ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2006, Vol. 35, No. 3

ITERATED TABU SEARCH: AN IMPROVEMENT TO STANDARD TABU SEARCH❖

Alfonsas Misevicius, Antanas Lenkevicius, Dalius Rubliauskas

Department of Multimedia Engineering, Kaunas University of Technology
Studentu St. 50, LT-51368 Kaunas, Lithuania

Abstract. The goal of this paper is to discuss the tabu search (TS) meta-heuristic and its enhancement for combinatorial optimization problems. Firstly, the issues related to the principles and specific features of the standard TS are concerned. Further, a promising extension to the classical tabu search scheme is introduced. The most important component of this extension is a special kind of diversification mechanism. We give the paradigm of this new improved TS strategy, which is called an iterated tabu search (ITS). ITS was applied to the difficult combinatorial optimization problems, the traveling salesman problem (TSP) and the quadratic assignment problem (QAP). The results of the experiments with the TSP and QAP show the high efficiency of the ITS strategy. The outstanding performance of ITS is also demonstrated by the fact that the new record-breaking solutions were found for the hard QAP instances – tai80a and tai100a.

Keywords: combinatorial optimization, heuristics, meta-heuristics, tabu search, iterated tabu search, quadratic assignment problem, traveling salesman problem.

Introduction

Complex optimization problems arising from both practice and theory pose a real challenge. These problems and their solution techniques attract the attention of the researchers around the world over several past decades. Many optimization problems belong to the class NP-hard and cannot be solved to optimality within polynomially bounded computation time. One of the ways to overcome such difficulties is to use the heuristic (local search) algorithms, i.e. the intelligent procedures (based upon human's intuition or nature inspired) [1,21,26] that seek for near-optimal solutions at reasonable computational time – but can not guarantee that a problem will be solved in terms of obtaining the exact solution. The heuristic approaches have an essential advantage over exact algorithms: the heuristics usually find high quality solutions much more faster than the exact algorithms; this is especially true by solving the large-scale problems.

Dozens of the improved heuristic algorithms appear every year. Nevertheless, the design of more elaborated, inventive and efficient variants of the existing methods – like simulated annealing, tabu search, genetic algorithms – as well as the creation of innovative optimization paradigms is still a matter of experience [12]. In this paper, the issues related, namely, to the new strategies for solving the difficult combinatorial (discrete) optimization (CO) problems are discussed. The focus is on the modern intelligent optimization technique, the well-known tabu search (TS) method. The concept of the tabu search was introduced by Hansen and Jaumard [14] and Glover [10,11]. Since that time, TS has been proven to be among the most powerful tools for solving various combinatorial optimization problems (for example, graph partitioning [28], quadratic assignment problem [32], scheduling [34], vehicle routing problem [35]). Still, the design of even more effective TS modifications for the specific problems is the research direction for many scientists. One of the possible extensions over the standard tabu search paradigm is a so-called iterated tabu search (ITS) we propose in this work. The approach we are going to discuss is not a pure heuristic. It is rather a universalized principle than the algorithm designed for a single problem's special benefit. We tried the preliminary variant of the ITS method on the well-known CO problems, the traveling salesman problem (TSP) and the quadratic assignment problem (QAP). However – after adding the corresponding modifications – ITS may be easily applied to other related problems.

The remaining part of this paper is organized as follows. The basic definitions of CO problems are introduced below. In Section 2, the descent local

❖ This work is supported by Lithuanian State Science and Studies Foundation through grant number T-06276.

search technique – as a basis for other developments – is briefly described. Then, the standard tabu search is outlined. The extension of the standard TS – an iterated tabu search – is discussed in Section 4. The basic characteristics and the template of ITS are given. In Section 5, we present the experimental results for the TSP and QAP, which demonstrate quite promising efficiency of the ITS technique. Finally, Section 6 completes the paper with conclusions.

1. Preliminaries

Before starting the next section, we introduce some very basic definitions related to combinatorial optimization. A combinatorial optimization problem P can be defined by a pair (S, f), where $S = \{s_1, s_2, ...\}$ is a finite (or possibly countable infinite) set of feasible solutions (a "solution space") and f: $S \rightarrow R^1$ is a real-valued objective (cost) function. Without loss of generality, we assume that f seeks a global minimum. For the sake of more clarity, let us consider the case where the solutions are permutations of the integers from 1 to n, that is $S = \{s \mid s = (s(1), s(2), ..., s(n)), s(i) \in \{1, 2, ..., n\}, i = 1, 2, ..., n, s(i) \neq s(j), i, j = 1, 2, ..., n, i \neq j\}$; where s is the permutation, $s(i)$ denotes the i-th element (item) of the permutation, and n is the problem size.

Thus, to solve the CO problem one has to search for a solution $s_{\text{opt}} \in S$ such that

$$s_{\text{opt}} \in S_{\text{opt}} = \left\{ s^{\nabla} \mid s^{\nabla} = \arg\min_{s \in S} f(s) \right\}. \qquad (1)$$

The solution s_{opt} is called a globally optimal solution (global optimum) of (S, f) and $S_{\text{opt}} \subseteq S$ denotes the set of global optima. It is very important for the heuristic algorithms that some appropriate neighbourhood structure of the solutions is defined. A neighbourhood function N: $S \rightarrow 2^S$ assigns for each $s \in S$ a set $N(s) \subseteq S$ – the set of neighbouring solutions (neighbours) of s (or simply the neighbourhood of s). As long as we operate upon permutation based solutions, we can define the neighbourhood function N_l of order l $(1 < l \leq n)$ in the following way: $?_l(s) = \{s' \mid s' \in S,\ r(s, s') \leq l\}$, where s is a solution from S and $r(s, s')$ is a "distance" between the solutions s and s'. An example of defining the "distance" is counting the elements that are assigned to different positions of the solutions, i.e. $r(s, s') = |\{i \mid s(i) \neq s'(i)\}|$ (see also Sections 5.1 and 5.2). Each solution $s' \in N(s)$ can be reached from s by an operation called a move, and s is said to move to s' when such an operation is performed (often, the move follows the objective function evaluation which is called a trial). Formally, the move appears as a transformation (perturbation) operator f: $S \rightarrow S$ such that $f(s) \in N(s)$, $\forall s \in S$.

2. Descent local search

The very early origins of the tabu search method go to the well-known technique – the descent local search (DLS) (also known as hill climbing) [25]. DLS starts from an initial (maybe, randomly generated) solution $s°$. Further, the search process is continued by performing the perturbations of solutions, i.e. making moves from solutions to solutions. A move is applied to the current solution s in order to get a new solution s' from the neighbourhood of the current solution, $N(s)$. The moves are controlled, i.e. decisions about to move to the neighbouring solutions, or not, are taken depending on the qualities of solutions (the objective function values f). If the decision is "positive", then the current solution is replaced by the neighbouring one, which will be used as a "starting point" for the subsequent trials; otherwise, the search is continued with the current solution. In classical DLS algorithms, the decision is "positive" if only the new solution is definitely better than the current one (i.e. the difference in the objective function values is negative $(\Delta f = f(s') - f(s) < 0$, where $s' \in N(s))$). The whole process is continued until the current solution s is locally optimal, that is, no better solution exists in the neighbourhood of the current solution (i.e. $f(s') \geq f(s)$, $\forall s' \in N(s)$). The descent local search paradigm (in PASCAL language like notation) is presented in Figure 1.

```
function DescentLocalSearch(s°);
  // input: s° – the initial solution; output: s – the locally optimal solution //
  s ← s°;
  while s not locally optimal do begin // main cycle //
    choose the solution s' from the neighbourhood of s, N(s),
    in such a way that f(s') – f(s) < 0;
    s ← s' // replace the current solution s by the new one s' (make a move to the new solution) //
  end; // while //
  return s
end.
```

Figure 1. Paradigm of descent local search

Given neighbourhood N_l, the solution obtained by DLS may be treated as an optimal solution with respect to this neighbourhood, i.e. l-opt(imal) solution – hence, the names of the corresponding procedures: 2-opt, 3-opt, and so on.

3. Standard tabu search

The tabu search framework [10,11] originates from the local search policy described above. However, the TS goes beyond this paradigm. In contrast to DLS, which is limited to finding one locally optimal solution only, TS enables to escape local optima. TS-based algorithms continue the search even if a locally optimal solution is encountered. Shortly speaking, TS is a process of chains of moves from one local optimum to another. The best local optimum found during this process is regarded as a result of TS. Thus, TS is an extended descent local search. Consequently, it explores much more larger part of the solution space when comparing with DLS. Hence, TS provides more room for discovering high quality solutions than the traditional DLS.

The key idea of TS is allowing climbing moves when no improving neighbouring solution exists, i.e. a move is allowed even if a new solution s' from the neighbourhood of the current solution s is worse than the current one. Naturally, the return to the locally optimal solutions previously visited is to be forbidden in order to avoid cycling. TS is based on the methodology of prohibitions: some moves are "frozen" (become "tabu") from time to time.

```
function TabuSearch(s°);
  // input: s° – the initial solution; output: s* – the best solution found; parameter: h – the tabu list size //
  s ← s°; s* ← s°;
  initialize the tabu list T;
  repeat // continue the main cycle of TS //
    given neighbourhood function N, tabu list T, and aspiration
    criterion, find the best possible solution s' ∈ N✧(s) ⊆ N(s),
    where N✧(s) consists of the solutions that (or their "attributes")
    are not in the tabu list T or satisfy the aspiration criterion;
    s ← s'; // replace the current solution by the new one //
    if f(s) < f(s*) then s* ← s; // save the best so far solution //
    insert the solution s (or its "attribute") into the tabu list T;
    if sizeof(T) > h then remove the "oldest" member of T
  until termination criterion is satisfied;
  return s*
end.
```

Figure 2. Paradigm of standard tabu search

More formally, the TS algorithm starts from an initial solution $s°$ in S. The process is then continued in an iterative way – moving from a solution s to a neighbouring one s'. At each step of the procedure, a subset $N^{\diamond}(s) \subseteq N(s)$ of the neighbouring solutions of the current solution is considered, and the move to the solution $s' \in N^{\diamond}(s)$ that improves most the objective function value f is chosen. Naturally, s' must not necessary be better than s: if there are no improving moves, the algorithm chooses the one that least degrades (increases) the objective function (a move is performed to the neighbour s' even if $f(s') > f(s)$). In order to eliminate an immediate returning to the solution just visited, the reverse move must be forbidden. This is done by storing the corresponding solution (move) (or its "attribute") in a memory (called a tabu list (T)). The tabu list keeps information on the last $h = |T|$ moves which have been done during the search process. Thus, a move from s to s' is considered as tabu if s' (or its "attribute") is contained in T. This way of proceeding hinders the algorithm from going back to a solution reached within the last h steps. However, the straightforward prohibition may sometimes lessen the efficiency of the algorithm. Moreover, it might be worth returning after a while to a solution visited previously to search in another promising direction. Consequently, an aspiration criterion is introduced to permit the tabu status to be dropped under certain circumstances. Usually, a move from s to s' (no matter its status) is permitted if $f(s') < f(s^*)$, where s^* is the best solution found so far. The resulting decision rule can thus be described as follows: replace the current solution s by the new solution s' if

$$f(s') < f(s^*) \text{ or } (s' = \arg\min_{s'' \in N^{\diamond}(s)} f(s'') \text{ and } s' \text{ (or "attribute" of } s' \text{) is not tabu}). \quad (2)$$

The search process is stopped as soon as a termination criterion is satisfied (for example, a fixed a priori number of iterations (trials) has been performed). The pseudo-code for the standard (simple)

tabu search paradigm is presented in Figure 2. More details on the fundamentals and principles of TS can be found in [8,13,15].

4. Iterated tabu search

TS is a powerful optimization tool. However, it typically faces, in its canonical form, less or more difficulties. They are as follows: a huge number of local optima over the solution space, presence of cycles (i.e. repeating sequences) of the search configurations (states), and the phenomenon of so-called "deterministic chaos" (or chaotic attractors). The last one can be characterized by the situation in which "getting stuck" in local optima and cycles are absent but the search trajectory is still confined in some "narrow region" of the solution space [2] (see Figure 3). So, the search trajectory will visit only a limited part of the solution space: if this portion does not contain the global minimum, it will never be found – a stagnation of the search is said to take place. Figure 4 depicts an example of such a situation.

Figure 3. Hypothetical view of the search trajectories

In order to try to overcome the difficulties mentioned, an essential extension of the standard TS – iterated tabu search[1] – can be proposed. It should be noted that several attempts to enhance the straightforward TS have been already made. One of the most famous enhancements is the reactive tabu search [2]. Nevertheless, we think of ITS as a, probably, more aggressive attempt. First of all, this is due to the new important features we discuss in this section.

The standard TS goes beyond the descent local search – similarly, ITS seeks to go beyond the standard TS. The central idea of ITS is the concept of intensification and diversification (I&D). The early origins of this concept go back to 1986 [3]. Since that time, various modifications of the basic idea have been proposed, among them: iterated Lin-Kernighan algorithm [16], combined local search (chained local optimization) [20], "ruin and recreate" principle [30], iterated local search [18].

Generally speaking, the I&D framework is distinguishing for three main factors (components): intensification, diversification, and candidate acceptance (selection) (see Figure 5).

The goal of intensification is the search for a better (locally optimal) solution in "surroundings", i.e. neighbourhood of the current solution. (Mathematically, intensification can be described as a special operator y: $S \to S$ such that $f(s) \le f(y(s))$, where s ($s \in S$) is the solution to be "intensified".) In the other words, one tries to improve the current solution as best as one can. If this improvement is performed by means of the classical tabu search, one just gets the ITS method. Intensification is always applied to the solution just reconstructed (i.e. the output of diversification), except the first iteration only at which intensification is applied to the initial solution (see Figure 5). It was revealed by experimentation that there is no need in the expensive runs of the tabu search based improvement procedure. Firstly, the short tabu search iterations allow saving considerable amount of CPU time. On the other hand, this limited tabu search in combination with the robust diversification operators is capable of seeking near-optimal solutions – better than those obtained by the long runs of the pure tabu search.

Diversification – it may be interpreted as a special sort of reconstruction (perturbation) of the solutions – is responsible for escaping from the current local optimum and moving towards new regions in the solution space. (Diversification can formally be defined by an operator z: $S \to S$ such that $z(s) \in S$ and $s \ne z(s)$, where s ($s \in S$) is the solution which undergoes the diversification process.). It is important that a proper level of diversification is kept up: if reconstruction is too strong, the resulting algorithm might be quite similar to a crude random (blind) multistart; if reconstruction is too weak, the process would periodically return to the solutions to which reconstruction has been applied.

Many different perturbation variants may be proposed. For example, for the solutions based on permutations, one can use the random pairwise interchanges which are the sequences of moves $f_{r_1 r_2}, f_{r_3 r_4}, \ldots, f_{r_{2m-1} r_{2m}}$. ($f_{r_i r_{i+1}}$ is a special case of the transformation operator (see Section 1), which interchanges the r_ith and r_{i+1}th elements in the current solution.) In this case, it is sufficient to simply generate the pairs of uniform random numbers (r_i, r_{i+1}), such that $1 \le r_i, r_{i+1} \le n$, $r_i \ne r_{i+1}$, $i = 1, 2, \ldots, m$ (n is the problem size). The larger the length of the sequence (i.e. the reconstruction level) m, the stronger the diversification effect, and vice versa. We achieve more robustness of the diversification process by letting the parameter m vary in some interval, say $[m_{\min}, m_{\max}] \subseteq [2, n]$. The following strategy of changing the values of m might be proposed. At the beginning, m is equal to $m_{\min}$; further, m is increased gradually, step by step, until some limit is reached; once the maximum level $m_{\max}$ has been reached (or,

[1] The term "iterated tabu search" is firstly mentioned in the paper by Smyth, Hoos, and Stützle [31].

possibly, a better local optimum has been found), the current value of m is immediately dropped to $m_{\min}$, and so on. In addition, if the best so far solution remains unchanged for a quite long time, then the value of $m_{\max}$ may be increased, too ($m_{\max}$ should be reset to the initial value as soon as a new local optimum has been found).

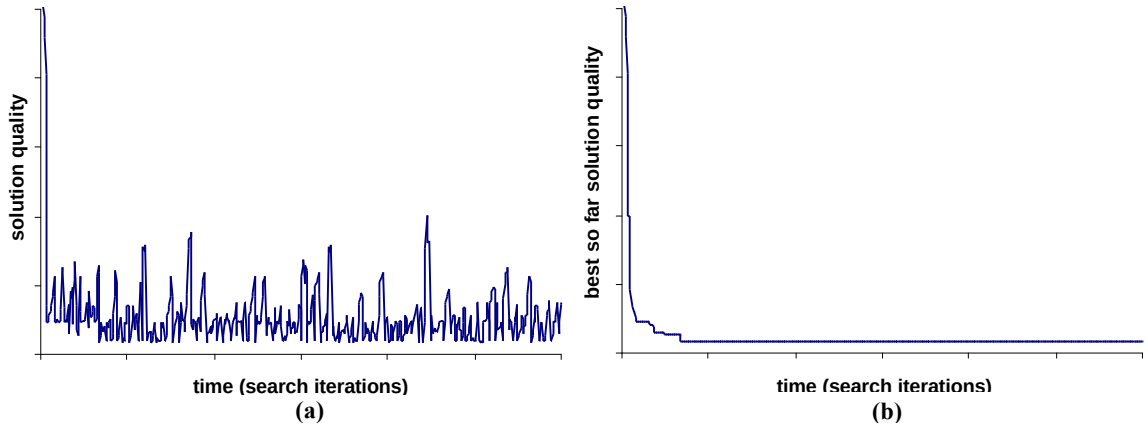

Figure 4. Illustration of the stagnation situation:
a) the detailed "history" is presented, i.e. points corresponding to the current objective function values are depicted;
b) only the best so far (record-breaking) values of the objective function are shown

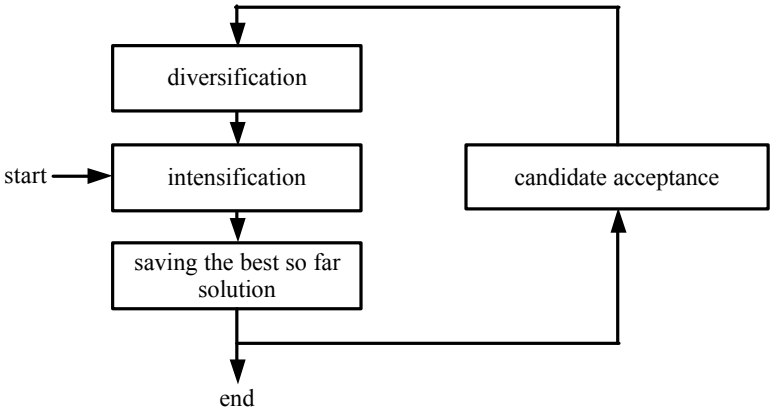

Figure 5. Generalized framework of I&D

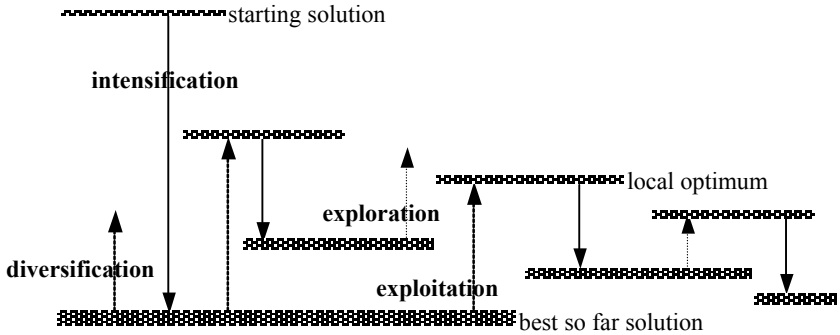

Figure 6. Towards the graphical interpretation of the I&D process

Regarding the selection of candidates for diversification, two main alternatives exist: a) exploitation and b) exploration. Exploitation is achieved by choosing only the currently best local optimum – the best so far (BSF) solution – as a candidate for reconstruction. In the case of exploration, a variety of policies may be used. In fact, each locally optimized solution (not necessary the best local optimum) can be considered as a potential candidate for diversification. Even generation of a new solution from scratch is possible as an extreme case. A so-called "where you are" (WYA) strategy is worth mentioning: in this case, every new local optimum (no matter its quality) is accepted for the reconstruction process. However, more

sophisticated strategies are available, for example, selection from a memory of locally optimal solutions, like in the population based (genetic) algorithms. (Exploitation/exploration could be formalized by introducing an operator x: $2^S \to S$. If exploitation is used, then the following equation must hold: $x(\cdot) = s^\bullet$, where $s^\bullet$ ($s^\bullet \in S$) is the BSF solution.)

A graphical illustration of the I&D process is shown in Figure 6.

The typical flow of the iterated tabu search process is as follows. ITS is initiated by the improvement of an initial solution by means of the traditional TS. As a result, the first optimized solution, say $s^\bullet$, is achieved. Further, a given solution undergoes perturbation, and a new solution, say $s^\sim$, is obtained. The goal of such a perturbation is not to destroy the current solution absolutely – on the contrary, it is highly desirable that the resulting solution inherits some characteristics of the previous local optimum, since parts of this optimum may be close to the ones of the globally optimal solution. The reconstructed solution $s^\sim$ serves as an input for the subsequent tabu search procedure, which starts immediately after the perturbation process is finished. The TS procedure returns the new optimized solution $s^\bullet$, which (or some other local optimum), in turn, is reconstructed, and so on. A new better solution (s^*) found during this iterative process is saved in a memory (as a potential resulting solution of ITS). This type of proceeding continues until a stopping condition is met, for example, a fixed number of iterations have been executed. The template of the ITS algorithm is shown in Figure 7.

```
function IteratedTabuSearch(s°);
  // input: s° – the initial solution; output: s* – the best solution found //
  s• ← TabuSearch(s°); // improve the initial solution s° by TS, get the resulting solution s• //
  s ← s•; s* ← s•;
  repeat // continue the cycle of the iterated tabu search //
    s ← CandidateAcceptance(s•,s, ...); // select a solution for reconstruction //
    s~ ← Reconstruction(s); // ruin the selected solution, obtain a new solution s~ //
    s• ← TabuSearch(s~); // improve the solution s~ by TS, get the resulting solution s• //
    if f(s•) < f(s*) then s* ← s• // save the best so far solution (as a possible result of ITS) //
  until termination criterion is satisfied;
  return s*
end.
```

Figure 7. Paradigm of iterated tabu search

5. Computational experiments

In order to evaluate the efficiency of the proposed iterated tabu search technique, some experiments have been carried out on the hard combinatorial optimization problems, the traveling salesman problem and the quadratic assignment problem.

5. 1. Experiments with the traveling salesman problem

The traveling salesman problem can be formulated as follows. Given the matrix $\boldsymbol{D} = (d_{ij})_{n\times n}$ and the set Π of permutations of the integers from 1 to n, find a permutation $p = (p(1), p(2), ..., p(n)) \in \Pi$ that minimizes

$$z(p) = \sum_{i=1}^{n-1} d_{p(i),p(i+1)} + d_{p(n),p(1)}. \qquad (3)$$

The interpretation of n, $\boldsymbol{D}$ and p is as follows: n is the number of cities; $\boldsymbol{D}$ is the matrix of distances between all pairs of these cities; $j = p(i)$ denotes city j to visit at step i. Usually, permutations are called tours, and the pairs $(p(1), p(2))$, ..., $(p(i), p(i+1))$, ..., $(p(n), p(1))$ are called edges. So, solving the TSP means searching for the shortest closed tour in which every city is visited exactly once. It has been proved that the TSP is NP-hard [7] and cannot be solved to optimality within polynomially bounded computation time.

The TSP is a representative example of CO problem (S, f), where $S \equiv \Pi$ and f corresponds to z. Regarding the neighbourhood function for the TSP-solutions, there are some specific things. The TSP-heuristics operate rather upon pairs of elements $(j_1 = p(i), j_2 = p(i+1))$ (i.e. edges) than single elements $(j = p(i))$. Taking this fact into account, the distance between two permutations (tours) is defined as the number of pairs of elements (edges) that are contained in the first permutation (tour) but not in the second one [4]. Thus, the distance between permutations p and p' may be declared as $r(p, p') = |\Omega|$, where Ω is the set that consists of all possible pairs $(p(i), p((i \bmod n)+1))$ ($i \in \{1, 2, ..., n\}$) such that $\exists j$:

$$(p(i), p((i \bmod n)+1)) = \begin{cases} (p'(j), p'(j+1)), 1 \le j < n \\ (p'(j), p'(1)), \; j = n \end{cases}$$

or

$$(p(i), p((i \bmod n)+1)) = \begin{cases} (p'(j), p'(j-1)), 1 < j \le n \\ (p'(j), p'(n)), \; j = 1 \end{cases}.$$

We can then easily define the neighbourhood function N_l of order l ($1 < l \le n$):
$N_l(p) = \{p' \mid p' \in \Pi,\ r(p,p') \le l\}$, where p is from Π. In the case of $l = 2$, a move from the current permutation p to the neighbouring one $p' \in N_2(p)$ may be described by using a perturbation operator $f(p, i, j)$: $\Pi \times ? \times ? \to \Pi$, which gives for each permutation the permutation that is obtained by removing the two edges at the ith and jth position and inserting two different edges. That is, the pairs $(p(i), p(i+1))$ and $(p(j), p((j \bmod n)+1))$ are deleted, and the pairs $(p(i), p(j))$ and $(p(i+1), p((j \bmod n)+1))$ are added. More specifically, $f(p, i, j)$ gives p' such that $p'(i) = p(i)$, $p'(i+1) = p(j)$, $p'(j) = p(i+1)$, $p'((j \bmod n)+1) = p((j \bmod n)+1)$, where $1 \le i, j \le n \wedge 1 < j-i < n-1$; in addition, if $j-i-2 \ge 1$, then $p'(i+k+1) = p(j-k)$ for every $k \in \{1, ..., j-i-2\}$.

Having the neighbourhood and solution perturbations defined, we implemented the ITS algorithm for the TSP based on the paradigm in Figure 7. For the comparison, the following algorithms were used: 1) the multi-start 2-opt (M-2-OPT) algorithm; 2) the 4-opt (4-OPT) algorithm; 3) the simulated annealing (SA) algorithm; 4) the standard tabu search (STS) algorithm. The test data are from the well-known library of the TSP instances TSPLIB [27]. The performance measures of the algorithms are: a) the average deviation of solutions from a provably optimal solution – $\bar{d}$ ($\bar{d} = 100(\bar{z} - z_{\text{opt}})/z_{\text{opt}}$ [%], where $\bar{z}$ is the average objective function value (tour length) over 10 runs (i.e. single applications of the algorithm to a given instance) and z_{opt} is the objective function value of the optimal solution (values z_{opt} are taken from [27])); b) the number of solutions that are within 1% optimality ($\bar{d} \le 1$) (over 10 runs) – $C_{1\%}$; c) the number of the optimal solutions found – C_{opt}. The results of the comparison are presented in Table 1.

Table 1. Comparison of the algorithms (Part I). The best results obtained are printed in bold face. CPU times per run are given in seconds. (900 MHz PENTIUM computer was used in the experiments)

Instance	n	z_{opt}	$\bar{d}$, $C_{1\%}/C_{\text{opt}}$, CPU time					
			M-2-OPT	4-OPT	SA	STS	ITS	
a280	280	2579	6.73, 0/0, *19*	—	0.03 10/9, *78*	2.04, 3/0, *140*	**0**	*25*
att48	48	10628	0.75, 6/0, *0.1*	1.59, 3/0, *1.0*	**0** *6.0*	0.85, 7/1, *0.8*	**0**	*0.1*
bayg29	29	1610	0.26, 10/4, *0.1*	1.44, 4/1, *0.1*	**0** *5.0*	**0** *0.2*	**0**	*0.0*
bays29	29	2020	0.03, 10/9, *0.1*	0.97, 6/0, *0.1*	**0** *5.0*	**0** *0.2*	**0**	*0.0*
berlin52	52	7542	0.63, 7/6, *0.2*	2.88, 1/0, *1.5*	**0** *7.0*	0.48, 8/8, *1.1*	**0**	*0.1*
bier127	127	118282	2.43, 0/0, *1.4*	1.97, 1/0, *170*	0.66, 5/5, *18*	2.51, 2/0, *12*	**0**	*1.5*
brazil58	58	25395	0.00, 10/7, *0.1*	0.94, 7/5, *2.9*	**0** *8.0*	**0** *1.4*	**0**	*0.2*
brg180	180	1950	8.82, 0/0, *3.9*	0.10, 10/8, *700*	16.30, 0/0, *36*	**0** *23*	**0**	*0.5*
burma14	14	3323	**0** *0.0*	0.57, 9/5, *0.0*	**0** *0.5*	**0** *0.1*	**0**	*0.0*
ch130	130	6110	2.64, 0/0, *1.6*	2.39, 1/0, *190*	0.27, 9/5, *19*	2.56, 0/0, *14*	**0**	*2.9*
ch150	150	6528	4.12, 0/0, *2.4*	2.51, 1/0, *400*	0.33, 10/1, *24*	1.06 6/1, *21*	**0**	*3.5*
d198	198	15780	2.22, 0/0, *6.0*	1.28, 4/0, *1800*	0.10, 10/2, *40*	0.37, 9/0, *42*	**0**	*40*
dantzig42	42	699	0.16, 9/8, *0.1*	0.36, 10/5, *0.5*	**0** *6.0*	0.07, 10/9, *0.6*	**0**	*0.0*
eil51	51	426	2.28, 0/0, *0.2*	1.83, 2/0, *1.6*	0.02, 10/9, *7.0*	0.09, 10/6, *0.9*	**0**	*0.5*
eil76	76	538	3.90, 0/0, *0.3*	2.36, 1/1, *17*	**0** *10*	**0** *2.7*	**0**	*0.4*
eil101	101	629	4.69, 0/0, *0.8*	2.99, 0/0, *48*	**0** *14*	0.18, 9/6, *5.9*	**0**	*1.2*
fri26	26	937	**0** *0.1*	0.38, 8/0, *0.0*	**0** *5.2*	**0** *0.2*	**0**	*0.0*
gil262	262	2378	5.13, 0/0, *17*	—	0.24, 10/0, *75*	2.94, 1/0, *145*	**0.00**, 10/9, *360*	
gr17	17	2085	**0** *0.0*	0.17, 10/4, *0.0*	**0** *2.0*	0.04, 10/8, *0.0*	**0**	*0.0*
gr21	21	2707	**0** *0.1*	1.77, 5/5, *0.1*	**0** *3.2*	**0** *0.1*	**0**	*0.0*
gr24	24	1272	**0** *0.1*	1.82, 6/4, *0.1*	**0** *4.9*	**0** *0.2*	**0**	*0.0*
gr48	48	5046	0.38, 10/0, *0.2*	1.00, 7/0, *6.0*	**0** *6.4*	0.20, 10/5, *0.8*	**0**	*0.1*
gr96	96	55209	2.05, 0/0, *0.7*	1.62, 5/0, *42*	0.20, 10/0, *14*	1.99, 3/0, *6.1*	**0**	*1.2*
gr120	120	6942	3.38, 2/0, *1.2*	3.16, 0/0, *120*	0.35, 10/0, *18*	0.42, 9/0, *10.6*	**0**	*6.9*
gr137	137	69853	2.73, 0/0, *2.0*	2.27, 2/0, *270*	0.15 10/2, *22*	1.17, 4/0, *16*	**0**	*2.4*
gr202	202	40160	4.13, 0/0, *6.0*	2.99, 0/0, *1900*	0.24, 10/2, *40*	2.23, 1/0, *50*	**0**	*160*
gr229	229	134602	4.10, 10/0, *9.0*	—	0.61, 9/0, *47*	2.50, 1/0, *78*	**0**	*190*
hk48	48	11461	1.27, 3/0, *0.1*	0.93, 6/3, *1.0*	0.46, 10/0, *4.8*	0.28, 9/5, *0.8*	**0**	*0.2*

It is obvious from the results that the iterated tabu search clearly outperforms the standard tabu search with respect to the performance measures used (first of all, the average deviation). In general, ITS produces definitely higher quality results than all the remaining algorithms tested, including the simulated annealing algorithm, which, unexpectedly, seems even to be better than the simple tabu search. The outstanding performance of ITS can also be confirmed by some indirect comparisons with other known algorithms. For example, in Knox's paper [17], a version of the standard tabu search based algorithm was proposed.

The author reports few results for small TSP instances ($n \leq 75$). The solutions are very close to optimal, however the CPU times are quite large (for example, more than 600 seconds are needed for the instance with 50 cities). In [16], there was introduced an improved variant of the famous Lin-Kernighan heuristic. This algorithm was able to produce solutions that are within about 0.8% optimality on instances of size $n \leq 100$. The other efficient implementation of Lin-Kernighan algorithm [19] yielded solutions that were from 0.24% to 3.04% of optimal solutions on instances of size 48 to 226. In a more recent work [9], the tabu search like algorithm (called a complete local search) could only find solutions that are from 1.13% to 8.62% far from optimal solutions for instances of size 52 to 200 (the CPU time reported is from 9.9 to 2206 seconds).

The efficiency of ITS can be improved even more by increasing the number of iterations (but at the cost of longer computation time) or tuning the values of other control parameters.

Table 1. Comparison of the algorithms (Part II). The best results obtained are printed in bold face. CPU times per run are given in seconds. (900 MHz PENTIUM computer was used in the experiments)

Instance	n	z_{opt}	$\bar{d}$, $C_{1\%}/C_{opt}$, CPU time				
			M-2-OPT	4-OPT	SA	STS	ITS
kroa100	100	21282	1.13, 5/0, *0.8*	0.56, 8/0, *48*	0.13, 10/5, *13*	2.62, 5/0, *6.7*	**0** *0.7*
kroa150	150	26524	3.64, 0/0, *2.4*	2.02, 1/0, *360*	0.06, 10/1, *24*	2.94, 1/0, *22*	**0** *14*
kroa200	200	29368	4.18, 0/0, *6.0*	2.85, 1/0, *1800*	0.41, 10/0, *39*	3.68, 0/0, *53*	**0** *11*
krob100	100	22141	2.03, 1/0, *0.7*	2.25, 3/0, *48*	0.09, 10/7, *14*	1.91, 2/0, *6.6*	**0** *0.9*
krob150	150	26130	2.88, 0/0, *2.5*	1.99, 0/0, *360*	0.19, 10/0, *25*	3.33, 0/0, *21*	**0** *8.5*
krob200	200	29437	4.71, 0/0, *6.0*	2.47, 0/0, *1900*	0.18, 9/0, *39*	5.00, 0/0, *54*	**0** *86*
kroc100	100	20749	1.81, 1/0, *0.8*	1.78, 4/1, *48*	0.03, 10/8, *13*	2.22, 2/0, *6.8*	**0** *0.8*
krod100	100	21294	2.37, 1/0, *0.8*	1.88, 2/0, *48*	0.07, 10/7, *13*	2.53, 2/0, *6.6*	**0** *0.9*
kroe100	100	22068	2.05, 0/0, *0.8*	1.43, 4/0, *48*	0.31, 10/0, *13*	1.01, 8/0, *6.7*	**0** *1.1*
lin105	105	14379	1.23, 3/0, *0.8*	1.97, 4/0, *60*	0.12, 19/7, *15*	3.13, 0/0, *7.7*	**0** *0.8*
lin318	318	42029	4.60, 0/0, *45*	—	0.83, 7/0, *120*	3.95, 0/0, *280*	**0.28**, 10/3, *180*
pr76	76	108159	0.91, 8/0, *0.4*	1.69, 3/0, *12*	**0** *10*	0.39, 9/0, *2.9*	**0** *0.3*
pr107	107	44303	0.90, 6/0, *0.9*	1.14, 5/0, *72*	**0** *14*	1.17, 9/1, *7.3*	**0** *0.8*
pr124	124	59030	0.49, 9/1, *1.5*	1.09, 4/1, *160*	0.06, 10/3, *18*	1.25, 4/1, *12*	**0** *0.6*
pr136	136	96772	2.87, 0/0, *1.8*	2.58, 1/0, *250*	0.43, 9/0, *21*	0.95, 5/0, *15*	**0** *11*
pr144	144	58537	0.15, 10/0, *2.4*	0.16, 9/5, *340*	0.15, 9/6, *23*	2.75, 2/0, *18*	**0** *1.2*
pr152	152	73682	0.84, 6/0, *2.8*	0.77, 7/0, *380*	0.22, 10/1, *25*	1.82, 2/0, *22*	**0** *10*
pr226	226	80369	1.23, 0/0, *9.5*	—	0.37, 10/0, *47*	2.22, 6/0, *66*	**0** *20*
pr264	264	49135	4.89, 0/0, *16*	—	0.05, 10/8, *66*	1.92, 3/1, *125*	**0** *18*
pr299	299	48191	5.04, 0/0, *30*	—	0.15, 10/1, *90*	4.22, 0/0, *200*	**0** *320*
rat99	99	1211	4.48, 0/0, *0.6*	2.95, 1/0, *48*	**0** *13*	0.26, 10/2, *6.3*	**0** *0.8*
rat195	195	2323	7.47, 0/0, *5.0*	3.44, 0/0, *1700*	0.20, 10/0, *37*	0.30, 10/0, *40*	**0** *150*
rd100	100	7910	3.03, 0/0, *0.7*	3.26, 2/0, *49*	0.15, 10/7, *13.5*	1.75, 2/1, *6.6*	**0** *0.8*
si175	175	21407	0.45, 10/0, *3.7*	0.21, 10/0, *900*	0.04, 10/1, *31*	0.27, 10/0, *28*	**0** *11*
st70	70	675	0.76, 8/0, *0.2*	1.84, 2/0, *70*	0.02, 10/9, *9.0*	1.04, 5/1, *2.2*	**0** *0.3*
swiss42	42	1273	**0** *0.1*	1.33, 5/2, *0.5*	**0** *5.8*	1.07, 7/7, *0.6*	**0** *0.1*
ts225	225	126643	1.67, 2/0, *9.0*	—	0.02, 10/7, *50*	1.65 4/3, *62*	**0** *4.3*
tsp225	225	3916	5.17, 0/0, *9.3*	—	1.05, 2/0, *49*	2.00, 1/0, *67*	**0** *15*
u159	159	42080	3.14, 0/0, *3.0*	2.55, 1/0, *370*	0.68, 10/1, *26*	3.26, 2/0, *25*	**0** *1.4*
ulysses16	16	6859	**0** *0.0*	**0** *0.0*	**0** *3.4*	**0** *0.1*	**0** *0.0*
ulysses22	22	7013	**0** *0.1*	0.11, 9/9, *0.1*	**0** *5.4*	**0** *0.1*	**0** *0.0*

5.2. Experiments with the quadratic assignment problem

The quadratic assignment problem is formulated as follows. Given two matrices $\boldsymbol{C} = (c_{ij})_{n\times n}$ and $\boldsymbol{D} = (d_{kl})_{n\times n}$ and the set Π of permutations of the integers from 1 to n, find a permutation $p = (p(1), p(2), ..., p(n)) \in \Pi$ that minimizes

$$z(p) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} d_{p(i)p(j)} \ . \qquad (4)$$

The QAP is also NP-hard [29]. Problems of size $n > 36$, are not, to this date, practically solvable to optimality.

The QAP is a typical CO problem, where solutions are permutations, and the objective function is described according to the formula (4). As a neighbourhood structure, the 2-exchange neighbourhood function is widely used: $N_2(p) = \{ p' \mid p' \in \Pi, r(p, p') \leq 2 \}$, where $p \in \Pi$, and $r(p, p')$ is the distance between permutations p and p': $r(p,p') = \sum_{i=1}^{n} \mathrm{sgn} \mid p(i) - p'(i) \mid$ (see also

Section 1). Exactly, the neighbourhood N_2 was applied in our ITS algorithm for the QAP. A move from p to $p' \in N_2(p)$ can easily be defined by the operator f_{ij} ($i, j = 1, 2, ..., n$) which simply swaps the i-th and j-th elements in the given permutation, i.e. $p'(i) = p(j)$, $p'(j) = p(i)$, $1 \le i, j \le n \wedge j-i \ge 1$, where p is the current permutation, and p' – the neighbouring permutation.

We compared our ITS algorithm with other four efficient algorithms for the QAP: 1) simulated annealing (SA) algorithm [22]; 2) genetic algorithm (GA) [6]; 3) robust tabu search (RoTS) algorithm [32], and 4) reactive tabu search (ReTS) algorithm [2]. We tested the above algorithms on a set of instances taken from the QAP instances library QAPLIB [5]. The performance measures are similar to those for the TSP: a) the average deviation from the best known solution – $\bar{d}$ ($\bar{d} = 100(\bar{z} - \breve{z})/\breve{z}$ [%], where $\bar{z}$ is the average objective function value over 10 runs and $\breve{z}$ is the best known value (BKV) of the objective function; b) the number of solutions that are within 1% optimality (over 10 runs) – $C_{1\%}$; c) the number of the best known solutions found – C_{bks}. The results of the comparison are shown in Tables 2 and 3 (in Table 2, the results for the randomly generated instances are given, whereas in Table 3, we present the results for the real-life like instances – instances of this type are generated in such a way that the entries of the data matrices resemble a distribution from real world problems).

It may be viewed that the quality of results depends on the type of instances. For the randomly generated instances, the results are inferior to those for the real-life like instances. This is an indication that the random instances are much more difficult to solve and still remain a real challenge for the researchers. Regarding the real-life like instances, they are relatively easy for many heuristics, the ITS algorithm, too. For these instances, ITS produces very strong results. For example, the average CPU time needed to find the pseudo-optimal solution in every run out of 10 for the instance tai80b is equal to about 350 seconds on 900 MHz computer.

Table 2. Results of comparison of the algorithms for the random QAP instances.
The best results obtained are printed in bold face. CPU times per run are given in seconds.
(900 MHz PENTIUM computer was used in the experiments)

Instance	n	BKV	$\bar{d}$, $C_{1\%}/C_{\text{bks}}$										CPU time
			SA		GA		RoTS		ReTS		ITS		
tai20a	20	703482 [a]	0.90	6/0	0.26	10/3	0.29	10/1	0.15	10/6	**0.06**	10/**8**	*0.6*
tai25a	25	1167256 [a]	0.76	8/0	0.39	10/1	0.25	10/4	0.13	10/7	**0**		*2.4*
tai30a	30	1818146 [a]	0.74	8/0	0.24	10/4	0.16	10/6	0.20	10/4	**0**		*6.6*
tai35a	35	2422002 [a]	0.75	8/1	0.48	10/2	0.36	10/1	0.24	10/1	**0**		*17*
tai40a	40	3139370 [a]	0.78	8/0	0.78	9/0	0.52	10/0	0.32	10/0	**0.21**	10/**1**	*45*
tai50a	50	4941410 [a]	0.73	9/0	0.88	7/0	0.78	9/0	0.49	10/0	**0.32**	10/**2**	*180*
tai60a	60	7205962 [b]	0.85	8/0	0.86	7/0	0.84	7/0	0.52	10/0	**0.33**	10/**1**	*580*
tai80a	80	13546960 [b]	0.57	10/0	0.53	10/0	0.67	10/0	0.16	10/0	**0.14**	10/0	*1500*
tai100a	100	21087588 [c]	0.56	10/0	0.59	10/0	0.81	10/0	0.28	10/0	**0.26**	10/0	*3600*

[a] comes from [5]; [b] comes from [23]; [c] comes from [24].

Table 3. Results of comparison of the algorithms for the real-life like QAP instances.
The best results obtained are printed in bold face. CPU times per run are given in seconds.
(900 MHz PENTIUM computer was used in the experiments)

Instance	n	BKV	$\bar{d}$, $C_{1\%}/C_{\text{bks}}$										CPU time
			SA		GA		RoTS		ReTS		ITS		
tai20b	20	122455319 [a]	0.14	10/7	0.05	10/9	0.04	10/9	0.19	10/6	**0**		*0.3*
tai25b	25	344355646 [a]	0.58	9/6	0.01	10/9	0.00	10/9	0.48	8/1	**0**		*0.9*
tai30b	30	637117113 [a]	1.61	3/2	0.00	10/9	0.07	10/4	0.60	8/0	**0**		*2.8*
tai35b	35	283315445 [a]	0.63	8/6	0.02	10/9	0.03	10/8	0.25	10/2	**0**		*5.6*
tai40b	40	637250948 [a]	1.41	3/2	**0**		**0**		0.20	9/3	**0**		*14*
tai50b	50	458821517 [a]	0.33	10/0	0.10	10/8	0.06	10/2	0.19	10/0	**0**		*56*
tai60b	60	608215054 [a]	0.15	10/1	0.01	10/8	0.31	8/3	0.13	10/1	**0**		*130*
tai80b	80	818415043 [a]	0.57	7/0	0.21	10/5	0.49	10/0	0.15	10/0	**0**		*350*
tai100b	100	1185996137 [a]	0.17	10/0	0.09	10/5	0.03	10/0	0.08	10/0	**0.00**	10/9	*1400*
tai150b	150	498896643 [b]	0.20	10/0	0.42	10/0	0.26	10/0	0.36	10/0	**0.10**	10/**1**	*3600*

[a] comes from [5]; [b] comes from [33].

The results of ITS may be improved even more by accurate tuning of the control parameters. Of course, we can obtain higher quality solutions by increasing the total number of iterations. After the additional long-lasting experimentation, ITS was successful in discovering new record-breaking solutions for two largest available random instances from QAPLIB. It took approximately 10 and 20 hours on 900 MHz computer to find the new best known solutions for the instances tai80a and tai100a, respectively. The results are summarized in Table 4.

Table 4. New results for the large random QAP instances

Instance	# of runs	# of best known solutions found	New best known objective function values
tai80a	10	**1**	**13526696**
tai100a	10	**3**	**21075558, 21072418, 21071558**

6. Conclusions

One meta-heuristic that has been widely applied to many combinatorial optimization problems is tabu search. In this paper, an innovative enhancement of this method is proposed. We call this approach an iterated tabu search. The novelty of ITS is the incorporation of the special kind of solution reconstruction into the classical tabu search paradigm. ITS can be seen as a "reconstruct and improve" principle based optimization policy. It is distinguished, in principle, for two main components: diversification and intensification. During the first step, an existing solution is reconstructed in a proper way. In the second step, the local improvement based on the traditional tabu search procedure is applied to the solution just "ruined"; hopefully, the new improved solution is better than the solutions obtained in the previous iterations. By repeating these phases many times, one tries to seek for near-optimal solutions. That is the heart of ITS.

On a basis of the proposed framework, two variants of ITS were designed, which were applied to the NP-hard combinatorial optimization problems, the traveling salesman problem and the quadratic assignment problem. The results obtained from the experiments demonstrate high performance of the proposed strategy. ITS appears to be superior to pure TS algorithms, as well as other heuristic algorithms. The power of ITS is also corroborated by the fact that the new record-breaking (best know) solutions were found for very hard QAP instances – tai80a and tai100a. This indicates that ITS seems to be one of the extremely efficient heuristics for the random QAP instances.

As the results obtained for the TSP and QAP show very promising efficiency of ITS, it may be worthy applying the versions of the ITS method to other well-known combinatorial optimization problems.

Regarding further possible extensions of ITS, some directions may be proposed:

- maintaining "the history" of the locally optimal solutions (as the potential candidates for diversification (reconstruction));
- using the reactive tabu search (instead of the straightforward tabu search) as a possibly more efficient intensification algorithm;
- implementing other, more elaborated diversification (reconstruction) procedures;
- incorporating the ITS based procedure into other meta-heuristics, for example, genetic algorithms (as a robust local improvement heuristic).

Acknowledgements

The authors are grateful to Prof. E. Taillard for providing a code of the robust tabu search algorithm for the QAP. We also would like to thank Prof. C. Fleurent for the paper that helped coding the genetic algorithm for the QAP.

References

[1] **E.H.L. Aarts, J.K. Lenstra (eds.).** Local Search in Combinatorial Optimization. *Wiley, Chichester*, 1997.

[2] **R. Battiti, G. Tecchiolli.** The reactive tabu search. *ORSA Journal on Computing*, 1994, *Vol.*6, 126–140.

[3] **E.B. Baum.** Towards practical "neural" computation for combinatorial optimization problems. *In J.S. Denker (ed.) Neural networks for computing, American Institute of Physics, New York*, 1986, 53–58.

[4] **K. Boese.** Cost versus distance in the traveling salesman problem. *Tech. Report CSD*-950018, *UCLA CS Dept., USA*, 1995.

[5] **R.E. Burkard, S. Karisch, F. Rendl.** QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 1997, *Vol.*10, 391–403. [*See also* `http://www.seas.upenn.edu/qaplib/`.]

[6] **C. Fleurent, J.A. Ferland.** Genetic hybrids for the quadratic assignment problem. *In P.M. Pardalos, H. Wolkowicz (eds.), Quadratic Assignment and Related Problems*. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, *Vol.*16, *AMS*, *Providence*, 1994, 173–188.

[7] **M.R. Garey, D.S. Johnson.** Computers and Intractability: A Guide to the Theory of NP-Completeness. *Freeman, San Francisco*, 1979.

[8] **M. Gendreau.** An introduction to tabu search. In F.Glover, G.Kochenberger (eds.), *Handbook of Metaheuristics*, *Kluwer, Norwell*, 2002, 37–54.

[9] **D. Ghosh, G. Sierksma.** Complete local search with memory. *Journal of Heuristics*, 2002, *Vol.*8, 571–584.

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

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

[12] F. Glover, G. Kochenberger (eds.). *Handbook of Metaheuristics*, *Kluwer, Norwell*, 2002.

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

[14] P. Hansen, B. Jaumard. Algorithms for the maximum satisfiability problem. *RUTCOR Research Report* 43–87, *Rutgers University, USA*, 1987.

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

[16] D.S. Johnson. Local optimization and the traveling salesman problem. *In Proceedings of the* 17*th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science*, *Vol*.443, *Springer, Berlin*, 1990, 446–461.

[17] J. Knox. Tabu search performance on the symmetric traveling salesman problem. *Computers & Operations Research*, 1994, *Vol*.21, 867–876.

[18] H.R. Lourenco, O. Martin, T. Stützle. Iterated local search. *In F.Glover, G.Kochenberger (eds.), Handbook of Metaheuristics, Kluwer, Norwell*, 2002, 321–353.

[19] K. Mak, A. Morton. A modified Lin-Kernighan traveling salesman heuristic. *ORSA Journal on Computing*, 1992, *Vol*.13, 127–132.

[20] O. Martin, S.W. Otto. Combining simulated annealing with local search heuristics. *Annals of Operations Research*, 1996, *Vol*.63, 57–75.

[21] Z. Michalewicz, D.B. Fogel. How to Solve It: Modern Heuristics. *Springer, Berlin-Heidelberg*, 2000.

[22] A. Misevicius. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica*, 2003, *Vol*.14, 497–514.

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

[24] A. Misevicius, J. Blonskis. Experiments with tabu search for random quadratic assignment problems. *Information Technology and Control*, 2005, *Vol*.34, *No*.3, 237–244.

[25] C.H. Papadimitriou, K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. *Prentice-Hall, Englwood Cliffs*, 1982.

[26] V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, G.D. Smith (eds.). Modern Heuristic Search Methods. *Wiley, Chichester*, 1996.

[27] G. Reinelt. TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing*, 1991, Vol.3-4, 376–385. [*See also* `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.]

[28] E. Rolland, H. Pirkul, F. Glover. Tabu search for graph partitioning. *Annals of Operations Research*, 1996, *Vol*.63, 209–232.

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

[30] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 2000, *Vol*.159, 139–171.

[31] K. Smyth, H.H. Hoos, T. Stützle. Iterated robust tabu search for MAX-SAT. *In Y.Xiang, B.Chaib-draa (eds.), Advances in Artificial Intelligence: Proceedings of the 16th Conference of the Canadian Society for Computational Studies of Intelligence. Lecture Notes in Artificial Intelligence, Vol*.2671, *Springer, Berlin*, 2003, 129–144.

[32] E. Taillard. Robust taboo search for the QAP. *Parallel Computing*, 1991, *Vol*.17, 443–455.

[33] E. Taillard, L.M. Gambardella. Adaptive memories for the quadratic assignment problem. *Tech. Report IDSIA*-87-97, *Lugano, Switzerland*, 1997.

[34] P. Thomas, S. Salhi. A tabu search heuristic for the resource constrained project scheduling problem. *Journal of Heuristics*, 1998, *Vol*.4, 123–139.

[35] N.A. Wassan, I.H. Osman. Tabu search variants for the mix fleet vehicle routing problem. *Journal of the Operational Research Society*, 2002, *Vol*.53, 768–782.

Received June 2006.