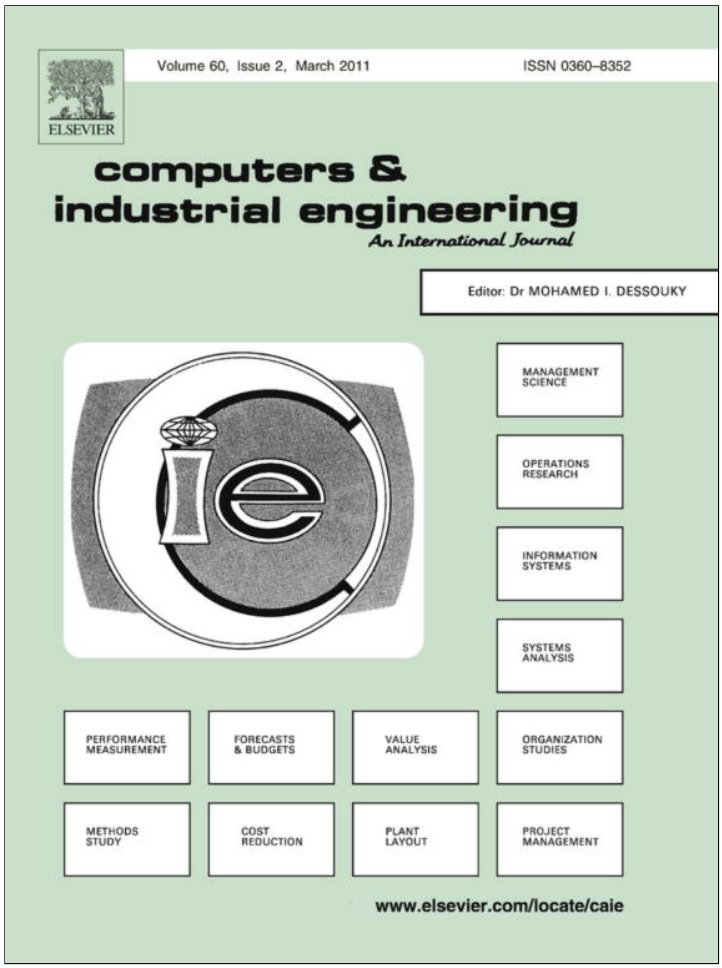

Computers & Industrial Engineering 60 (2011) 310–319

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Self Controlling Tabu Search algorithm for the Quadratic Assignment Problem ☆

Nilgun Fescioglu-Unver [a,*], Mieczyslaw M. Kokar [b]

[a] Department of Industrial Engineering, TOBB University of Economics and Technology, Ankara, Turkey
[b] Electrical and Computer Engineering Department, Northeastern University, Boston, MA, USA

ARTICLE INFO

Article history:
Received 10 November 2008
Received in revised form 24 November 2010
Accepted 24 November 2010
Available online 29 November 2010

Keywords:
Self-controlling software
Tabu search
Reactive search
Quadratic Assignment Problem

ABSTRACT

This paper addresses the application of the principles of feedback and self-controlling software to the tabu search algorithm. We introduce two new reaction strategies for the tabu search algorithm. The first strategy treats the tabu search algorithm as a target system to be controlled and uses a control-theoretic approach to adjust the algorithm parameters that affect search intensification. The second strategy is a flexible diversification strategy which can adjust the algorithm's parameters based on the search history. These two strategies, combined with tabu search, form the Self Controlling Tabu Search (SC-Tabu) algorithm. The algorithm is implemented and tested on the Quadratic Assignment Problem (QAP). The results show that the self-controlling features of the algorithm make it possible to achieve good performance on different types of QAP instances.

1. Introduction

Many real life combinatorial optimization problems are classified as NP-hard. Finding solutions to such problems within reasonable time limits, may not be possible (Garey & Johnson, 1979; Hertz, Taillard, & de Werra, 1997, cha 5). Heuristic algorithms often provide good solutions to optimization problems within reasonable time limits, however heuristic algorithm performance depends heavily upon the selected values for several parameters of the algorithms. A heuristic with a given set of parameter values that performs well on one type of problem may perform poorly on another type. The selection of appropriate values for search parameters is an active area of research (Adenso-Díaz & Laguna, 2006; Hutter, Hamadi, Hoos, & Leyton-Brown, 2006; Xu, Chiu, & Glover, 1998).

Two approaches to parameter selection have been described in the literature: parameter tuning and reactive search. Off-line parameter tuning methods use experimental design techniques and statistical analysis methods (Xu et al., 1998) to adjust algorithm parameters so that a search algorithm performs well on a given set of problems. Procedures combining experimental design and local search techniques are also used for this purpose (Adenso-Díaz & Laguna, 2006). Machine learning is often used to tune algorithm parameters automatically (Birattari, Stutzle, Paquete, & Varrentrapp, 2002; Hutter et al., 2006). Reactive search algorithms modify algorithm parameters and/or strategies during the search through a feedback mechanism which uses information about the search history (Battiti & Brunato, 2007, chap. 21). These algorithms can adapt to the local characteristics of the search space.

One of the best known reactive search algorithms is Reactive Tabu Search (Battiti & Tecchiolli, 1994), which is a modification of tabu search. The tabu search algorithm searches the solution space by applying moves to a solution in order to produce a new one while forbidding the reversal of these moves for a certain number of iterations. Reactive Tabu Search uses a feedback mechanism to determine the number of iterations during which reversing a move is forbidden.

The Guided Local Search approach (Voudouris & Tsang, 1999) guides the search by adjusting the penalty parameters, which are added to the objective function value. Some genetic algorithms (Eiben, Hinterding, & Michalewicz, 1999) use feedback to control the mutation parameters. Another reactive search strategy involves adapting the greediness of the search process according to feedback coming from the search (Hoos, 2002). The improvement, however, significantly depends upon the type of the problem being solved.

In engineering, reaction mechanisms are based on *control theory* (Franklin, Powell, & Emami-Naemini, 2002), which provides a systematic approach to the design and analysis of the adaptation process. In this research, we contribute to the area of reactive search with two reaction strategies. The first strategy uses a control-theoretic approach, in which a controller is used to keep outputs of a dynamical target system at a desired level. In this study, we treat the tabu search algorithm as a target system to be controlled, and use control theory to design a feedback-based reaction mechanism to improve the performance of the search by

☆ This manuscript was processed by Area Editor Ibrahim H. Osman.

* Corresponding author. Tel.: +90 3122924278; fax: +90 3122924091.
E-mail addresses: nfunver@etu.edu.tr (N. Fescioglu-Unver), mkokar@ece.neu.edu (M.M. Kokar).

0360-8352/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2010.11.014

controlling its intensification. The second strategy is a diversification mechanism, which adapts algorithm parameters according to feedback coming from the search history. These two strategies work together to form a new tabu search algorithm – Self Controlling Tabu Search (SC-Tabu). The goal of this paper is not to develop the most efficient search algorithm for a specific problem domain. Rather, the goal is to show that control theory principles can be used to design reaction mechanisms for heuristic algorithms.

We implemented the SC-Tabu algorithm for the NP-hard Quadratic Assignment Problem (QAP) (Sahni & Gonzalez, 1976). Many industrial problems, such as airport gate assignments, factory/office layout design, image processing, network design and printed circuit board design can be modeled as QAP; therefore, the QAP is a good testing domain. In our experiments, we used benchmark problems from the Quadratic Assignment Problem Library (QAPLIB) (Burkard, Çela, Karisch, & Rendl, 2008).

In Section 2, we present short descriptions of tabu search algorithm, intensification and diversification notions, the self-controlling software concept, and the Quadratic Assignment Problem. Section 3 introduces the Self Controlling Tabu Search algorithm. The experimental results with the problem domain and comparisons are presented in Section 4. Finally, Section 5 discusses overall results and future research directions.

2. Background

In this section, we present information about the building blocks of the developed algorithm. We also discuss the Quadratic Assignment Problem, which is the application and test domain of our algorithm.

2.1. Tabu search

Tabu search (TS) was introduced by Fred Glover (Glover, 1989, 1990a, 1990b) as an iterative heuristic-based algorithm for solving combinatorial optimization problems. TS searches the solution space in *iterations*. In each iteration, TS generates a solution *neighborhood* via *moves*. Each move is a transformation that leads from one solution to another and TS keeps the best solution.

The distinctive property of TS is its use of search history (*memory*) to prevent search from cycling back to previously visited solutions. The search history is kept in the memory, called *tabu list*. Tabu list keeps either some of the moves or just their attributes. Reversing these moves is forbidden for a given number of iterations, called the *tabu tenure*. Any moves that would undo these transformations are listed as *tabu*. However, if a tabu move meets the *aspiration condition*, also called the *aspiration criterion* (e.g. it hitherto provides the best solution), then the move is allowed.

2.2. Intensification and diversification

Search algorithms often incorporate two strategies: greedy and exploratory. In a minimization problem, the greedy strategy drives the search toward the steepest decrease in the value of the objective function. This, however, may result in finding local optima, instead of global. The exploratory strategy forces the search to different regions in the search space so as not to miss the global optimum.

Similar ideas are captured by the notions of *intensification* and *diversification*. Intensification refers to focusing the search within an attractive solution region, whereas diversification refers to driving the search to explore unvisited regions (Glover & Laguna, 1998).

When such opposing strategies like intensification and diversification are implemented, the balance between them is determined by certain parameters of the algorithm. Most combinatorial optimization problems have different solution distribution characteristics within the search space. This creates unique requirements for intensification and diversification which must be customized to each situation. Using the same intensification and diversification parameter settings for all types of problems would disregard a specific problem's needs. Moreover, these requirements not only differ from problem to problem, but also often change in the course of solving a single problem. In some problems, good solutions are distributed more or less uniformly within the search space, while in others good solutions are found in clusters (localities). Even if the parameters are tuned for a specific problem instance, using the same settings throughout a search leads to intensifying or diversifying either more or less than needed.

2.3. Self-controlling software

Self-adaptive software is software that modifies itself in runtime to achieve better performance. The idea of self-modifying software has been explored for some time now. For instance, the discipline of machine learning develops software that modifies itself in response to (typically) external feedback. Kokar, Eracar and Baclawski (Eracar & Kokar, 1997; Kokar & Eracar, 2000; Kokar, Baclawski, & Eracar, 1999) introduced the idea of *self-controlling software* in which software can be considered as a *target system* with its own dynamics and can be controlled with feedback based on a Quality of Service function. A similar idea was later independently introduced by Herring (Herring, 2002). Herring's concept was called the Viable Software Approach (VSA). It was modeled after Stafford Beer's Viable System Model (VSM) (Beer, 1981). In software engineering, similar ideas are known as *active software*, *self-adapting software* (Laddaga, 1999; Robertson & Laddaga, 2004), *software cybernetics* (Kai-Yuan, Cangussu, DeCarlo, & Mathur, 2003) and *autonomic computing* (Kephard & Chess, 2003). VSM was also used as part of the design methodology for autonomic systems (Taleb-Bendiab, Bustard, Sterritt, Laws, & Keenan, 2005).

In this paper, we follow the approach described by Kokar et al. (1999), which uses control theory principles to control the software parameters. Control theory is used in engineering to regulate mechanical, electrical, chemical, and computing systems. The essential elements of a feedback control system are as follows:

- *Target system*, also referred to as *Plant*: The system to be controlled.
- *Controlled output*: A characteristic (a variable) of the target system that is controlled.
- *Control input*: A variable that affects the controlled output.
- *Reference input*: The desired value of the measured output.
- *Disturbance*: Variables that affect the measured output but are not controlled.
- *Control error*: The difference between the value of the reference input and the measured output.
- *Controller*: An equation (referred to as the *control law*) that determines the control input value needed in order to achieve the reference input.

The self-controlling software approach identifies the software as a target system whose efficiency can be improved by dynamic adjustments provided by a feedback-based controller (Kokar et al., 1999).

2.4. Problem domain: Quadratic Assignment Problem

The Quadratic Assignment Problem can be described as a logistics problem where n units are assigned to n different locations. For

each pair of units i and j, there is a *flow* $f(i,j)$ (the amount of supplies to be transferred between each pair of units) and for each pair of locations q and r, there is a *distance* $d(q,r)$ specified. The problem is finding a unit location assignment $s^* \in S(n)$ that minimizes the sum of the products of distance and flow between the units, where $S(n)$ is the set of all permutations of $\{1,\ldots,n\}$ and $s(i)$ gives the location of unit i in the permutation $s \in S(n)$. Eq. (1) represents the formulation of the Quadratic Assignment Problem.

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

The number of units and locations determine the size of the problem. A problem with n units to be located is called a problem of size n.

Taillard (1995) showed that the efficiency of a heuristic method in solving a Quadratic Assignment Problem depends upon the type of problem. Quadratic Assignment Problems have been classified into four types (Taillard, 1995):

(1) *Random uniform instances*: Problems in this class are randomly generated from a uniform distribution of distances and flows.
(2) *Random flows on grids*: In this class of problems, the distance between locations equals the Manhattan distance between grid points of a $n_1 \times n_2$ grid. The flows are randomly generated.
(3) *Real life instances*: Examples of these problems include facility layout, circuit design and typewriter design.
(4) *Real life like instances*: These problems are designed to resemble real life instances, but they are bigger in size. The problems are generated randomly and the distributions of distances and flows are not uniform.

In our experiments, we applied our strategies to all of these types of problems.

3. Self Controlling Tabu Search for the QAP

Referring to Section 2.2, satisfying dynamically changing intensification and diversification needs requires a search algorithm that can evaluate and modify, in other words, control its own behavior during search. In this paper, we implemented self-controlling software concepts in order to control the tabu search algorithm. We treated the tabu search algorithm as a plant whose parameters could be changed during the search by a feedback-based controller. To develop such a control mechanism, we first needed to conceptualize our problem as a feedback control system, with the elements described in Section 2.3.

We present the plant in Section 3.1, the control mechanism for intensification in Section 3.2, and the supplementary diversification mechanism in Section 3.3. Section 3.4 describes the complete SC-Tabu algorithm and its behavior on different problem types.

3.1. Tabu search for the QAP

The tabu search algorithm is the plant in our control system. It's Quadratic Assignment Problem specific implementation details are as follows: The algorithm uses the same type of move and tabu list structure used in Taillard's Robust Tabu Search (Ro-Tabu) algorithm (Taillard, 1991). A move exchanges the locations of two units such that each unit occupies the location, which was previously occupied by the other unit. Starting from an initial solution s, a neighboring solution s' is obtained by exchanging the locations of two units i and j such that unit i will occupy the space previously occupied by unit j and vice versa.

$$s'(k) = s(k)\ \forall k \neq i,j\ s'(j) = s(i)\ s'(i) = s(j) \tag{2}$$

The tabu list is a two-dimensional matrix, tl, indexed by units and locations. Specific entries in the matrix are iteration numbers. An entry of $tl(i,j)$ represents the most recent iteration when unit i was moved to location j by the algorithm.

A move (a swap of units i and j) is tabu if unit i has occupied the current location of unit j and unit j has occupied the current location of unit i, within the last tabu tenure ts iterations. Formally, a move is labeled tabu if the following condition is satisfied:

$$tl(i,s(j)) > (currentIteration - ts)\ \&\\ tl(j,s(i)) > (currentIteration - ts) \tag{3}$$

There is a single aspiration criterion: If a move is tabu but the new solution resulting from the move gives a better objective function value than the best solution so far, the move is permitted.

3.2. Controlling intensification

In this section, we identify the controlled output, control input, and the controller of the system.

3.2.1. Identification of intensification control variables

3.2.1.1. Candidate variables for controlled output and control input. The issue of controlling intensification and diversification of reactive search algorithms has been studied before (cf. Battiti & Brunato, 2007, chap. 21), though not through the lens of control theory. It has been recognized that tabu tenure ts is a critical parameter for balancing intensification and diversification. Large ts values encourage diversification while small ts values promote intensification. Decreasing ts reduces the number of forbidden moves so the search is not forced out of the immediate search space. Therefore we selected tabu tenure ts as our control input candidate.

Then we identified three possible candidates for controlled output variable. The candidate variables were: the value of the objective function, the Hamming distance (between solutions), and the repetition length (number of iterations past since the current solution was last found). After identifying these candidate variables, we conducted experiments to evaluate the effectiveness of our control input candidate, and select our controlled output variable.

An obvious choice for a controlled output variable would have been the objective function value. However, since it does not give any information on the amount of intensification/diversification, it was not selected. It was used nevertheless, in the implementation of the basic tabu search algorithm to select the best neighboring solution.

The second candidate variable was Hamming distance. Hamming distance is often used to conceptualize the distance between solutions (Kauffman, 1993; Mattfeld, Bierwirth, & Kopfer, 1999). The Hamming distance between two solutions for a given QAP is measured by counting the number of differing unit-to-location assignments between the solutions. Small Hamming distances indicate a greater degree of similarity between two solutions. The algorithm for computing the Hamming distance d_h between two solutions s and s' of size n is shown below.

$$\forall k \in \{1,\ldots,n\} \quad \text{if}(s(k)\neq s'(k))\ d_h = d_h + 1 \tag{4}$$

The third candidate variable was the repetition length, l_r, which is the length of the interval between solution repetitions, measured in terms of the number of iterations of the algorithm. The repetition length shows how frequently the search is returning to the same solution. The idea of considering repetition as evidence of the need for diversification was first introduced in Reactive Tabu Search (Re-Tabu) (Battiti & Tecchiolli, 1994). In Re-Tabu, the existence of

repetitions is taken into consideration while reacting, whereas in SC-Tabu the actual frequency of repetitions is used.

3.2.1.2. Control variables determination. To determine the controlled output variables and to support our decision on the control input variable, we performed some experiments. In the first set of experiments, tabu search was run on a set of QAP instances. Each run was 5000 iterations long. The value of tabu tenure, *ts*, changed after a fixed number of iterations (e.g., 200) according to a predefined schedule. The schedule followed a roughly triangular waveform as shown in Fig. 1. In this test, the Tai35a problem of the QAP set was used (Burkard et al., 2008). This problem has a size of $n = 35$ and the maximum d_h that can be achieved between any two solutions is 35.

In each experiment, two variables were measured: the Hamming distance, d_h, and the repetition length, l_r. The Hamming distance was calculated for solution pairs that were a fixed distance apart (e.g., $2 \cdot n = 70$ iterations). The repetition length was measured in a sliding window of size 100. Thus, if the current solution was the same as the solution that had been found 23 iterations ago, the value of l_r was 23. On the other hand, if the current solution had not been found at all within the window of 100 iterations, the repetition length l_r of that solution was 100. The use of the sliding window for measuring repetitions was a way of considering the recent search history. The window size of 100 was chosen arbitrarily.

Fig. 1a and b demonstrate the impact of the change in tabu tenure (*ts*) (the light straight lines) on Hamming distance (d_h), and repetition length (l_r), (the dark square marks) for the same iterations within the experiments. When Fig. 1a shows that the value of d_h is low for iteration number 2080, it means that the value of d_h in Fig. 1b is also low for that iteration.

It can be seen that both d_h and l_r decreased when *ts* decreased and increased when *ts* increased. The d_h values increased up to the maximum level for high values of *ts*. This shows that the search successfully diversified in the last 70 iterations. The response of the repetition length was less profound, but the correlation with *ts* is still clearly visible.

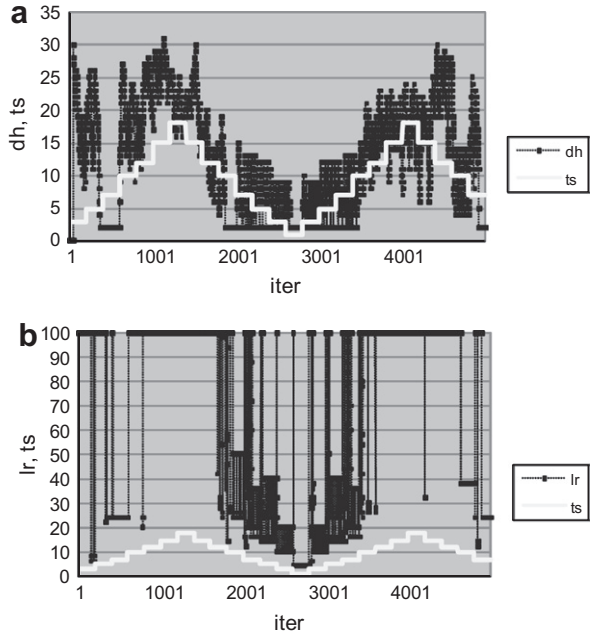

Fig. 1. Impact of changes in tabu tenure *ts* on: (a) Hamming distance (d_h) and (b) repetition length (l_r).

We observed this kind of behavior in all of our experiments. Since both d_h and l_r responded to changes in *ts*, we concluded that tabu tenure would be a good candidate for a control input variable, and we selected it for our SC-Tabu algorithm.

3.2.1.3. Selection of controlled variable. The next step was to decide whether the Hamming distance was a good candidate for a controlled output variable. While Fig. 1a shows that the value of this variable was driven by the change in *ts* even to its highest possible value (i.e., 35), Fig. 1b indicates that at low values of d_h, the search kept jumping between identical solutions that were a specific number of iterations apart. This means that when the Hamming distance was kept relatively low using a small *ts* value, the algorithm was performing unnecessary steps and was simply wasting time. The Hamming distance measurement alone did not allow us to conclude whether a search was intense, or simply repeating previously found solutions. Taking all these aspects into account, the Hamming distance was not selected as the controlled variable.

Fig. 1b shows that when the tabu tenure decreased, repetition length decreased sharply. The decrease in repetition length shows that the search was trapped around a local optimum and the tabu tenure was too small to divert the search from that point. For very small *ts* values such as $ts = 1$, the search was quickly trapped in a local optimum and the l_r dropped to 4, which meant the search was finding the same solution every 4 iterations. For *ts* values of around 12, the l_r varied between 20 and 100 and as *ts* values increased, the l_r reached the maximum value of 100, which meant there were no repetitions within the solutions found in the previous 100 iterations. As a result of these experiments, we selected repetition length l_r as the measured output. Tabu tenure's effectiveness in controlling repetition length further confirmed our choice of *ts* as the control input.

3.2.2. The controller for intensification

After the control variables were identified, the feedback control system elements (discussed in Section 2.3) were able to be determined: *Target System (Plant):* Tabu search algorithm; *Control Input:* Tabu tenure (*ts*); *Controlled Output:* Repetition length (l_r); *Reference Input:* The desired repetition length $\left(l_r^{ref}\right)$; *Control Error:* The difference between the desired repetition length and the measured output $\left(e = l_r^{ref} - l_r\right)$. The only remaining element left to identify was the *controller*.

We decided to use an *integral controller*, which is one of the most popular control mechanisms in feedback control theory. We selected it primarily because, as seen in Fig. 1, repetition length is a characteristic that varies significantly between two consecutive iterations. Consequently, the error would exhibit the same kind of behavior. In integral control, the change in the controller output is proportionate to the integral of all the past errors. Thus, the integral controller takes behavior history, rather than instantaneous error variations into account when computing the controller output. The integral controller can therefore "smooth out" variation effects (Franklin et al., 2002; Hellerstein, Diao, Parekh, & Tilbury, 2004).

The classical integral control law is defined as in Eqs. (5) and (6) (Franklin et al., 2002; Hellerstein et al., 2004):

$$e(k) = y^{ref} - y(k-1) \tag{5}$$

$$u(k) = u(k-1) + K_i e(k) \tag{6}$$

where y is the controlled output, y^{ref} is the reference input, k is the time step, e is the control error, u is the control input and K_i is the controller's gain parameter.

When we placed our control input and controlled output variables into the integral control equations, the control law for our system became:

$$e(k) = l_r^{ref} - l_r(k-1) \tag{7}$$

$$ts(k) = ts(k-1) + K_i e(k) \tag{8}$$

where k is the iteration number and $ts(0) = 1$. The values for the two parameters of the controller (the reference value l_r^{ref} and the gain parameter K_i) must be selected.

Continuing a search with few repetitions is possible when a high reference value l_r^{ref} is selected. This could be achieved by the high values of the control variable *ts*, but then intensification would not be achieved. The aim of the system is to conduct an intensified search while limiting the repetitions. On the other hand, for low values of l_r, a search becomes inefficient due to the fact that the same solutions are found repeatedly. Therefore, the value of l_r^{ref} should be selected so that it does not drive the search into either of those two extremes. The size of the problem should also be considered while determining l_r^{ref}. Problems with small size have smaller neighborhoods than large ones. Therefore, frequent repetitions can be considered more acceptable in small-sized problems. In order to make the reference repetition length proportional to the size of the problem, we used $l_r^{ref} = n$, where n is the number of units/locations. The size of the repetition length measurement window was set as $2 \cdot l_r^{ref}$. If the current solution was not found within the window, the value of l_r was set to the window size.

Given a specific value of l_r^{ref}, the controller will compute the control input *ts* based on the control error $l_r^{ref} - l_r$. The control behavior will be symmetric (i.e., the amount of change in *ts* will be the same for both positive and negative values of the control error), which is undesirable. Instead, it is better to eliminate repetitions with small l_r quickly and move toward small values of l_r slowly. A controller with such asymmetric behavior helps the system escape the frequent repetition region within a couple of iterations and prevents it from returning to that region too quickly.

In order to obtain this asymmetric behavior, an adaptation loop was added to the integral controller. Adaptive controllers were previously used in studies for achieving asymmetric behavior (Wang, Zhu, & Singhal, 2005; Zhu, Wang, & Singhal, 2006). We designed an adapter that modifies the controller's gain K_i as a function of l_r^{ref} and l_r using Eq. (9). Fig. 2 shows the asymmetric behavior achieved for a problem of size $n = 30$.

$$K_i(k) = \alpha(k) \cdot l_r^{ref} / l_r(k-1) \tag{9}$$

where

$$\alpha(k) = a + b/\left(1 + \exp(l_r(k-1) - l_r^{ref})\right) \tag{10}$$

The rate of increase and decrease for *ts* depends on the values of the parameters a and b. To determine the parameter values in Eq. (10), we performed an off-line parameter tuning on a small set of benchmark problems for the QAP, and selected $a = 0.001$, $b = 0.06$.

The behavior of this adaptive controller is illustrated in Fig. 3, which presents changes in the control input *ts* and the measured output l_r. As shown, *ts* starts at 1, and after a few iterations, repetitions appear. Because the repetition length is very low, the controller drives tabu tenure to a high value. As the repetitions continue, the *ts* value increases. When repetitions are nonexistent, and when they are above the reference level, *ts* decreases slowly, until repetitions below the reference level appear again. This causes the value of *ts* to increase. This behavioral pattern continues until the supplementary diversification mechanism (presented in Section 3.3) is activated, and starts again after diversification period.

3.3. Supplementary diversification mechanism

In this section, we discuss the diversification mechanism that encourages exploratory behavior during search. This diversification mechanism encourages the use of moves that have not been selected recently, by adjusting the tabu tenure *ts*. The mechanism uses search history to determine both the timing of diversification and its duration.

The supplementary diversification mechanism is composed of two elements: the observer and diversifier. The observer divides the intensification phase into *observation periods* and studies how the search uses the available moves at the end of each period. The observation period size is equal to the number of possible moves, so that the search is given enough time to use all the moves. If the observer monitors that search is not using the moves effectively, it activates the diversifier. In the diversification phase, the diversifier adjusts the tabu tenure *ts* as ordered by the observer. The actions of the observer and diversifier are explained in detail as follows.

The observer starts monitoring the search at the end of the first *stagnation period*. A stagnation period is defined as a period of iterations without any solution improvement (Misevicius, 2005). The observer is activated when the length of the stagnation period exceeds the stagnation threshold. The value of the stagnation threshold is selected to be equal to the observation period, p_o. When the observer is activated, it records the iteration number when stagnation started as the observation point *observePoint*, and it counts how many of the available moves have been used since the iteration *observePoint*. Then, after each observation period, the observer checks whether the number of unused moves since *observePoint* is smaller than at the end of the previous observation period. If the rate of decrease is smaller than 15%, the diversification phase starts (i.e., the diversifier is activated).

Once the diversification phase starts, the observer and adaptive control loop are turned off. In this phase, the diversifier forces

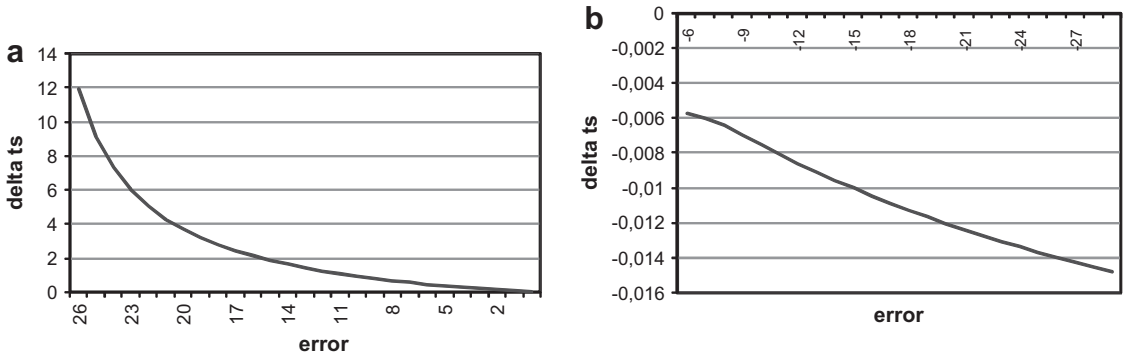

Fig. 2. Controller behavior: (a) Change in ts when tracking error is positive. (b) Change in ts when tracking error is negative.

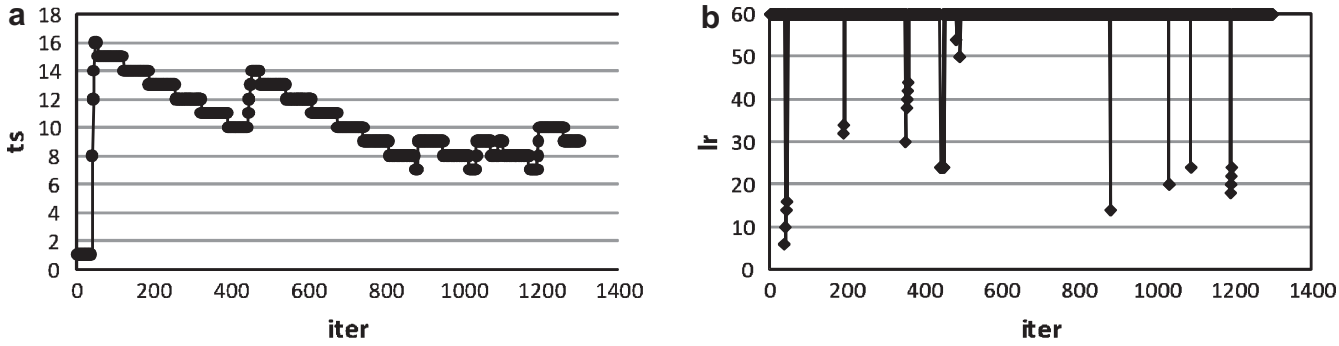

Fig. 3. Adaptive control behavior: (a) Tabu tenure vs. iterations. (b)Repetition length vs. iterations.

moves that have not been used since the beginning of the observation point *observePoint* by increasing the tabu tenure *ts* such that the moves that have been used are forbidden.

The length of the diversification period is equal to the number of unused moves. Within the diversification period, the algorithm periodically decreases *ts* to a low value (one third of the problem size) to allow some of the old moves to be combined with new ones. Once the diversification period ends, both the observer and the adaptive control loop are turned on again. The observer records the iteration number when diversification ended as *observePoint*, and starts monitoring the search every p_o iterations.

The supplementary diversification strategy presented in this paper has a similarity with the frequency-based long-term memory of tabu search. Both strategies promote selection of unused moves. However, they use different methods to achieve this. Frequency-based memory does it by adding a penalty to the objective function value, which is proportionate to the frequency of the selected move. This way, frequently selected moves are penalized continuously, or for a pre-specified period. In the supplementary diversification mechanism presented in this paper, diversification is achieved by adjusting the tabu tenure *ts*. The frequency of moves is not counted. Diversification starts only if the rate of decrease for unused moves is below a threshold. The length of the diversification period is determined according to information coming from the search history.

3.4. Self Controlling Tabu Search algorithm

The structure of Self Controlling Tabu Search (SC-Tabu) is shown in Fig. 4. The algorithm has three loops. The basic feedback loop and the adaptation loop, which control intensification, were discussed in Section 3.2.2, and the diversification loop was discussed in Section 3.3. At each iteration, either the adaptive controller, or the diversifier computes the tabu tenure *ts* for the next iteration. Fig. 5 illustrates how the control of tabu tenure transfers between the adaptive controller and the diversifier. The tabu search gets the tabu tenure value as an input, makes the best move and returns the repetition length l_r of the current solution.

The Self Controlling Tabu Search algorithm for the Quadratic Assignment Problem is presented below. The observation period p_o for QAP is equal to the number of possible moves, which is $n \cdot (n-1)/2$, where n is the number of units/locations. The iteration number at which a unit last moved into a location is recorded in the tabu list *tl*. The tabu list in this algorithm stores the iteration numbers at which allocations are made. A"move" in the pseudo code below is interpreted as an allocation. To determine whether or not a move is old, the iteration number recorded in the tabu list is used. The explanation for the parameters in the below algorithm are as follows: *ts* – tabu tenure, *tl* – tabu list, l_r – repetition length, n problem size, $iter_{adp}$- iteration counter for adaptation loop, $iter_{div}$ – iteration counter for diversification loop, $iter_{cur}$ – current iteration, $iter_{best}$ – iteration number that best solution was found, l_{stag} – stagnation length, d_{length} – diversification period length, d_{start} – tabu tenure at the beginning of diversification, *movesNotUsed* – number of unused moves, *prevMovesNotUsed* – previous number of unused moves, *observePoint* – iteration number observation starts

Self Controlling Tabu Search

Step 1.
Initialize:
$ts \leftarrow 1$
$p_o \leftarrow n \cdot (n-1)/2$
$iter_{cur} \leftarrow 0$
$iter_{best} \leftarrow 0$
Step 2.
Search with the adaptive control loop until the end of the stagnation period:
$l_{stag} \leftarrow 0$
$iter_{adp} \leftarrow 0$
while $l_{stag} < p_o$

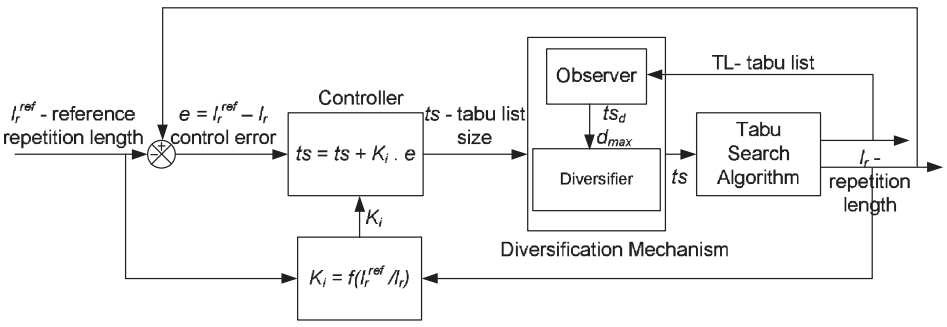

Fig. 4. Self Controlling Tabu Search mechanism.

$l_r \leftarrow$ tabuSearch(ts)
$ts \leftarrow$ adaptiveControl(ts, l_r)
$iter_{adp} \leftarrow iter_{adp} + 1$
$iter_{cur} \leftarrow iter_{cur} + 1$
$l_{stag} \leftarrow iter_{adp} - iter_{best}$
end while
$observePoint \leftarrow iter_{cur} - l_{stag}$

Step 3.
Count unused moves since the observation point:
$movesNotUsed \leftarrow$ countOldMoves($observePoint, tl$)

Step 4.
Continue search with adaptive control loop for p_o iterations:
$iter_{adp} \leftarrow 0$
while $iter_{adp} < p_o$
$l_r \leftarrow$ tabuSearch(ts)
$ts \leftarrow$ adaptiveControl(ts, l_r)
$iter_{adp} \leftarrow iter_{adp} + 1$
$iter_{cur} \leftarrow iter_{cur} + 1$
end while

Step 5.
Count unused moves since iteration *observePoint* and compute rate of decrease of unused moves:
$prevMovesNotUsed \leftarrow movesNotUsed$
$movesNotUsed \leftarrow$ countOldMoves($observePoint, tl$)
$rateOfDecrease \leftarrow \frac{(prevMovesNotUsed - movesNotUsed)}{prevMovesNotUsed}$
If *rateOfDecrease* < 15% and *prevMovesNotUsed* > 0
go to step 6
else
go to step 4
end if

Step 6.
Start diversification:
$d_{length} \leftarrow movesNotUsed$
$d_{start} \leftarrow iter_{cur} - observePoint$
$iter_{div} \leftarrow 0$
while ($iter_{div} < d_{length}$)
$ts \leftarrow$ diversifier($d_{start}, iter_{div}, n$)
tabuSearch(ts)
$iter_{div} \leftarrow iter_{div} + 1$
$iter_{cur} \leftarrow iter_{cur} + 1$
end while

Step 7.
End of diversification:
$ts \leftarrow 1$
$observePoint \leftarrow iter_{cur}$
$movesNotUsed \leftarrow 0$
go to step 4

Function adaptiveControl(ts, l_r)
Function returns the *ts* to be used in next iteration during the adaptive control period
begin
$a = 0.001 \quad b = 0.06$
$l_r^{ref} \leftarrow 2n$
$\alpha = a + b/\left(1 + \exp\left(l_r - l_r^{ref}\right)\right)$
$K_i = \alpha \cdot l_r^{ref}/l_r$
$e = l_r^{ref} - l_r$
$ts = ts + K_i e$
return *ts*
end

Function diversifier($d_{start}, iter_{div}, n$)
Function returns the tabu tenure *ts* to be used in next iteration during the diversification period
begin
$relaxPeriod \leftarrow n/5$
$relaxFrequency \leftarrow 2n$
if $((iter_{div} + n) \bmod relaxFrequency) \in$ {1,. . .*relaxPeriod*}
then $ts \leftarrow n/3$
else $ts \leftarrow iter_{div} + d_{start}$
return *ts*
end

Function countOldMoves(*observePoint*, *tl*)
Function returns the number of moves which have not been made since *observePoint*
begin
for all moves
if $tl(move) <$ *observePoint*
$count \leftarrow count + 1$
end if
end for
return *count*
end

Function tabuSearch(ts)
Function gets *ts* as input, conducts one iteration of tabuSearch
The move type and aspiration condition is described in Section 3
Updates $iter_{best}$ when best solution found improves
Updates *tl*
Function returns the current solution's repetition length l_r

Fig. 6 illustrates the workings of SC-Tabu on the Tai30a problem, a QAP instance of size $n = 30$. The figure shows that, in this

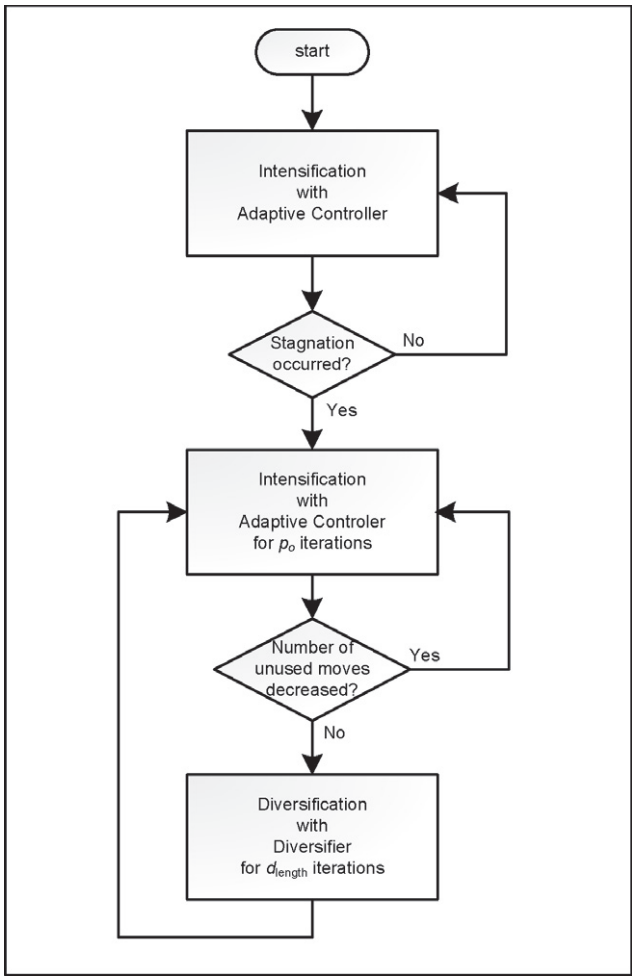

Fig. 5. Transfer of control between adaptive controller and diversifier.

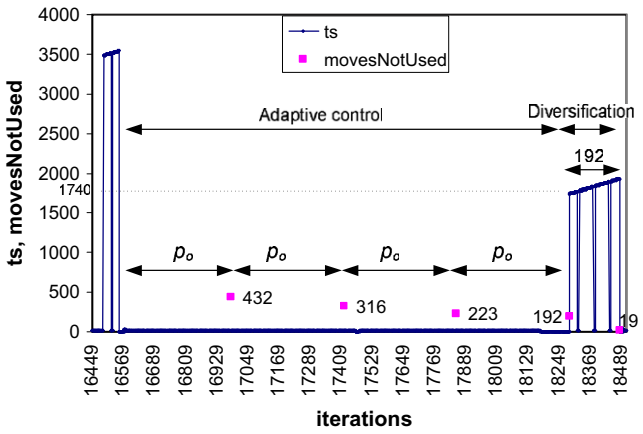

Fig. 6. Observation and diversification.

run of the SC-Tabu algorithm, a diversification period is followed by four observation periods and one more diversification period. The first diversification period in the figure ends at iteration 16,551, at that point the observer records this iteration number as *observePoint* and the adaptive controller starts. At the end of each observation period, the rate of decrease in unused moves is measured. The number of unused moves since *observePoint*, as indicated in the figure, are 432, 316, 223 and 192. At the end of the fourth period, the rate of decrease becomes $(223 - 192)/223 = 14\%$ (i.e., smaller than 15%). At this point, (iteration 18,291), the diversification phase starts. To enforce the use of new moves, the tabu tenure is increased to $d_{start} = 1740$, which is equal to the difference between the current iteration number and *observePoint* (18,291 − 16,551 = 1740). Since the size of the interval grows by one with each iteration, and it is important to ensure that the moves made since the end of the last diversification period stay tabu, *ts* is increased by one along with the interval. The length of the diversification period d_{length} is equal to the last number of *movesNotUsed*, which in this case is 192. The tabu tenure *ts* is dropped to $n/3$ periodically to let old moves combine with new ones. This shows up in the figure as three drops in *ts* to the value of 10. After the diversification, there are only 19 unused moves. At the end of the diversification period, the adaptive integral controller takes over and continues the search with an initial *ts* value of 1.

Fig. 7 illustrates the behavior of SC-Tabu on different kinds of problems. Referring to the classification of the QAP problems discussed in Section 2.4, Fig. 7a illustrates the behavior of SC-Tabu on a random uniform problem, and Fig. 7b demonstrates its behavior on a real life like problem. As we can see, the algorithm behaves differently on different problems. In the case of random problems, the diversification periods are very short and less frequent, while for real life like instances, the diversification periods are longer and more frequent. The importance of diversification for real life problems, has been recognized in the literature (cf. Taillard, 1995). SC-Tabu exhibits longer and frequent diversification periods due to built-in self-adaptation mechanisms.

4. Results

There have been several heuristic algorithms developed for the Quadratic Assignment Problem. Among the best performing algorithms are Battiti and Tecchiolli's Reactive Tabu Search (Re-Tabu) (Battiti & Tecchiolli, 1994), Taillard's Robust Tabu Search (Ro-Tabu) (Taillard, 1991), Misevicius' Enhanced Tabu Search (ETS) and Improved Hybrid Genetic Algorithm (IHGA) (Misevicius, 2005, 2004), and Drezner's Concentric Tabu Search (Drezner, 2005). These algorithms achieve successful results by using different methods. Reactive Tabu Search (Battiti & Tecchiolli, 1994) changes the tabu tenure reactively during the search and uses random moves for diversification. Robust Tabu (Taillard, 1991) changes the tabu tenure dynamically and introduces an additional aspiration strategy. Enhanced Tabu Search (Misevicius, 2005) uses different parameter values for different problem types and employs mutation strategies during the search. Improved Hybrid Genetic Algorithm (Misevicius, 2004) combines genetic algorithms and tabu search. Concentric Tabu Search (Drezner, 2005) uses a strategy similar to the variable neighborhood strategy.

The Self Controlling Tabu Search algorithm introduces the use of control theory principles and a search history-dependent diversification strategy to adjust the tabu tenure. In order to assess the performance of these strategies, we needed to compare SC-Tabu with algorithms using the same neighborhood strategies and differing from SC-Tabu in the way they change the tabu tenure. The ETS and IHGA algorithms are not suitable for comparison because of their use of mutation strategies, and Concentric Tabu Search is not suitable because it uses a different neighborhood strategy. Therefore, we selected Re-Tabu and Ro-Tabu as our comparison algorithms.

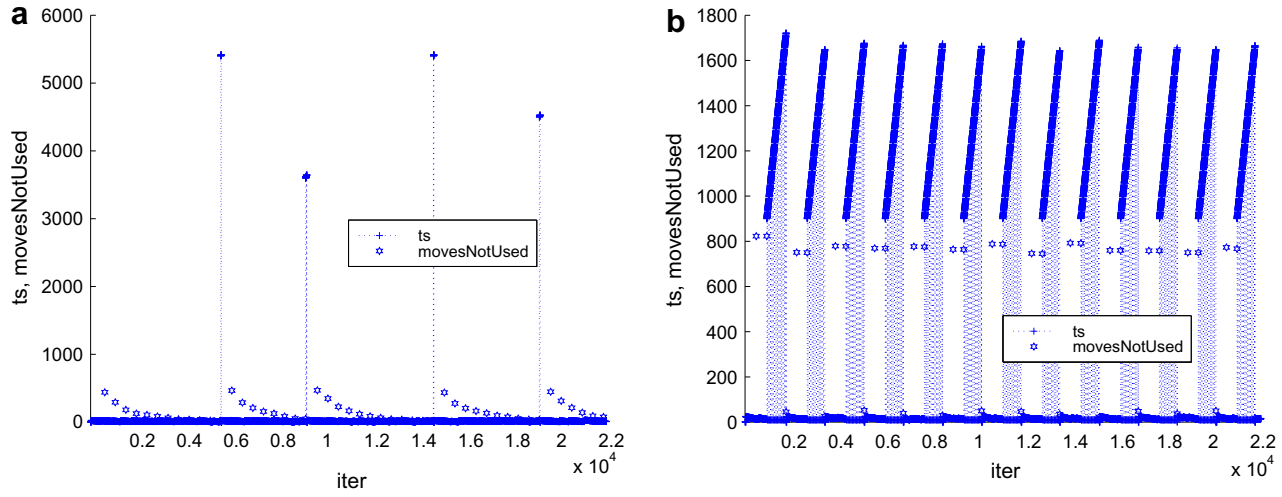

Fig. 7. SC-Tabu diversification on different problem types: (a) SC-Tabu behavior on a QAP – random uniform distribution problem. (b) SC-Tabu behavior on a QAP – real life like problem.

Self Controlling Tabu Search differs from the Reactive Tabu and Robust Tabu algorithms in how it changes the tabu tenure *ts* to control the intensification and diversification of the search. The basic differences between these three algorithms are as follows: Reactive Tabu Search increases the tabu tenure by a constant factor if the current solution has been found within last fifty iterations. If there are no repetitions for a number of iterations (equal to the moving average of the repetition length), the tabu tenure is decreased by a constant factor. If three solutions are found more than three times, Re-Tabu starts diversification and makes random moves for a random number of iterations (Battiti & Tecchiolli, 1994). Robust Tabu Search changes the tabu tenure between $0.9n$ and $1.1n$ randomly during the search, where n is equal to the size of the problem. For diversification, Ro-Tabu uses the following aspiration condition: If there exists a move which places the units in locations that they have not occupied within the last $n^2/2$ iterations, that move is selected (Taillard, 1991). The most important difference of SC-Tabu is use of a control-theoretic feedback mechanism during intensification and efficient use of search history during diversification. An adaptive integral controller adjusts the tabu tenure according to the control error during intensification. Diversification starts if the rate of decrease in the number of unused moves is below a threshold. The number of these moves determines the length of the diversification period. To achieve diversification, SC-Tabu increases the tabu tenure. The algorithm does not use random numbers. SC-Tabu only changes the tabu tenure, and all changes on the tabu tenure depend on the search history. As a result, the self-controlling features of SC-Tabu result in good performance on different problem types.

The performance of the SC-Tabu, Re-Tabu and Ro-Tabu algorithms were compared using Taillard's Type A and Type B, Skorin-Kapov, Steinberg, Elshafei, Burkard and Offemann, Wilhelm and Ward, Krarup and Pruzan, Eschermann and Wunderlich, Thonemann and Bölte, and Nugent, Vollmann and Ruml problems documented in the Quadratic Assignment Problem Library (Burkard et al., 2008). Taillard's Type A problems belong to the class of random uniformly generated problems and Taillard's Type B problems belong to the class of real life like problems. Steinberg, Elshafei, Burkard and Offemann, Krarup and Pruzan, Eschermann and Wunderlich problems belong to the real-life problem class. Skorin-Kapov, Wilhelm and Ward, Thonemann and Bölte, and Nugent, Vollmann and Ruml problems belong to the random flow on grids class. These problem sets have been widely used to compare the performances of different search algorithms (Battiti & Tecchiolli, 1994; Fleurent & Ferland, 1994; Taillard, 1995).

The tests were run on a Xeon EMT 64 processor with 2048 KB of cache and 4 GB of RAM.

4.1. Comparison of the quality of suboptimal results

Our primary goal with these experiments was to compare SC-Tabu with Re-Tabu and Ro-Tabu in terms of the quality of the suboptimal solutions with a limited time. To allow a direct comparison with the published results, the time limit for running the algorithms was selected to allow for the number of iterations specified by Taillard (1995). The performance measure was the average percent deviation from the objective function value of the best known solution. The average percent deviation is defined according to the formula $100 \times (obj_{avg} - obj_{best})/obj_{best}[\%]$ where obj_{avg} is the average of the objective function values of the best solutions found during each of the 30 runs, and obj_{best} is the objective function value of the best known solution (best known value – BKV). In Tables 1–3 column 1 shows the problem instance name, columns 2, 3 and 4 show the average percent deviation from the best known value for Re-Tabu, Ro-Tabu, and SC-Tabu, respectively (the best values are printed in bold face), column 5 shows the CPU time in seconds, and column 6 shows the best known value. The best known values were obtained from the Quadratic Assignment Problem Library (Burkard et al., 2008).

Table 1
Quality of suboptimal solutions on random uniform instances for Re-Tabu, Ro-Tabu and SC-Tabu.

Problem	Re-Tabu	Ro-Tabu	SC-Tabu	Time	BKV
Tai20a	0.312	**0.235**	0.246	0.53	703,482
Tai25a	0.247	0.304	**0.239**	1.87	1,167,256
Tai30a	0.186	0.326	**0.154**	4.48	1,818,146
Tai35a	0.310	0.546	**0.280**	10.77	2,422,002
Tai40a	**0.383**	0.898	0.561	11.98	3,139,370
Tai50a	0.953	1.105	**0.889**	14.05	4,938,796
Tai60a	**0.859**	1.278	0.940	24.67	7,205,962
Tai80a	**0.571**	0.964	0.648	60.34	13,515,450
Tai100a	**0.389**	0.826	0.977	119.42	21,052,466

Table 2
Quality of suboptimal solutions on real life and real life like instances.

Problem	Re-Tabu	Ro-Tabu	SC-Tabu	Time	BKV
Bur26a-h	0.560	0.035	**0.010**	0.54	5426670.8
Ste36a-c	0.817	**0.053**	0.761	5.11	2754829.3
Els19	7.836	9.177	**3.822**	0.08	17,212,548
Kra30a	0.254	**0.216**	0.714	2.88	88,900
Kra30b	**0.019**	0.038	0.178	2.88	91,420
Esc64a	**0**	0.805	1.207	0.04	116
Esc128	**0**	11.875	14.271	7.44	64
Tai20b	3.919	0.540	**0.335**	0.15	122,455,319
Tai25b	1.021	**0.011**	0.702	0.61	344,355,646
Tai30b	1.101	**0.307**	0.313	2.08	637,117,113
Tai40b	1.330	0.744	**0.219**	5.52	637,250,948
Tai50b	0.731	0.439	**0.281**	14.07	458,821,517
Tai60b	**0.366**	0.899	0.886	24.88	608,215,054
Tai80b	1.800	1.004	**0.798**	60.24	818,415,043
Tai100b	1.490	0.968	**0.553**	119.27	1,185,996,137
Tai150b	0.808	1.906	**0.648**	457.23	498,896,643

Table 3
Quality of suboptimal solutions on grid instances.

Problem	Re-Tabu	Ro-Tabu	SC-Tabu	Time	BKV
Nug30	0.56	0.023	**0.022**	2.88	6124
Wil50	0.25	**0.094**	0.105	2.85	48,816
Tho40	**0.044**	0.296	0.217	1.75	240,516
Sko42	0.48	0.036	**0.016**	8.3	15,812
Sko49	**0.068**	0.096	0.085	13.02	23,386
Sko56	0.145	0.090	**0.069**	20.22	34,458
Sko64	0.125	**0.063**	0.074	30.5	48,498
Sko72	**0.110**	0.181	0.159	43.73	66,256
Sko81	0.110	0.088	**0.076**	62.88	90,998
Sko90	0.164	0.179	**0.134**	86.33	115,534
Sko100a	0.317	0.135	**0.094**	119.23	152,002
Sko100b	0.298	0.123	**0.059**	119.23	153,890
Sko100c	0.269	0.094	**0.039**	119.23	147,862
Sko100d	0.354	0.138	**0.091**	119.23	149,576
Sko100e	0.304	0.111	**0.037**	119.23	149,150
Sko100f	0.367	0.166	**0.122**	119.23	149,036
Wil100	0.205	0.102	**0.099**	59.62	273,038
Tho150	**0.129**	0.186	0.173	305.6	8,133,398

Table 1 compares the performances of the algorithms using random uniform problem instances (Taillard's Type A problems). In these experiments, SC-Tabu and Re-Tabu were the best performing algorithms in four out of eight cases.

Table 2 compares the performance of the algorithms using real life and real life like problems. SC-Tabu shows better performance than both Re-Tabu and Ro-Tabu for these types of problems. The best performing algorithm is SC-Tabu in eight, Ro-Tabu in four, and Re-Tabu in four out of sixteen cases.

Table 4
Best solutions found by Re-Tabu, Ro-Tabu and SC-Tabu for large size problems.

Problem	Re-Tabu	Ro-Tabu	SC-Tabu	BKV
Tai40a	3,141,702	3,139,370	**3,139,370**	3,139,370
Tai50a	4,948,508	4,951,186	**4,938,796**	4,938,796
Tai60a	7,228,214	7,272,020	**7,205,962**	7,205,962
Tai80a	13,558,710	13,582,038	*13,551,152*	13,515,450
Tai100a	21,160,946	21,245,778	*21,147,572*	21,052,466

Table 3 compares the performances of the algorithms using grid instances. SC-Tabu shows better performance than both algorithms in twelve out of eighteen cases. Ro-Tabu performs better on two cases and Re-Tabu performs better on four cases. Overall, SC-Tabu shows better performance than both Re-Tabu and Ro-Tabu for these types of problems.

4.2. Comparison of best solutions found for large-size problems

In these experiments, Re-Tabu, Ro-Tabu and SC-Tabu were tested on large-scale type A problems. Table 4 presents the results. Columns 2 and 3 of this table show the best known solutions found by Re-Tabu and Ro-Tabu, respectively. The best solutions found by SC-Tabu are shown in column 4. Column 5 shows the best known value for the particular problem known in the literature.

The results show that SC-Tabu was able to find the best known solution in three out of five cases (printed in bold face). For the other two cases, although SC-Tabu had not found the best known solution, it still found better solutions than the ones found by Re-Tabu and Ro-Tabu (printed in italic).

5. Conclusions

In this paper, we presented a reactive tabu search algorithm, which was developed using the self-controlling software approach. The self-controlling software approach is a paradigm in which software is treated as a plant to be controlled in run time by feedback. To implement the self-control, we introduced two new strategies for reactive tabu search. The result was the Self Controlling Tabu Search (SC-Tabu) algorithm, which was then implemented and tested on the Quadratic Assignment Problem (QAP).

The first strategy applied control theory to the reaction mechanism. Control theory has been used in engineering to control dynamical systems and its application to software control has been the focus of many studies. We used a control-theoretic approach to adjust the algorithm parameters that affect search intensification by treating the tabu search algorithm as a plant to be controlled by a controller. The second strategy was a flexible diversification strategy which determined the time to start diversification and the length of the diversification period by using search history, and adjusted the algorithm parameters accordingly.

The results show that, the self-controlling features of SC-Tabu make it possible to adapt to different QAP problem classes and achieve good performance on them. In addition, use of control theory principles provided a systematic approach to the design of the reaction mechanism. In this study, we used a simple (I) control mechanism. Future studies could investigate the use of more sophisticated control architecture and controllers for the reaction mechanism. Instead of generating ad-hoc reaction methods, algorithm designers can adopt techniques from control theory (e.g., plant identification, sensitivity analysis, stability analysis).

References

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

Battiti, R., & Brunato, M. (2007). Reactive search: Machine learning for memory-based heuristics. In T. F. Gonzalez (Ed.), *Approximation Algorithms and Metaheuristics*. Washington, DC: Taylor & Francis Books (CRC Press).

Battiti, R., & Tecchiolli, G. (1994). The reactive tabu search. *ORSA Journal on Computing, 6*, 126–140.

Beer, S. (1981). *Brain of the firm* (2nd ed.). Chichester: John Wiley & Sons.

Birattari, M., Stutzle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of GECCO-02* (pp. 11–18).

Burkard, R., Çela, E., Karisch, S., & Rendl, F. (2008). Qaplib – A quadratic assignment problem library. <http://www.seas.upenn.edu/qaplib/>.

Drezner, Z. (2005). The extended concentric tabu for the quadratic assignment problem. *European Journal of Operational Research, 160*, 416–422.

Eiben, A. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation, 3*, 124–141.

Eracar, Y., & Kokar, M. (1997). An experiment in using control techniques in software engineering. In *Proceedings of the 1997 IEEE international symposium on intelligent control* (pp. 275–280).

Fleurent, C., & Ferland, J. A. (1994). Genetic hybrids for the quadratic assignment problem. *DIMACS Series in Mathematics and Theoretical Computer Science, 16*, 173–187.

Franklin, G. F., Powell, J. D., & Emami-Naemini, A. (2002). *Feedback control of dynamic systems*. Prentice-Hall.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. Freeman.

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

Glover, F. (1990a). Artificial intelligence, heuristic frameworks and tabu search. *Managerial and Decision Economics, 11*, 365–375.

Glover, F. (1990b). Tabu search-part 2. *ORSA Journal on Computing, 2*, 4–32.

Glover, F., & Laguna, M. (1998). *Tabu search*. Kluwer.

Hellerstein, J. L., Diao, Y., Parekh, S., & Tilbury, D. (2004). *Feedback control of computing systems*. John Wiley and Sons.

Herring, C. E. (2002). *Viable software the intelligent control paradigm for adaptable and adaptive architecture*. Ph.D. thesis, The University of Queensland.

Hertz, A., Taillard, E., & de Werra, D. (1997). *Local search in combinatorial optimization*. John Wiley and Sons.

Hoos, H. H. (2002). An adaptive noise mechanism for walksat. In *Proceedings of AAAI-02* (pp. 655–660).

Hutter, F., Hamadi, Y., Hoos, H. H., & Leyton-Brown, K. (2006). Performance prediction and automated tuning of randomized and parametric algorithms. In *Proceedings of principles and practice of constraint programming*.

Kai-Yuan, C., Cangussu, J., DeCarlo, R., & Mathur, A. (2003). An overview of software cybernetics. In *Proceedings of eleventh annual international workshop on software technology and engineering practice* (pp. 77–86).

Kauffman, S. A. (1993). *The origins of order self-organization and selection in evolution*. Oxford University Press.

Kephard, J., & Chess, D. (2003). The vision of autonomic computing. *IEEE Computer Magazine, 36*, 41–52.

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

Kokar, M. M., & Eracar, Y. (2000). An architecture for software that adapts to changes in requirements. *Journal of Systems and Software, 50*, 209–219.

Laddaga, R. (1999). Creating robust software through self-adaptation. *IEEE Intelligent Systems and Their Applications, 14*, 25–29.

Mattfeld, J., Bierwirth, C., & Kopfer, H. (1999). A search space analysis of the job shop scheduling problem. *Annals of Operations Research, 86*, 441–453.

Misevicius, A. (2004). An improved hybrid genetic algorithm: New results for the quadratic assignment problem. *Knowledge-Based Systems, 17*, 65–73.

Misevicius, A. (2005). A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications, 30*, 95–111.

Robertson, P., & Laddaga, R. (2004). The grava self-adaptive architecture: history; design; applications; and challenges. In *Proceedings of 24th international conference on distributed computing systems workshops* (pp. 298–303).

Sahni, S., & Gonzalez, T. (1976). P-complete approximation problems. *Journal of the ACM, 23*, 555–565.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing, 17*, 443–455.

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

Taleb-Bendiab, A., Bustard, D., Sterritt, R., Laws, A., & Keenan, F. (2005). Model-based self-managing systems engineering. In *Proceedings of sixteenth international workshop on database and expert systems applications* (pp. 155–159).

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

Wang, Z., Zhu, X., & Singhal, S. (2005). Utilization vs. slo-based control for dynamic sizing of resource partitions. In *Proceedings of 16th IFIP/IEEE distributed systems: Operations and management*.

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

Zhu, X., Wang, Z., & Singhal, S. (2006). Utility-driven workload management using nested control design. In *Proceedings of 2006 American control conference*.