ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2005, Vol. 34, No. 4

ITERATED TABU SEARCH FOR THE TRAVELING SALESMAN PROBLEM: NEW RESULTS

Alfonsas Misevičius, Jonas Smolinskas, Arūnas Tomkevičius

Kaunas University of Technology, Department of Multimedia Engineering
Studentų St. 50–416a, LT–51368 Kaunas, Lithuania

Abstract. In this paper, we present some new results obtained for the traveling salesman problem (TSP) by using the iterated tabu search (ITS) meta-heuristic. ITS is a promising extension to the ordinary tabu search scheme. It seeks near-optimal solutions by combining intensification (standard tabu search) and diversification (perturbation of solutions) in a proper way. For the TSP, the main effect is achieved due to decomposition of the solution neighbourhood structure and considerably speeding-up the tabu search process, which is used, namely, in the role of intensification. This fast-iterated tabu search (FITS) technique resulted in quite encouraging solutions for the TSP instances from the TSP instance library TSPLIB. FITS obviously outperformed the other heuristic algorithms used in the experimentation, especially, on the smaller TSP instances.

Keywords: combinatorial optimization, traveling salesman problem, heuristic algorithms, tabu search, iterated tabu search.

Indroduction

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

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

The interpretation of n, $\boldsymbol{D}$ and π is as follows: n is the number of cities; $\boldsymbol{D}$ is the matrix that contains distances between all the pairs of cities; $j = \pi(i)$ denotes city j to visit at step i. Usually, permutations are called tours, and the pairs $(\pi(1),\pi(2))$, ..., $(\pi(i),\pi(i+1))$, ..., $(\pi(n),\pi(1))$ are referred to as edges. Thus, solving the TSP means searching for the shortest closed tour in which every city is visited exactly once.

The TSP is a representative example of combinatorial optimization (CO) problems. In a general case, an instance of a CO problem is described as a pair (S, f), where S is the set of feasible solutions (solution space), and f: $S \to \Re$ is the objective (cost) function which assigns a real value to each solution. The goal is to find a solution $s_{\text{opt}} \in S$ such that

$$s_{\text{opt}} \in S_{\text{opt}} = \left\{ s^{\nabla} \mid s^{\nabla} = \arg\min_{s\in S} f(s) \right\} \qquad (2)$$

(we assume that f seeks a global minimum).

The solution s_{opt} is called a globally optimal solution (global optimum), and $S_{\text{opt}} \subseteq S$ denotes the set of optimal solutions. In addition, a neighbourhood function Θ: $S \to 2^S$ is defined: it attaches for every s in S a set $\Theta(s) \subseteq S$ – the set of neighbouring solutions of s. Each solution $s' \in \Theta(s)$ 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.

In the case of the TSP, Π plays the role of S, and z corresponds to f. An example of the neighbourhood function for the TSP is the 2-exchange neighbourhood Θ_2: $\Theta_2(\pi) = \{\pi' \mid \pi' \in \Pi,\ \rho(\pi,\pi') = 2\}$, where $\pi \in \Pi$ and $\rho(\pi,\pi')$ denotes the distance between permutations π and π'. As to the TSP, the distance between two permutations (tours) is usually described as the number of pairs of elements (edges) that are contained in the first permutation but not in the second one [3]. Formally, a transition from the permutation π to the neighbouring one $\pi' \in \Theta_2(\pi)$ may be defined by an operator $move(\pi,i,j)$: $\Pi \times \mathrm{N} \times \mathrm{N} \to \Pi$, which gives π' such that $\pi'(i)=\pi(i)$, $\pi'(i+1)=\pi(j)$, $\pi'(j)=\pi(i+1)$, $\pi'((j \bmod n)+1)=\pi((j \bmod n)+1)$, where $1 \le i, j \le n \wedge 1 < j-i < n-1$; in addition, if $j-i-2 \ge 1$, then $\pi'(i+k+1)=\pi(j-k)$ for every $k \in \{1, ..., j-i-2\}$ (that is, two edges at the positions i and j are removed and two different edges are added (see Figure 1)). The above operator is often referred to as a 2-opt(imal) move, because it enables to obtain an optimal solution with respect to the 2-exchange neighbourhood Θ_2. Similarly, higher order

moves may be defined: 3-opt, 4-opt, and so on. For a 2-opt move, we will also use a compact notation m_{ij}. The expression $\pi' = \pi \oplus m_{ij}$ would mean that π' is obtained from π by applying $move(\pi,i,j)$.

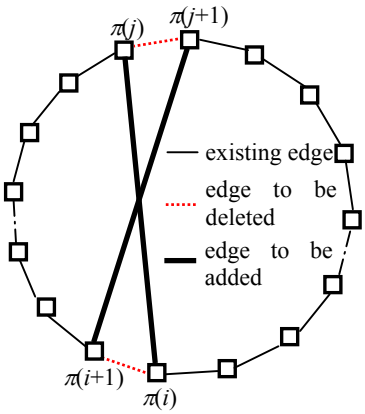


Figure 1. An example of 2-exchange move

For many years, the TSP serves as a "platform" for investigation of the artificial intelligence techniques. Since the TSP is NP-hard [8] and cannot be solved to optimality within polynomial computation time, heuristic algorithms have widely been used. Heuristics are not able to guarantee optimality, instead they often allow to find near-optimal solutions at a reasonable computational cost. Both specific, "tailored" procedures (for example, tour construction heuristics [2, 29], Lin-Kernighan algorithm [17], "elastic nets" [5]) and general purpose methods (like simulated annealing [26], tabu search [6, 15], genetic algorithms [7]) have been tried for solving the TSP. (More exhaustive overviews of the heuristic algorithms for the TSP can be found in [14, 16, 25, 28].)

In this paper, an improved tabu search (TS) algorithm version for the TSP is proposed. The paper is organized as follows. Firstly, we outline the iterated tabu search (ITS) framework, which is based on intensification and diversification (I&D) policy. In Section 2, an enhanced ITS procedure (called fast iterated tabu search (FITS)) for the traveling salesman problem is described. Computational results are presented in Section 3. Finally, Section 4 completes the paper with concluding remarks.

1. The iterated tabu search framework

Before describing the iterated tabu search framework, let us remind the basic features of the "canonical" tabu search. The central 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 moves resulting in going back to the previous solutions are to be prohibited in order to avoid cycling. Such moves become "tabu" – hence the name of the method. The TS algorithm starts from an initial solution $s°$ in S. At each step of the procedure, a subset $\Theta^*(s) \subseteq \Theta(s)$ of the neighbouring solutions of the current solution s is considered, and the move to the solution $s' \in \Theta^*(s)$ that improves most the objective function value f is chosen. s' must not necessary be better than s: if there are no improving moves, the algorithm chooses the one that least degrades the objective function. 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, which keeps information on the last h moves that have been done during the search process (h is called a tabu list size (or tenure)). If a move is actually contained in this list, it is considered as tabu. This way of proceeding hinders the algorithm from going back to a solution reached within the last h iterations. The straightforward prohibition, however, may lessen the efficiency of the search; so, an aspiration criterion is introduced to permit the tabu status to be ignored under certain circumstances. Typically, a move to s' is permitted if $f(s') < f(s^*)$, where s^* is the best solution found so far. The process is stopped as soon as a termination criterion is satisfied, for example, an a priori number of iterations have been performed. For more details on the principles of TS, the reader is addressed to [9, 11, 12].

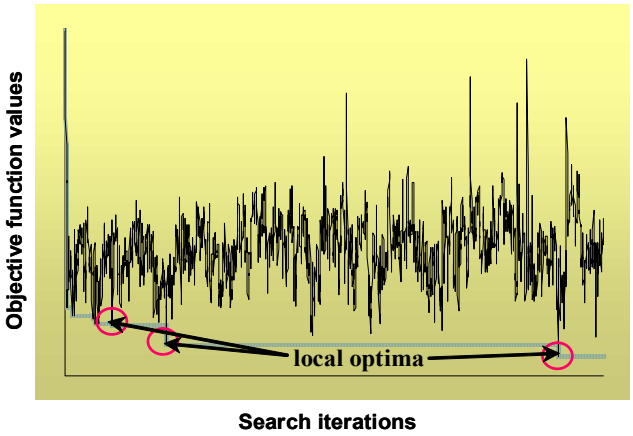


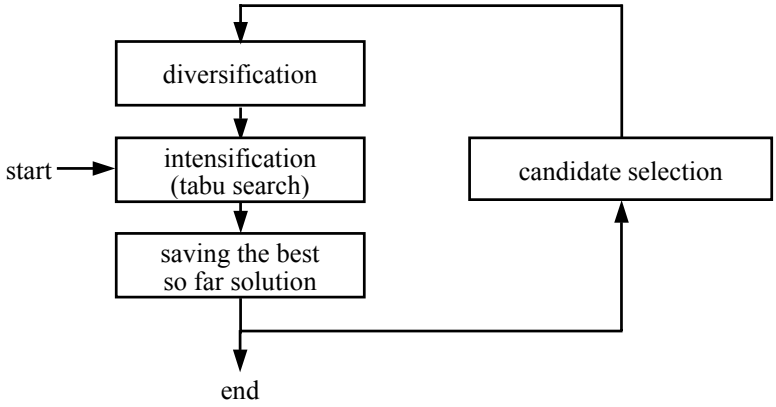
Figure 2. Illustration of the extremely rugged "landscape"

The ordinary tabu search implementations usually face some difficulties, first of all, a huge number of locally optimal solutions over the solution set. Isolated optima and highly rugged landscapes of the objective function make things even more complicated (see Figure 2). This is why the standard tabu search algorithms suffer from the stagnation phenomenon despite the fact that these algorithms are capable of escaping separate local optima. In addition, the repeating sequences of the search configurations, i.e. cycles may occur frequently, especially in the cases when the tabu list size is too small. There is a third situation where getting stuck in local optima and the cycles are absent but the search trajectories are still confined in limited

portions of the search space. This situation is known as "chaotic attractors" (or "deterministic chaos").

In order to try to overcome these barriers, an extension of the straightforward TS – the iterated tabu search – can be proposed. Basically, ITS relies on the intensification and diversification (I&D) policy. This concept is more than fifteen years old and goes back to Baum [1]. Later, various enhancements of the basic idea have been proposed, among them, iterated Lin-Kernighan [13], large step Markov chains [19], variable neighbourhood search (VNS) [24], "ruin and recreate" (R&R) principle [30], iterated local search (ILS) [18]. Note that most of these methods have been applied with success to the traveling salesman problem.

The heart of I&D is to obtain better optimization results by a perturbation (reconstruction) of an existing solution and a following improvement procedure. The rationale is that continuing the search from the reconstructed solution may allow to find new solutions that are better than those obtained in the previous iterations. By applying this type of process in an iterative way, one seeks for high quality solutions. The I&D framework is distinguishing for three main components (factors): intensification, diversification, and candidate selection (see Figure 3).



I&D is initiated by the local improvement of an initial solution. As a result, the first optimized solution, say $s^{\bullet}$, is achieved. Further, a given solution undergoes a "destruction", and a new solution, say $s^{\sim}$, is obtained. The reconstructed solution $s^{\sim}$ serves as an input for the subsequent intensification procedure, which starts immediately after the reconstruction is finished. This procedure returns a new optimized solution $s^{\bullet}$, which (or some other local optimum), in turn, is reconstructed, and so on. The new better solution (s^{*}) found at the current iteration is saved in a memory. The process continues until a stopping condition is met.

Figure 3. Basic flowchart of I&D

The goal of intensification is to concentrate the search in a localized region, i.e. the neighbourhood of the current solution. Mathematically, intensification can be viewed as a mapping ψ: $S \to S$ such that $f(s) \le f(\psi(s))$, where $s \in S$ is the current solution. If this intensification is performed by means of the standard tabu search, one just gets the ITS method. (TS seems to be an ideal local optimizer, because the power of TS has been demonstrated for plenty of optimization problems.) 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.

Diversification is responsible for escaping from the current local optimum and moving towards new regions in the solution space. It may be viewed as a certain kind of perturbation of solutions. (Instead of saying "perturbation", other terms may be utilized: "mutation", "reconstruction" [20], "ruin" [30], or "kick move" [19].) Perturbation can formally be defined as an operator ζ: $S \to S$ such that $\zeta(s) \in S$ and $\zeta(s) \ne s$, where $s \in S$ is the solution that undergoes the perturbation. It is important that a proper strength of perturbations is kept up: if the perturbations are too strong, the resulting algorithm might be similar to a random multistart, whereas if they are too weak, the process would cyclically return to the previous solutions. Perturbations are applied to locally optimal solutions only, as a result, ITS is the "walking" over an optimized search space, which consists solely of local optima. Such a process appears to be much more effective than when searching in a pure random solution space.

Regarding the candidate selection, two main alternatives exist: a) an exploitation and b) an exploration. The exploitation is achieved by choosing only the currently best local optimum – the best so far (BSF) solution – as a candidate for the reconstruction. In the case of exploration, a variety of policies exist. In fact, each locally optimized solution (not necessary the best local optimum) can be considered as a potential perturbation candidate. An example of exploration strategies is a "where you are" (WYA) approach: in this case, every new local optimum (no matter its quality) is accepted for perturbation. However, more sophisticated strategies are available, for example, selection from a memory of locally optimal solutions, like in the population-based algorithms. Even generation of a new solution from scratch (i.e. random restarts of the search) could be considered as a very special case of the candidate selection. The selection may be formalized

by introducing an operator ξ: $2^S \to S$. If exploitation is used, then the following equation holds: $\xi(\cdot) = s^\bullet$, where $s^\bullet$ ($s^\bullet \in S$) is the BSF solution.

ITS is a high level meta-heuristic – not a pure heuristic. The structure of this meta-heuristic is surprisingly simple. However, such a generalized method, like ITS, succeeds in search if only it involves the specific problem knowledge. The algorithm designer must be very careful by implementing both the tabu search and diversification procedures. These procedures should be as much problem-oriented as possible. The examples of such "tailored" algorithms are discussed in the next section.

2. An improved iterated tabu search algorithm for the TSP

2.1. Initial solutions for the iterated tabu search

During the preliminary experimentation, we tried the well-known insertion heuristic [29] for the construction of the initial solutions. However, we didn't observed any improvement neither in the quality of the final results of ITS, nor in the total computation time when comparing with a pure random generation of the initial solutions. In particular, the last variant was used in the main experiments.

2.2. Intensification (local improvement)

By designing the intensification algorithm, our attention was turned to the reducing the computational cost. This improvement does not directly refer to the ITS paradigm but to the tabu search itself (in particular, the way the tabu search explores the neighbourhood). As a rule, the tabu search explores the whole neighbourhood of the current solution and selects the best available solution that is not forbidden. This can be quite time consuming, especially if we are dealing with large problems. We present the fast tabu search (FTS), which considerably speeds up the neighbourhood search process.

FTS is founded on the fast local search (FLS) [31] and "don't look bit" [4] methodologies. The key idea is that the current neighbourhood is broken down into a number of small sub-neighbourhoods (SNs)[1]. The type of the neighbourhood and the number of sub-neighbourhoods are arbitrarily chosen by the user (we use the 2-exchange neighbourhood Θ_2, which is divided into $\lfloor 0.1n \rfloor$ sub-neighbourhoods). The information needed by FTS is saved in a sub-neighbourhood memory (SNM), which can be thought of as an analogue of frequency-based (long-term) memory and candidate lists [10]. The structure of SNM is very simple. It is organized as three one-dimensional arrays. The entries of these arrays contain the following data: starting location (position) of the current sub-neighbourhood within the whole neighbourhood (a pair of indices (i, j) is enough to unambiguously define this location), the size of the sub-neighbourhood (it is equivalent to the number of steps to be performed by exploring the current SN), and the tabu tenure for the current sub-neighbourhood.

Our FTS method can be seen as a two-level process. The higher level process maintains the set of the sub-neighbourhoods in the given order and guides the lower level tabu search procedure, which, in turn, is responsible for the exploration of the current sub-neighbourhood.

The following are the main features of the high level process. At the beginning, all the tabu tenures are set to zero, i.e. all sub-neighbourhoods are active. The idea is to scan continuously the sub-neighbourhoods, searching only those that are active (not tabu), i.e. the actual tabu tenure is less or equal to the current iteration number. The sub-neighbourhoods that do not meet this criterion are not being searched. If a sub-neighbourhood is examined – this is done by means of the lower level tabu search – and does not contain any improving moves, then it becomes tabu (the non-improving move, however, is still performed). Otherwise, it remains active and the improving move is performed. In both cases, the tabu search process does not restart but it continues with the next sub-neighbourhood. As the search progresses, more and more SNs become inactive (if the tenure is sufficiently large) until all the sub-neighbourhoods are tabu. If no active sub-neighbourhood exists, then a perturbation based on a double-bridge move (see Section 2.3) takes place and all the tabu tenure are again set to zero. The overall process is continued until a termination criterion is satisfied, for example, a pre-defined number of iterations have been executed. The solution obtained to this point serves as the current locally optimal solution, which is to be reconstructed if WYA strategy is applied.

It should be noted that our approach is quite different from that proposed by Fiechter [6], in which tours are divided in a number of separate open sub-tours (slices) and the tabu search is done completely independently on these sub-tours.

The low level tabu search procedure deals with rather short-term (recency-based) memory. It is organized as an $n \times n$ integer matrix STM. Initially, all the entries of STM are set to zero (the initialization is performed once before starting the high level process). As the search progresses, the entries $STM[i][j]$ store the current number of the iteration plus the tabu tenure h_{STM}. Thus, a move m_{ij} (this is equivalent to adding the edges $(\pi(i), \pi(j))$, $(\pi(i+1), \pi(j+1))$ and deleting the edges $(\pi(i), \pi(i+1))$, $(\pi(j), \pi(j+1))$) is tabu if the value of $STM[i][j]$ is equal or greater than the current iteration number. By using this type of the tabu list, testing whether a move is tabu or not requires only $O(1)$ time – this fact is very important for the fast tabu search.

[1] Remind that the technique of using small portions of the neighbourhood was also tried in [21], but without maintaining a sub-neighbourhood memory.

```
function FTS(π, n, τ);  // fast tabu search procedure for the TSP //
  // input: π – the current permutation (tour); n – the problem size; τ – the number of iterations; //
  //        h_STM, h_SNM – the tabu tenures for STM and SNM; ω – the alternative intensification factor //
  //        γ – the expected mild stagnation period (emsp) factor //
  // output: π• – the best solution found //
  SNM_Size := ⌊0.1n⌋;  // SNM_Size denotes the size of the sub-neighbourhood memory //
  initialize the sub-neighbourhood memory (SNM);
  SNM_Pointer := 0; // set the SNM pointer to an initial value //
  STM := 0; A := 0; c := 1; c' := 1; c'' := 1;
  α := 0.05, r := ωh_STM; emsp := ⌊γ τ⌋; improved := FALSE;
  π• := π;
  while (c ≤ τ) or improved = TRUE then begin // main cycle //
    repeat
      SNM_Pointer := (SNM_Pointer mod SNM_Size) + 1;
      if SNM.Tenure[SNM_Pointer] ≤ c then break
    until all the SNM entries are scanned;
    if (all SNM tenures are greater than c) or (c – c' ≥ emsp) then begin
       apply double-bridge mutation to π;
       re-initialize the sub-neighbourhood memory;
       SNM_Pointer := 1;
       c' := c
    end;
    // the search will be performed in the current active sub-neighbourhood defined by the SNM pointer //
    i := SNM.I[SNM_Pointer];  // indices i, j define the sub-neighbourhood to be explored //
    j := SNM.J[SNM_Pointer];
    n' := IF(i = 1, n – 1, n) ;
    θ := SNM.Iter_N[SNM_Pointer];  // θ is the size of the current sub-neighbourhood //
    Δz_min := ∞;
    for w := 1 to θ do begin // find the best non-tabu move in the current sub-neighbourhood //
       i := IF( j < n', i, IF(i < n – 2, i + 1, 1)) ; n' := IF(i = 1, n – 1, n) ; j := IF( j < n', j + 1, i + 2) ;
       Δz := z( π ⊕ m_ij ) – z(π);
       tabu := IF(STM[i][j] ≥ c and RANDOM() ≥ α, TRUE, FALSE);
       aspired := IF((z(π) + Δz < z(π•)) or (z(π) + Δz < A[i][j]), TRUE, FALSE);
       if ((Δz < Δz_min) and NOT(tabu)) or aspired then begin Δz_min := Δz; u := i; v := j
                                                            end
    end;  // for //
    improved := IF(Δz_min < 0, TRUE, FALSE);
    if improved
        then SNM.Tenure[SNM_Pointer] := 0            // the sub-neighbourhood stays active //
        else SNM.Tenure[SNM_Pointer] := c + h_SNM;  // the sub-neighbourhood becomes inactive //
    if Δz_min < ∞ then begin
       π := π ⊕ m_uv ;            // replace the current permutation (tour) by the new one //
       STM[u][v] := c + h_STM;  // make the move m_ij tabu //
       A[u][v] := z(π)            // update the corresponding position in the aspiration list //
    end;  // if //
    if (improved = TRUE) and (c – c'' ≥ r) then begin
       apply the alternative intensification (two-opt) procedure to π;
       c'' := c
    end;
    if z(π) < z(π•) then begin π• := π; c' := c end;  // save the best so far solution //
    c := c + 1
  end;  // while //
  return π•
end.
```

Figure 4. Template of the fast tabu search algorithm for the TSP

The basic steps of the low level tabu search procedure are as follows:

- find a neighbour p' of the current solution p in such a way that $p' = \arg\min_{p'' \in N_2(p)} z(p'')$ and the corresponding move from p to p' is not tabu or the aspiration criterion holds;
- update the short-term memory (matrix *STM*) by including the move m_{uv}, where m_{uv} is the move from the solution p to the solution p';
- replace the current permutation p by the neighbour p', and use as a starting solution for the future steps;
- save the current solution if it appears better than the best solution found so far.

Our aspiration criterion is as follows: a move m_{ij} is allowed (even if it is tabu) if the best so far solution has been found, or m_{ij} results in the objective function value that is less than the appropriate value in the aspiration list, i.e. $z(p \oplus m_{ij}) < A[i][j]$. The aspiration value $A[i][j]$ is updated each time the move m_{ij} is performed, that is, it is equivalent to the objective function value at the moment of performing m_{ij}.

We are also using a trick proposed in [23]: the tabu status is ignored (rejected) with a small probability even if the aspiration criterion does not hold. In our experimentation, the value of this rejection probability, a, was set to 0.05.

Another enhancement of the low level TS is the embedding of an alternative intensification mechanism [22]. 2-opt based steepest descent (SD) procedure is tried in this role. The rationale of applying 2-opt algorithm is to prevent an accidental miss of local optima and to intensify the search even more at the moments of decreasing of the values of the objective function. The alternative intensification procedure is omitted if it already took place within the last r iterations ($r = wh_{\mathrm{STM}}$, where w is the alternative intensification frequency factor).

The detailed template of the FTS algorithm is given in Figure 4.

```
function ITS(p, n, Q, t, h_SNM, h_STM, w, h, g, x1, x2);  // iterated tabu search procedure for the TSP //
  // input: p – the current (initial) permutation (tour); n – the problem size; //
  //        Q – the total number of iterations; t – the number of iterations for the TS procedure; //
  //        h_STM, h_SNM – the tabu tenures for STM and SNM; w – the alternative intensification factor //
  //        h – the expected deep stagnation period (edsp) factor, g – the mild stagnation period factor //
  //        x1, x2 – the perturbation strength factors //
  // output: p* – the best permutation found //
  p• := FTS(p, n, t);  // improve the initial solution //
  p := p•; p* := p•;
  q' := 1;  m_a := MAX(4, ⌊x1·n⌋); m_b := MAX(4, ⌊x2·n⌋); m := m_a – 1;  edsp := ⌊h·n⌋;
  for q := 1 to Q do begin  // main cycle //
    p := IF(z(p•) < z(p), p•, p);  // choose the candidate for the perturbation //
    m := IF(m < m_b, m + 1, m_a);  // update the perturbation strength //
    if (q – q' ≥ edsp) then begin  // deep stagnation condition is met //
        apply NN-reconnect procedure to p with the strength 1.5m,
        get resulting solution p~;
        apply random 5-opt moves to p~;
        q' := q
      end
      else apply NN-reconnect procedure to p with the strength m,
          get resulting solution p~;
    p• := FTS(p~, n, t);  // try to improve the reconstructed solution by fast tabu search //
    if z(p•) < z(p*) then begin
      p* := p•;  // save the best so far permutation //
      m := m_a – 1;  // reset the perturbation strength //
      q' := q      // q' denotes the iteration at which the new better solution is found //
    end // if //
  end; // for //
  return p*
end.
```

Figure 5. Template of the fast iterated tabu search algorithm for the TSP

2.3. Diversification (solution perturbation)

For the diversification, we use three different sort perturbations of solutions: a nearest-neighbour-reconnect (NN-reconnect) procedure, double-bridge (4-opt) moves, and random 5-opt moves.

In FITS, the NN-reconnect algorithm serves as a main diversification instrument. This algorithm (in [21], it is entitled as a randomized greedy reconnect procedure) consists of three main steps (see also Figure 6): 1) disintegration; 2) nearest neighbour (NN) procedure; 3) recreation. In more details, the current solution is ruined (destroyed) in a stochastic way – so that two sub-tours (segments) are obtained. Further, the cities of one segment are reconnected by using the nearest neighbour heuristic [29]; consequently, a new sub-tour is created. Finally, this locally optimized segment is copied back to its original position, i.e. the tour is recreated. The only control parameter for the NN-reconnect procedure is the size of the tour segment to be reconnected. We let this size vary in some interval $[m_a,m_b] \subseteq [4,n]$; here, the values of m_a, m_b are related to the problem size n, i.e. $m_a = \max(4,\lfloor x_1 n \rfloor)$ and $m_b = \max(4,\lfloor x_2 n \rfloor)$, where x_1, x_2 ($0<x_1 \le x_2 \le 1$) are user-defined coefficients. The detailed template of the NN-reconnect procedure can be found in [21].

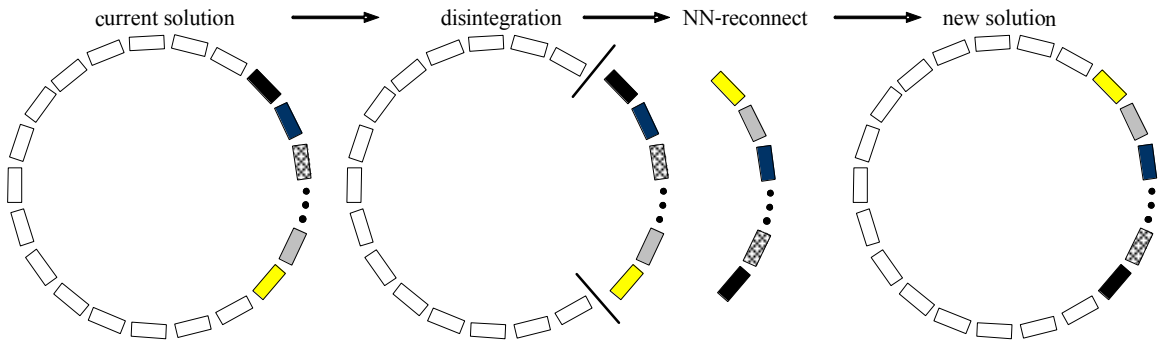


Figure 6. The graphical interpretation of the NN-reconnect perturbation

The NN-reconnect perturbation cannot be undone easily by 2-opt based local (tabu) search. Furthermore, this perturbation does not increase the tour length significantly, since it incorporates the intelligent NN heuristic instead of a blind random mutation. These two properties – fundamentally different change nature and small strength (low degree of disruptiveness) – makes the NN-reconnect procedure a robust diversification operator.

In addition to the NN-reconnect procedure described, the alternative perturbations (the so-called double-bridge (DB) and random 5-opt moves) are applied under certain conditions; in particular, we employ these perturbations when the stagnation of the search is observed. The stagnation is said to take place if the best so far solution remains unchanged for a relatively long time. Depending on this time, we distinguish between a mild and deep stagnation.

The double-bridge perturbation cuts four edges (thus, it is a particular case of a 4-opt move) and introduces four new edges as shown in Figure 7a. Most of the I&D-based algorithms for the TSP have incorporated this kind of perturbation. Surprisingly, it has been found almost equally effective for different instance sizes. We apply the DB perturbation in the case of the mild stagnation being detected during the execution of the low level TS procedure. In addition, the DB move is utilized if all the sub-neighbourhoods become inactive while in the higher level TS process (see Section 2.2).

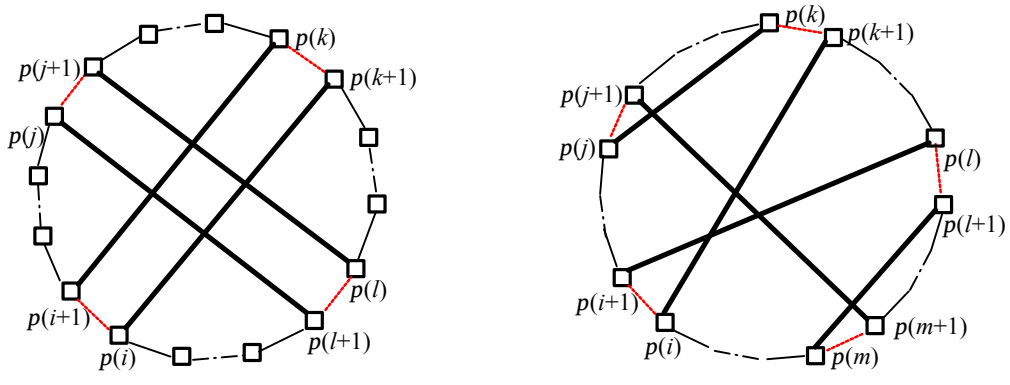


Figure 7. Examples of double-bridge (a) and 5-opt (b) moves

Finally, we also make use of random 5-opt moves (coupled with the NN-reconnect perturbation) in the cases where the situation of the deep stagnation is identified. 5-opt moves are in similar spirit with DB moves, except that one deals with five edges chosen in a pure random way (see Figure 7b).

2.4. Candidate selection

As to the candidate selection, we have chosen the exploitation strategy. That is, only the currently best solution (BSF-solution) has a chance to be accepted for the subsequent perturbation. We found that, at least for the smaller TSP instances, the exploitation strategy yields better results than the alternative exploration (WYA) strategy. However, the other intelligent candidate selection schemes (for example, combination of exploitation and exploration) are worth examining, especially at the longer ITS runs.

The template of the resulting fast iterated tabu search algorithm is presented in Figure 5. The default values of the control parameters for this algorithm are as follows: $Q = 100$; $t = 0.5n$; $h_{SNM} = 0.05n$; $h_{STM} = 0.1n$; $w = 0.8$; $h = 0.08$; $g = 0.2$; $x_1 = 0.2$; $x_2 = 0.25$.

3. Computational experiments

To examine the effectiveness of the fast iterated tabu search algorithm, a number of experiments have been carried out. We used the TSP instances taken from the publicly available library of the TSP instances, TSPLIB [27]. In the experiments conducted, the following five heuristic algorithms were tried:

- the random multi-start (RMS) algorithm based on 2-opt moves;
- the simulated annealing (SA) algorithm;
- the straightforward tabu search (STS) algorithm (without diversification mechanism);
- the iterated tabu search (ITS) algorithm (with NN-reconnect based diversification procedure);
- the fast iterated tabu search (FITS) algorithm.

All the algorithms are coded by A. Misevicius in the programming language Free Pascal.

Our standard performance measures of the algorithms are used: a) the average deviation of solutions from a provably optimal solution – $\bar{d}$ ($\bar{d} = 100(\bar{z} - z_{opt})/z_{opt}$ [%], where $\bar{z}$ is the average objective function value (i.e. the tour length) over 10 independent runs of the corresponding algorithm, and z_{opt} is the optimal objective function value (values z_{opt} are from [27])); b) the number of solutions that are within 1% optimality (over 10 runs) – $C_{1\%}$; c) the number of the optimal solutions – C_{opt}.

In the experiments, the similar conditions are guaranteed for all the algorithms tested: they use the identical initial solutions, runs on the same computer, and require approximately equivalent execution (CPU) time.

The results of the comparison are presented in Table 1. They confirm the relatively high efficiency of the new proposed algorithm with respect to the performance measures used (especially, the average deviation). It can be seen that FITS obviously outperforms RMS, STS, and SA, in particular, for the smaller instances. It also appears to be superior to the earlier iterated tabu search algorithm version presented in [21]. To achieve more fairness, the experiments on the larger instances, as well as comparison with the other powerful algorithms would be useful.

Although the results of FITS for the TSP seem to be quite encouraging, there is still a room for the further improvements.

Firstly, the quality of the results could be increased by a more careful tuning of the control parameters to the particular problems. This is especially true for the tabu tenure, the perturbation strength, and the number of sub-neighbourhoods. We may also expect to improve the results by increasing the total number of tabu search iterations. However, there is always a danger of encountering deep stagnations if the run time is augmented. During our experiments, we observed that even in the cases of the moderate run times, stagnation still reveals itself, especially, for the larger problems ($n > 200$) (see Figure 8).

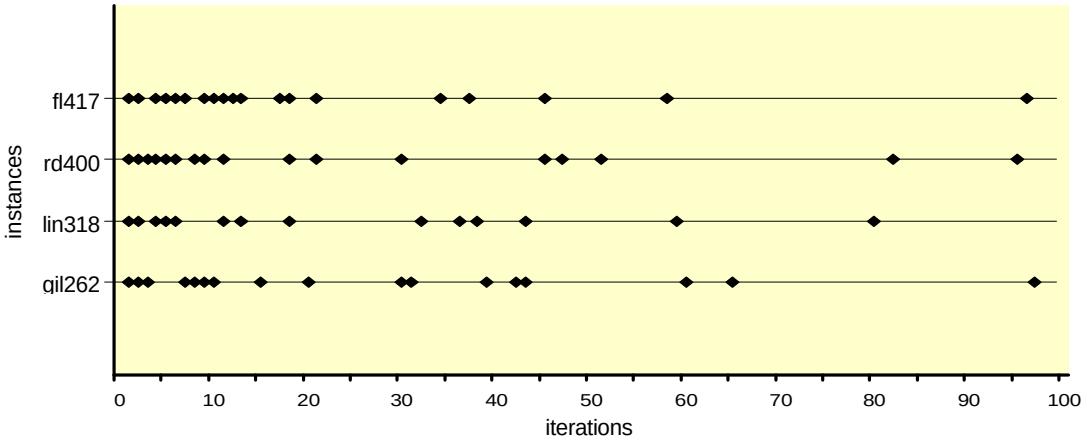


Figure 8. Illustration of the stagnation situation for different TSP instances.
Note. Tick marks correspond to the moments (iterations) at which a new (better) objective function value is found

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

Instance	n	z_{opt}	$\bar{d}$, $C_{1\%}/C_{opt}$										CPU time
			RMS		SA		STS		ITS		FITS		
a280	280	2579	7.984,	0/0	0.314,	9/2	3.715,	0/0	0.305,	9/1	**0.136**,	10/7	*7.8*
att48	48	10628	0.810,	6/0	0.436,	9/3	0.472,	9/0	**0**		**0**		*0.06*
bayg29	29	1610	0.335,	10/4	0.031,	10/9	0.120,	10/6	0.031,	10/9	**0**		*0.02*
bays29	29	2020	0.069,	10/8	0.059,	10/8	**0**		**0**		**0**		*0.02*
berlin52	52	7542	0.959,	7/5	**0**		1.003,	7/6	**0**		**0**		*0.08*
bier127	127	118282	2.390,	0/0	1.277,	2/0	1.644,	3/0	0.064,	7/2	**0.030**,	10/9	*0.6*
brazil58	58	25395	0.120,	10/4	**0**		**0**		**0**		**0**		*0.09*
brg180	180	1950	10.205,	0/0	13.077,	0/0	**0**		**0**		**0**		*2.3*
burma14	14	3323	**0**		**0**		**0**		**0**		**0**		*0.003*
ch130	130	6110	3.231,	0/0	0.453,	8/1	2.132,	6/0	0.136,	10/3	**0.049**,	10/9	*1.0*
ch150	150	6528	4.884,	0/0	0.695,	9/0	2.032,	3/0	0.129,	10/4	**0.050**,	10/8	*1.5*
d198	198	15780	2.404,	0/0	0.181,	10/1	0.770,	9/0	0.091,	10/3	**0.069**,	10/8	*3.8*
d493	493	35002	5.575,	0/0	0.737,	9/0	3.176,	7/6	0.656,	3/0	**0.597**,	7/1	*58*
dantzig42	42	699	0.129,	9/9	0.072,	10/9	**0**		**0**		**0**		*0.04*
eil51	51	426	2.653,	0/0	0.093,	10/7	0.657,	7/0	**0**		**0**		*0.06*
eil76	76	538	3.941,	0/0	0.376,	9/3	1.281,	2/0	**0**		**0**		*0.2*
eil101	101	629	4.897,	0/0	0.493,	8/2	2.005,	0/0	0.107,	9/2	**0.017**,	10/9	*0.4*
fl417	417	11861	2.300,	0/0	1.139,	5/0	1.117,	4/0	**0.129**,	10/2	0.138,	10/1	*31*
fri26	26	937	**0**		**0**		**0**		**0**		**0**		*0.012*
gil262	262	2378	5.938,	0/0	0.538,	9/0	3.990,	0/0	**0.344**,	5/1	0.236,	9/1	*6.8*
gr17	17	2085	**0**		**0**		**0**		**0**		**0**		*0.004*
gr21	21	2707	**0**		**0**		**0**		**0**		**0**		*0.007*
gr24	24	1272	0.003,	10/9	**0**		**0**		**0**		**0**		*0.01*
gr48	48	5046	0.404,	10/0	0.220,	9/7	0.421,	10/3	**0**		**0**		*0.07*
gr96	96	55209	2.068,	0/0	0.594,	9/2	2.172,	2/0	0.013,	10/9	**0**		*0.4*
gr120	120	6942	3.755,	1/0	0.964,	3/0	1.229,	3/0	**0.056**,	7/6	0.100,	10/9	*0.7*
gr137	137	69853	3.175,	0/0	0.879,	6/0	1.422,	2/0	0.008,	10/9	**0**		*1.2*
gr202	202	40160	4.421,	0/0	0.513,	10/0	2.790,	0/0	0.099,	10/5	**0.098**,	9/7	*3.2*
gr229	229	134602	4.439,	0/0	0.780,	8/0	3.200,	0/0	0.345,	9/0	**0.198**,	8/4	*4.9*
gr431	431	171414	5.724,	0/0	1.300,	2/0	4.524,	0/0	0.820,	3/0	**0.546**,	5/0	*42*
hk48	48	11461	1.587,	1/0	0.031,	10/6	0.218,	10/3	**0**		**0**		*0.07*
kroa100	100	21282	1.257,	4/0	0.429,	10/2	0.808,	7/1	**0**		**0**		*0.5*
kroa150	150	26524	3.921,	0/0	0.658,	8/0	2.038,	2/0	0.013,	10/9	**0**		*1.5*
kroa200	200	29368	4.679,	0/0	0.725,	7/1	2.378,	1/0	0.022,	10/6	**0.002**,	10/9	*3.5*
krob100	100	22141	2.331,	1/0	0.364,	6/0	1.366,	5/0	0.036,	10/5	**0**		*0.5*
krob150	150	26130	3.182,	0/0	0.840,	3/1	1.081,	2/0	0.024,	10/6	**0.021**,	10/9	*1.5*
krob200	200	29437	4.974,	0/0	0.916,	3/0	3.890,	0/0	0.509,	7/2	**0.143**,	10/5	*3.4*
kroc100	100	20749	2.204,	1/0	0.697,	10/1	0.478,	10/1	**0**		**0**		*1.5*
krod100	100	21294	2.421,	1/0	0.547,	7/2	1.285,	5/1	0.019,	10/8	**0**		*1.5*
kroe100	100	22068	2.390,	1/0	0.601,	6/1	1.110,	5/0	0.001,	10/9	**0**		*1.5*
lin105	105	14379	1.376,	3/0	0.189,	10/4	0.974,	5/1	**0**		**0**		*1.55*
lin318	318	42029	5.075,	0/0	1.288,	0/0	3.812,	0/0	0.769,	3/0	**0.433**,	7/1	*13.6*
pcb442	442	50778	7.848,	0/0	1.029,	7/6	2.392,	0/0	0.699,	6/0	**0.487**,	9/0	*40*
pr76	76	108159	1.203,	6/0	0.187,	10/6	0.379,	10/4	**0**		**0**		*0.2*
pr107	107	44303	1.236,	3/0	0.103,	10/8	0.442,	9/3	**0**		**0**		*0.6*
pr124	124	59030	0.733,	8/1	0.225,	6/1	0.718,	8/2	**0**		**0**		*1.0*
pr136	136	96772	3.819,	0/0	0.752,	4/1	1.046,	2/0	0.005,	10/9	**0**		*1.1*
pr144	144	58537	0.241,	10/0	0.404,	10/2	0.182,	10/3	0.000,	10/9	**0**		*1.4*
pr152	152	73682	0.981,	5/0	0.297,	9/3	1.093,	4/0	0.029,	10/8	**0**		*1.6*
pr226	226	80369	1.365,	2/0	0.947,	3/0	1.437,	1/0	0.032,	10/7	**0.005**,	10/9	*5.3*
pr264	264	49135	5.463,	0/0	0.099,	8/3	1.002,	6/0	0.009,	10/9	**0**		*8.0*
pr299	299	48191	5.684,	0/0	0.612,	7/2	3.184,	1/0	0.066,	10/3	**0.043**,	8/8	*10.4*
pr439	439	107217	5.978,	0/0	2.179,	0/0	2.792,	0/0	0.572,	7/0	**0.335**,	10/1	*42*
rat99	99	1211	4.550,	0/0	0.429,	7/4	0.533,	9/1	**0**		**0**		*0.4*
rat195	195	2323	7.645,	0/0	0.908,	5/1	2.361,	0/0	0.075,	10/1	**0.006**,	10/7	*3.0*
rd100	100	7910	3.195,	0/0	0.939,	3/0	2.100,	3/0	0.001,	10/9	**0**		*0.4*
rd400	400	15281	6.746,	0/0	0.777,	6/0	3.759,	0/0	0.714,	7/0	**0.512**,	8/1	*41*
si175	175	21407	0.515,	10/0	0.044,	10/8	0.112,	10/2	0.004,	10/4	**0**		*2.2*
st70	70	675	0.889,	6/0	0.415,	9/2	0.596,	7/1	**0**		**0**		*0.18*
swiss42	42	1273	0.110,	9/8	**0**		**0**		**0**		**0**		*0.05*
ts225	225	126643	2.139,	1/0	1.360,	1/0	1.207,	6/0	**0**		**0**		*4.8*
tsp225	225	3916	5.789,	0/0	1.147,	1/0	3.244,	1/0	0.380,	7/2	**0.230**,	10/5	*4.6*
u159	159	42080	3.506,	0/0	0.689,	8/0	1.651,	4/1	0.003,	10/9	**0**		*1.8*
ulysses16	16	6859	**0**		**0**		**0**		**0**		**0**		*0.003*
ulysses22	22	7013	0.002,	10/9	**0**		**0**		**0**		**0**		*0.007*

Secondly, the new conceptual enhancements of the basic ITS idea may be investigated. We can optimize separately each of the three components of the intensification and diversification framework (not counting the construction of the initial solutions). That is, if we are considering the important characteristics of one component, we are keeping the other components fixed. But, obviously, the optimization of one component depends on the choices made for the others. For example, a good perturbation must have the property that it cannot be (easily) undone by the intensification algorithm. This should be taken into consideration when we investigate new enhancements of ITS.

4. Concluding remarks

The purpose of this paper was to introduce a new improved version of the iterated tabu search (ITS) method for the well-known combinatorial optimization problem, the traveling salesman problem.

ITS is a promising extension to the ordinary tabu search scheme. It seeks near-optimal solutions by combining intensification (standard tabu search) and diversification (perturbation of solutions) in a proper way. The goal of the intensification is the search for a (better) locally optimal solution in the neighbourhood of the current solution, while the diversification is responsible for escaping from the current local optimum and exploring new regions of the solution space. For the TSP, the essential improvement is achieved due to decomposition of the neighbourhood of solutions, which allows significant speeding-up of the tabu search process. The fast tabu search used, namely, in the role of intensification within ITS framework resulted in quite encouraging solutions, especially for the smaller TSP instances taken from the TSP instance library TSPLIB.

During the experiments, it was observed that, in most cases, the simulated annealing algorithm produces better results than the straightforward implementation of the tabu search. This fact implies the idea of combining the simulated annealing and tabu search, for example, the SA algorithm could be examined in the role of the robust diversification (perturbation) procedure within the FITS framework. Moreover, it may also be worthy trying the intensification and diversification paradigm with TS being substituted by SA.

These enhancements coupled with the additional ones (such as implementing other innovative perturbation operators, or incorporating new stagnation avoidance techniques) could be a proper subject for the future work. Another research direction may be related to the investigations of hybridization of FITS and population-based (genetic) algorithms.

References

[1] 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.

[2] J.L. Bentley. Experiments on traveling salesman heuristics. *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, 1990, 91–99.

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

[4] B. Codenotti, G. Manzini, L. Margara, G. Resta. Perturbation: an efficient technique for the solution of very large instances of the Euclidean TSP. *ORSA Journal of Computing*, 1996, *Vol*.8, 125–133.

[5] R. Durbin, D. Willshow. An analogue approach to the traveling salesman problem using an elastic net method. *Nature*, 1987, *Vol*.326, 689–691.

[6] C.-N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, 1994, *Vol*.51, 243–267.

[7] B. Freisleben, P. Merz. A genetic local search algorithm for solving symmetric and a symmetric traveling salesman problems. *Proceedings of the* 1996 *IEEE International Conference on Evolutionary Computation, Nagoya, Japan*, 1996, 616–621.

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

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

[10] F. Glover. Tabu search fundamentals and uses. *Working paper, University of Colorado, Boulder, USA*, 1995.

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

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

[13] 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.

[14] D.S. Johnson, L.A. McGeoch. The traveling salesman problem: a case study. *In E. Aarts, J.K. Lenstra (eds.), Local Search in Combinatorial Optimization, Wiley, Chichester*, 1997, 215–310.

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

[16] G. Laporte. The traveling salesman problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 1992, *Vol*.59, 231–247.

[17] S. Lin, B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 1973, *Vol*.21, 498–516.

[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] **O. Martin, S.W. Otto, E.W. Felten.** Large-step Markov chains for the traveling salesman problem. *Complex Systems*, 1991, *Vol*.5, 299–326.

[20] **A. Misevicius, D. Rubliauskas, J. Smolinskas.** Reconstruct and improve principle based algorithm for the quadratic assignment problem. *Information Technology and Control*, 2002, *No*.2(23), 7–17.

[21] **A. Misevicius.** Using iterated tabu search for the traveling salesman problem. *Information Technology and Control*, 2004, *No*.3(32), 29–40.

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

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

[24] **N. Mladenovic, P. Hansen.** Variable neighbourhood search. *Computers & Operations Research*, 1997, *Vol*.24, 1097–1100.

[25] **C. Nilsson.** Heuristics for the traveling salesman problem. *Tech. Report, Linköping University, Sweden,* 2003. `http://www.ida.liu.se/~TDDB19/reports_2003/htsp.pdf`.

[26] **J. Pepper, B. Golden, E. Wasil.** Solving the traveling salesman problem with annealing-based heuristics: a computational study. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 2002, *Vol*.32, 72–77.

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

[28] **G. Reinelt.** The traveling salesman: computational solutions for TSP applications. *Lecture Notes in Computer Science*, 1994, *Vol*.840, *Springer, Berlin*.

[29] **D.E. Rosenkrantz, R.E. Stearns, P.M. Lewis.** An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 1977, *Vol*.6, 563–581.

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

DOI: 10.5755/j01.itc.34.4.12028