagnation period length exceeds p_o the Observer counts the number of moves that have not been used since the beginning of the stagna-

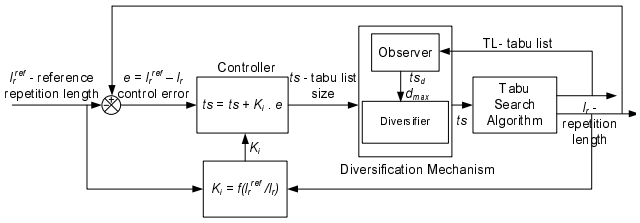


Figure 4. Self Controlling Tabu Search mechanism

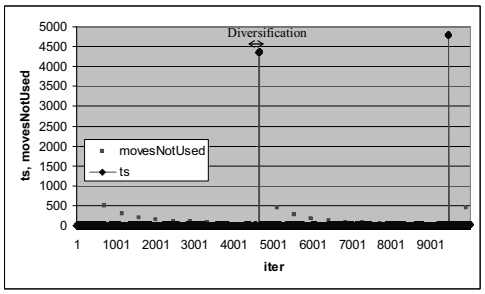
tion period. Starting from this iteration, Observer counts the number of not used moves every p_o iterations. When the decrease in not used moves is less than 15% the Observer cuts the intensification control loop and starts the diversification period.

In the beginning of the diversification period, the Diversifier gets information from the Observer. This information includes the iteration number when the stagnation started or the last diversification period ended, and the number of moves that have not been used since that iteration. The Diversifier increases and adjusts ts in each iteration such that the moves that have been used since that iteration are prohibited. Diversification period length equals to the number of not used moves. Although diversification mechanism targets the use of old moves during this period, it decreases the ts to a small value (one third of the problem size) for short periods so that the old moves and new moves are combined.

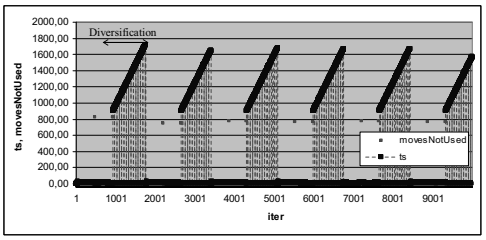
SC-Tabu shows different behavior on different types of problems. We experimented on the random uniform distribution and real life like problems. Figure 5(a) demonstrates an example behavior of SC-Tabu on a random uniform problem and Figure 5(b) illustrates an example behavior on a real life like problem. Figures 5(a) and 5(b) show that real life like problems have longer and more frequent diversification periods than the random uniform problems. The increased diversification need of the real life like problems has also been recognized in the literature [31]. The self-adaptation mechanism adapts the software such that this need is automatically satisfied.

5 Results

The performance of SC-Tabu is compared with Reactive Tabu Seach (Re-Tabu) [4] and Robust Tabu Search (Ro-Tabu) [31]. These algorithms are selected for their good performance on Quadratic Assignment Problem. Both algorithms adjust the tabu list size dynamically. Re-Tabu increase ts by a fixed amount when there is a repetition, and decrease it when there are no repetitions for a predefined period of time. For additional diversification, Re-Tabu use random moves and divert the search to unexplored areas.



(a) SC-Tabu behavior on a QAP - random uniform distribution problem



(b) SC-Tabu behavior on a QAP - real life like problem

Figure 5. SC-Tabu diversification on different problem types

Ro-Tabu adjusts ts according to a predefined schedule during the search. We experimented on the random uniform distributed (Type-A) and real life like (Type-B) problem instances of Taillard. Both of these problem sets can be found in the Quadratic Assignment Problem Library [7]. Type-A problems are named Tai$n$a and real life like problems are named Tai$n$b where n is the number of units and locations. For direct comparison with published results we performed the experiments using the same number of runs and iterations as in tests of Taillard [31].

5.1 Comparison of Solution Quality for Medium Sized Problems

In this section we compare Re-Tabu, Ro-Tabu and SC-Tabu in terms of the quality of solutions found in medium sized problems. In these experiments the number of iterations is limited and the limits are selected as in [31]. Each problem is run for 30 times starting from random initial solutions. Table 1 shows the mean percent deviation of the objective function value from the optimal solution value.

It is seen that for uniform random distributed problems (Type A) both Re-Tabu and SC-Tabu have better performance than Ro-Tabu in three out of four cases. For real life like problems (Type B) SC-Tabu shows better performance than Re-Tabu in all cases and has better performance than Ro-Tabu in three out of five cases.

Table 1. Quality of suboptimal solutions for medium sized problems for Re-Tabu, Ro-Tabu and SC-Tabu

Problem	Iterations	Re-Tabu	Ro-Tabu	SC-Tabu
Tai20a	13043	0.312	0.235	0.246
Tai25a	28913	0.247	0.304	0.251
Tai30a	46864	0.186	0.326	0.178
Tai35a	82264	0.310	0.546	0.347
Tai20b	3675	3.919	0.540	0.189
Tai25b	9473	1.021	0.011	0.486
Tai30b	22048	1.101	0.307	0.327
Tai40b	31595	1.330	0.744	0.299
Tai50b	124867	0.409	0.228	0.212

Table 2. Best solutions found by Re-Tabu, Ro-Tabu and SC-Tabu for large size problems

Problem	Re-Tabu	Ro-Tabu	SC-Tabu	BKV
Tai40a	3141702	3146541	3139370	3139370
Tai50a	4948508	4951186	4938796	4938796
Tai60a	7228214	7272020	7205962	7205962
Tai80a	13558710	13582038	13556348	13515450
Tai100a	21160946	21245778	21110846	21059006

5.2 Comparison of Best Solutions Found for Large Size Problems

In these experiments we compare the performances of Re-Tabu, Ro-Tabu and SC-Tabu on large size problems. Table 2 presents the objective function value of the best solutions found by each algorithm. Results show that SC-Tabu is able to achieve the best known value in the literature in three out of five cases, and improve the values found by Re-Tabu and Ro-Tabu in the other cases.

5.3 Comparison of Solution Quality for Large Size Problems

The last comparison was made on the large size problems using short run times. Each problem was run for 30 times for $1000n$ iterations and the mean percent deviation of the objective function value from the best known value is presented for each algorithm. In order to compare the SC-Tabu results directly with the published results, the best known value is taken as the previously best known value (BKV_{prev}) at the time of the publication of Taillard [31]. The results show that SC-Tabu shows superior performance to Re-Tabu in six out of nine cases and to Ro-Tabu in all cases.

Table 3. Quality of suboptimal solutions found by Re-Tabu, Ro-Tabu and SC-Tabu for large size problems

Problem	Re-Tabu	Ro-Tabu	SC-Tabu	BKV_{prev}
Tai50a	0.952	1.104	0.856	4941410
Tai60a	0.859	1.278	0.886	7208572
Tai80a	0.569	0.961	0.527	13557864
Tai100a	0.387	0.823	0.392	21125314
Tai50b	0.731	0.439	0.423	458821517
Tai60b	0.366	0.899	0.557	608215054
Tai80b	1.800	1.004	0.879	818415043
Tai100b	1.490	0.968	0.681	1185996137
Tai150b	0.807	1.904	0.657	499348972

6 Conclusions

This paper presents an application of the self adaptation principles to software. The specific application subject is a heuristic search algorithm - the reactive tabu search. In self adaptive software approach the software evaluates and changes its behavior to improve its performance. To achieve this behavior we designed two reaction strategies for reactive tabu search.

The first strategy is based on the control theoretic self controlling software methodology. In this methodology the software is treated as a plant to be controlled and the parameters of the plant are adjusted through a feedback mechanism. We controlled the intensification of the search using this mechanism. We first determined the control input and output, and then designed a controller to control the tabu search algorithm. The goal of this control mechanism is to control the repetitions such that the search is intensified and the repetitions are limited.

The second strategy is a supplementary diversification mechanism. This strategy also uses a feedback mechanism and determines when and how long to diversify by using the history of the search. The goal of the supplementary diversification mechanism is to drive the search to unexplored areas of the search space. Both of these strategies combine together and form the Self Controlling Tabu Search (SC-Tabu).

SC-Tabu is tested on the Quadratic Assignment Problem (QAP). We experimented on the random uniform distributed problems and the real life like instances. The algorithm was compared to two well known tabu algorithms, Reactive Tabu Search and Robust Tabu Search. Test results showed that Reactive Tabu performs well on especially random uniform problems, Robust Tabu shows good performance on real life like problems while SC-Tabu can achieve good performance on both types.

The self adaptive nature of the SC-Tabu enables it to modify its behavior and perform good on different types of problems. The use of adaptation techniques improve the quality of results of the tabu search algorithm for the QAP. The use of control theory provided a systematic approach to design the reaction mechanism. Control theory has many other well established techniques such as sensitivity analysis, stability analysis and plant identification. Future research studies could apply these techniques while designing adaptive software algorithms.

References

[1] B. Adenso-Díaz and M. Laguna. Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1), 2006.

[2] R. Battiti. *Reactive Search: Toward Self-Tuning Heuristics*, pages 61–83. John Wiley & Sons Ltd., Chichester, 1996.

[3] R. Battiti and M. Brunato. *Reactive Search: Machine Learning for Memory-based Heuristics*, chapter 21. Taylor & Francis Books (CRC Press), Washington, DC, 2007.

[4] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[5] S. Beer. *Brain of the Firm, 2nd ed.* John Wiley & Sons, Chichester, 1981.

[6] M. Birattari, T. Stutzle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of GECCO-02*, pages 11–18, 2002.

[7] R. Burkard, E. Çela, S. Karisch, and F. Rendl. Qaplib - a quadratic assignment problem library. `http://www.seas.upenn.edu/qaplib/`, 2007.

[8] P. Checkland. *Systems Thinking, Systems Practice*. John Wiley & Sons, Chichester, 1981.

[9] Y. Diao, J. L. Hellerstein, S. Parekh, and J. P. Bigus. Managing web server performance with autotune agents. *IBM Systems Journal*, 42(1), 2003.

[10] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[11] Y. Eracar and M. Kokar. An experiment in using control techniques in software engineering. In *Proceedings of the 1997 IEEE International Symposium on Intelligent Control*, pages 275–280, 1997.

[12] G. F. Franklin, J. D. Powell, and A. Emami-Naemini. *Feedback Control of Dynamic Systems*. Prentice-Hall, 2002.

[13] F. Glover. Tabu search-part 1. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[14] F. Glover. Artificial intelligence, heuristic frameworks and tabu search. *Managerial and Decision Economics*, 11(5):365–375, 1990.

[15] F. Glover. Tabu search-part 2. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[16] J. L. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. John Wiley and Sons, Inc., 2004.

[17] C. E. Herring. *Viable Software the Intelligent Control Paradigm for Adaptable and Adaptive Architecture*. PhD thesis, The University of Queensland, 2002.

[18] H. H. Hoos. An adaptive noise mechanism for walksat. In *Proceedings of AAAI-02*, pages 655–660, 2002.

[19] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Proceedings of Principles and Practice of Constraint Programming*, September 2006.

[20] C. Kai-Yuan, J. Cangussu, R. DeCarlo, and A. Mathur. An overview of software cybernetics. In *Proceedings of Eleventh Annual International Workshop on Software Technology and Engineering Practice*, pages 77–86, September 2003.

[21] J. Kephard and D. Chess. The vision of autonomic computing. *IEEE Computer Magazine*, 36:41–52, 2003.

[22] M. M. Kokar, K. Baclawski, and Y. Eracar. Control theory based foundations of self-controlling software. *IEEE Intelligent Systems and Their Applications*, 14:37–45, 1999.

[23] M. M. Kokar and Y. Eracar. An architecture for software that adapts to changes in requirements. *Journal of Systems and Software*, 50:209–219, 2000.

[24] R. Laddaga. Creating robust software through self-adaptation. *IEEE Intelligent Systems and Their Applications*, 14:25–29, 1999.

[25] R. Laddaga. Active software. *Lecture Notes in Computer Science*, 1936:2–3, 2001.

[26] A. Misevicius. A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, 30:95–111, 2005.

[27] P. Robertson and R. Laddaga. The grava self-adaptive architecture: history; design; applications; and challenges. In *Proceedings of 24th International Conference on Distributed Computing Systems Workshops*, pages 298–303, 2004.

[28] P. Robertson and R. Laddaga. The grava self-adaptive architecture: history; design; applications; and challenges. In *Proceedings of 24th International Conference on Distributed Computing Systems Workshops*, pages 298–303, 2004.

[29] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976.

[30] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel computing*, 17(4-5):443–455, 1991.

[31] E. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3(2):87–105, 1995.

[32] A. Taleb-Bendiab, D. Bustard, R. Sterritt, A. Laws, and F. Keenan. Model-based self-managing systems engineering. In *Proceedings of Sixteenth International Workshop on Database and Expert Systems Applications*, 2005.

[33] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[34] H. Wang and J. Ying. Toward runtime self-adaptation method in software-intensive systems based on hidden markov model. In *Proceedings of 31st Annual International Computer Software and Applications Conference, COMPSAC 2007*, volume 2, pages 601–606, July 2007.

[35] Z. Wang, X. Zhu, and S. Singhal. Utilization vs. slo-based control for dynamic sizing of resource partitions. In *Proceedings of 16th IFIP/IEEE Distributed Systems: Operations and Management*, October 2005.

[36] J. Xu, S. Y. Chiu, and F. Glover. Fine-tuning a tabu search algorithm with statistical tests. *International Transactions in Operational Research*, 5(3):233–244, 1998.

[37] X. Zhu, Z. Wang, and S. Singhal. Utility-driven workload management using nested control design. In *Proceedings of 2006 American Control Conference*, June 2006.