ISSN 1392 – 124X INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 2004, Nr. 3(32)

USING ITERATED TABU SEARCH FOR THE TRAVELING SALESMAN PROBLEM[♣]

Alfonsas Misevičius

Kaunas University of Technology, Department of Practical Informatics
Studentų St. 50–400a/416a, LT–51368 Kaunas, Lithuania

Abstract. In this paper, we propose an iterated tabu search (ITS) algorithm for the well-known combinatorial optimization problem, the traveling salesman problem (TSP). ITS is based on so-called intensification (improvement) and diversification (reconstruction) (I&D) paradigm. The goal of the intensification is the search for a locally optimal solution in the neighbourhood of the current solution. The diversification is responsible for escaping from the current local optimum and moving towards new regions in the solution space. Using the limited standard tabu search (TS) in the role of an effective intensification (local improvement) procedure resulted in promising solutions obtained during the experimentation with a number of the test data from the library of TSP instances TSPLIB. The results show that the proposed variant of ITS outperforms both the straightforward TS algorithm and the other heuristic algorithms tested.

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

Indroduction

The traveling salesman problem (TSP) can be formulated as follows. Given matrix $\boldsymbol{D} = (d_{ij})_{n \times n}$ and the 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 of distances between all pairs of these 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 called edges. So, solving the TSP means searching for the shortest closed tour in which every city is visited exactly once.

The TSP is a typical problem of combinatorial optimization (CO). This means that theoretical and practical insight achieved in the study of the TSP can often be helpful in solving other problems in this area. This problem is easy to state, but hard to solve. It has been proved that the TSP is NP-hard [9], and cannot be solved to optimality within polynomially bounded computation time. Therefore, heuristic algorithms have to be used in order to find near-optimal (locally optimal) solutions. The heuristics [25, 28, 30] are not able to guarantee that a problem will be solved in terms of obtaining the exact solution (it may not even be possible to state how close to optimality a particular solution is); however, heuristics seek good, high quality solutions at a reasonable computational cost (time).

Many heuristics have been "tailored" especially for the TSP, among them, tour construction heuristics [3, 5, 33], descent local search (2-opt [6], 3-opt [20], k-opt (Lin-Kernighan like heuristics) [21, 23], etc.). TSP has also been widely used as a problem for testing various meta-heuristics, like simulated annealing [17, 29], tabu search [7, 18], genetic algorithms [8]. (For a more exhaustive list of the heuristics for the TSP, the reader is addressed to [16, 19 ,27, 32].)

We are also using this problem as a "platform" for investigation of the performance of optimization technique we call the iterated tabu search (ITS). The paper is organized as follows. In Section 1, the paradigm of iterated tabu search is outlined. A variant of ITS for the traveling salesman problem is discussed in Sections 2. In Section 3, we present some experimental results. Section 4 completes the paper with concluding remarks.

[♣] This work is supported by Lithuanian State Science and Studies Foundation through grant number T-04078.

1. The paradigm

Since the TSP is a representative example (instance) of combinatorial optimization problems, we will briefly introduce the basic definitions related to these problems. CO studies hard problems in which the task is to find "the best element" in a finite configuration (solution) space, with respect to some (real-valued) function. More formally, an instance of a CO problem can be described as a pair (S, f), where S is the set of feasible solutions (also called solution space), and f: $S \to \Re$ is the objective (cost) function which assigns a (real) value to each solution. (Without loss of generality, we assume that f seeks a global minimum.) The goal is to find a solution $s_{\text{opt}} \in S$ such that

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

The solution s_{opt} is called a globally optimal solution (global optimum) of the problem (S, f). $S_{\text{opt}} \subseteq S$ denotes the set of optimal solutions, and $f_{\text{opt}} = f(s_{\text{opt}})$ denotes the optimal value of the objective function (optimal cost).

In addition, a neighbourhood function Θ: $S \to 2^S$ is given: it attaches for each $s \in S$ a set $\Theta(s) \subseteq S$ – a 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.

TSP-heuristics operate rather upon pairs of elements ($j_1 = \pi(i)$, $j_2 = \pi(i+1)$) (i.e. edges) than single elements ($j = \pi(i)$). Taking this fact into account, the distance betwee 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 permutation (tour) [4]. Mathematically, the distance between permutations π and π' may be declared as $\rho(\pi, \pi') = |\Omega|$, where Ω is the set that consists of all possible pairs $(\pi(i), \pi((i \bmod n) + 1))$ $(i \in \{1, 2, ..., n\})$ such that $\exists j$:

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

or

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

We can then easily define the neighbourhood function Θ_λ of order λ ($1 < \lambda \le n$): $\Theta_\lambda(\pi) = \{\pi' \mid \pi' \in \Pi, \quad \rho(\pi, \pi') \le \lambda\}$, where π is from Π. If $\lambda=2$, one obtains 2-exchange neighbourhood function, which is often used in the TSP-heuristics. In this case, a move from the current permutation π to the neighbouring one $\pi' \in \Theta_2(\pi)$ may be described by using a perturbation operator (function) $p(\pi,i,j)$: $\Pi \times \mathrm{N} \times \mathrm{N} \to \Pi$, 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. In the other words, the pairs $(\pi(i),\pi(i+1))$ and $(\pi(j), \pi((j \bmod n) + 1))$ are deleted, and the pairs $(\pi(i), \pi(j))$ and $(\pi(i+1), \pi((j \bmod n) + 1))$ are added (see Figure 1). More specifically, $p(\pi,i,j)$ 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 $k \in \{1, ..., j - i - 2\}$, in order to guarantee $\rho(\pi, \pi') = 2$. (For a move from certain permutation to $p(\cdot, i, j)$, we will also use a compact notation m_{ij}. An expression $\pi' = \pi \oplus m_{ij}$ would mean that π' is obtained from π by applying $p(\pi, i, j)$.) Note that the formula for calculation the objective function (tour length) difference $\Delta z = z(p(\pi, i, j)) - z(\pi)$ is very simple (i.e. $z(p(\pi,i,j)) - z(\pi) = d_{\pi(i),\pi(j)} + d_{\pi(i+1),\pi((j \bmod n)+1)} - d_{\pi(i),\pi(i+1)} - d_{\pi(j),\pi((j \bmod n)+1)}$) and takes $O(1)$ time.

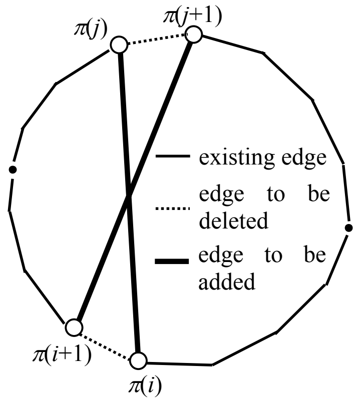

Figure 1. Deleting and adding edges in a tour

1.1. Standard tabu search

Before describing the tabu search (TS) method, let us recall for a moment the well-known descent local search (LS) heuristic (also known as hill climbing). The descent LS algorithm starts from an initial (maybe, randomly generated) solution $s°$. Further, the search process is continued by performing some sequential transformations 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 $\Theta(s)$. 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). So, 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 descent LS 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 $s' \in \Theta(s)$)). The whole process is continued until the current solution s becomes to be locally optimal, that is, no better solution exists in the neighbourhood of the current solution ($\forall s' \in \Theta(s)$: $f(s') \geq f(s)$). (Given a neighbourhood Θ_k, the solution obtained by descent LS may be regarded to as an optimal solution with respect to this neighbourhood, i.e. k-opt(imal) solution. Hence, the names of corresponding procedures: 2-opt, 3-opt, and so on.)

In some sense, tabu search [11,12] originates from the policy described above. However, the TS goes beyond this paradigm. In contrast to classical LS (which is limited to finding one locally optimal solution only), TS-based algorithms continue the search even if a locally optimal solution is found. Briefly speaking, TS is a process of subsequent moves from one local optimum to another. The best local optimum found during this process is the resulting solution of TS. Thus, TS is an extended descent local search. TS enables to escape local optima. Consequently, it explores much more larger part of the solution space when comparing with LS. Hence, TS offers more opportunities for discovering high quality solutions than traditional LS.

The central idea of the TS method 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 of the search. TS is based on a methodology of prohibitions: some moves are "frozen" (become "tabu") from time to time.

More formally, TS 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 certain subset $\Theta'(s)$ of the neighbouring solutions of the current solution is considered, and the move (to the solution $s' \in \Theta'(s) \subseteq \Theta(s)$) 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 TS algorithm chooses one that least degrades (increases) the objective function, i.e. 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 $|T|$ 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 in the last $|T|$ steps. However, the straightforward prohibition may sometimes lessen the efficiency of the search. 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 favourable 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

$$f(s') < f(s^*) \text{ or } \Big(s' = \operatorname*{arg\,min}_{s'' \in \Theta(s)} f(s'') \text{ and } s' \text{ (or "attribute" of } s' \text{) is not tabu} \Big). \tag{3}$$

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 (pure) tabu search paradigm is presented in Figure 2. More details on fundamentals of TS, its modifications and applications can be found in [13, 14].

1.2. Iterated tabu search

Although TS is a powerful optimization tool, it typically face, in its canonical form, less or more difficulties. Some of them are: a huge number of local optima over the solution space, cycles (i.e. repeating sequences) of the search configurations (states), and the phenomenon of so-called "deterministic chaos" (or chaotic attractors) [1]. 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 a limited region of the solution space. (The trajectories are random although the system (the set of solutions) is deterministic (finite).) So, the search trajectory will visit only a limited part of the solution space: if this portion does not contain the global minimum (optimum), it will never be found.

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

The standard (pure) TS goes beyond the descent LS, and ITS tries to go beyond the standard TS. The heart of ITS is the concept of intensification and diversification (I&D). The early origins of this concept go back to 1986 [2]. Since that time, various modifications and enhancements of the basic idea have been proposed, among them, iterated Lin-Kernighan algorithm [15], "large step Markov chains" [24], variable neighbourhood search [26], and, finally, iterated local search (ILS) [22].

```
function tabu_search(s);
  // input: s – the initial solution; output: $s^*$ – the best solution found //
  $s^*$ := s;
  initialize the tabu list T;
  repeat // continue the main cycle of TS //
    given neighbourhood function $\Theta$, tabu list T, and aspiration
criterion,
    find the best possible solution $s' \in \Theta'(s) \subseteq \Theta(s)$, where $\Theta'(s)$ consists of
    solutions that (or their "attributes") are not currently in the tabu
    list T or satisfy the aspiration criterion;
    s := s'; // replace the current solution by the new one //
    insert the solution s (or its "attribute") into the tabu list T;
    if $f(s) < f(s^*)$ then $s^*$ := s; // save the best so far solution //
    update the tabu list T
  until termination criterion is satisfied;
  return $s^*$
end.
```

Figure 2. Paradigm of standard tabu search

Very roughly, ILS can be thought of as a "ruin and recruit" principle based optimization policy. There are two main phases (components) in the ILS paradigm: 1) the reconstruction phase (it can be viewed as diversification of the search), and 2) the local improvement phase (it can be viewed as intensification of the search). The additional component is the selection of a candidate for the reconstruction. During the first phase, an existing solution is reconstructed (perturbed) in a proper way. In the second phase, one tries to improve the solution just "ruined" as best as one can; hopefully, the new (improved) solution is better than the solutions obtained in the previous iterations. By repeating these phases many times one seeks for high quality results. ITS is very similar to ILS. The main distinguishing feature of ITS is that the standard (or, maybe, modified) TS procedure plays a role of an effective intensification (i.e. local improvement).

So, 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 a "destruction", and a new solution, say $s^\sim$, is obtained. The goal of such a reconstruction 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. It is important, however, that a proper level of diversification is kept up. The reconstruction should be neither too strong, nor too weak – otherwise the resulting algorithm might be quite similar to a pure random ("blind") multistart, or the process would periodically return to solutions to which the reconstruction has been applied.

Regarding the solutions to be reconstructed, two 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. The exploration takes place if one of locally optimal solutions (not necessary the best local optimum) found so far is accepted as a candidate solution – in fact, each optimized solution can be considered as a potential candidate for the reconstruction. In the case of exploration, a variety of strategies may be used for the candidate selection. In the simplest case, so-called "where you are" (WYA) strategy is applied, i.e. every new local optimum is accepted. However, more sophisticated policies are available, for example, the selection from a pool (memory) of locally optimal solutions, like in the population based (genetic) algorithms.

The reconstructed solution $s^\sim$ serves as an input for the subsequent tabu search procedure, which starts immediately after the reconstruction 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. The new better solution (s^*) found at the current iteration is saved in a memory. This type of proceeding continues until a stopping condition is met, for example, a fixed number of iterations has been executed. The pseudo-code of the ITS paradigm is shown in Figure 3.

```
function iterated_tabu_search(s);
   // input: s – the initial solution; output: s* – the best solution found //
   s• := tabu_search(s); // improve the initial solution by using TS procedure, get the resulting solution s• //
   s := s•; s* := s•;
   repeat // continue the cycle of the iterated tabu search //
      s := candidate_acceptance(s•,s, ...); // select a solution for the subsequent reconstruction //
      s~ := reconstruction(s); // "ruin" the selected solution, obtain a new solution s~ //
      s• := tabu_search(s~); // improve the solution s~ by TS, get the resulting optimized 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 3. Paradigm of iterated tabu search

It should be noted that ITS is a general purpose meta-heuristic – not a pure heuristic algorithm. Such a meta-heuristic succeeds in search if only involves the specific problem knowledge. So, the local improvement (intensification), as well as the reconstruction (diversification) procedures must be "tailored" for a particular problem (i.e. must be as much problem-oriented as possible), while the framework itself is, in general, invariable.

2. A variant of iterated tabu search for the TSP

2.1. Initial solution construction

The question of whether or not to use a tour construction heuristic for the initial solution is not that simple to answer. For example, Reinelt [32] found that is better to start with an efficient construction heuristic. However, Lin and Kernighan [21] concluded that the use of sophisticated construction heuristic is just wasting time. Besides, the construction heuristics are usually deterministic, so it may not be possible to obtain more than one different solution.

In our algorithm, the initial solutions are generated in a random way, although using construction heuristics may result in a significant reduction of CPU time.

2.2. Local improvement (intensification)

In ITS, the local improvement is based on the standard tabu search. In fact, we apply only short runs of the TS procedure – we call this approach a limited tabu search (LTS). The experiments have demonstrated that these limited iterations allow saving considerable amount of CPU time; on the other hand, LTS in combination with diversification operators is quite enough to seek for near-optimal solutions. Another modification is related to the way which TS explores the neighbourhood. (Note that we use the 2-exchange neighbourhood Θ_2.) So, instead of exploring each time the complete neighbourhood Θ_2, we apply the TS runs for some smaller portions of this neighbourhood (again, this is done for the sake of reduction of CPU time). The portions are processed sequentially, one by one. The size of these portions θ can be defined by a user-defined parameter.

The tabu list is organized as an $n \times n$ matrix $\boldsymbol{T}$ of integer numbers. At the beginning, all the entries of $\boldsymbol{T}$ are set to zero. As the search progresses, the entries t_{ij} store the current number of the iteration plus the tabu list factor h. In this case, a move m_{ij} (as described in Section 1) is tabu if the value of t_{ij} is equal or greater than the current iteration number. Note, that by using the matrix based tabu list, testing whether a move is tabu or not requires only one comparison (i.e. it is performed in a constant time). The detailed template of the TS procedure for the TSP is given in Figure 4.

We also propose to add an additional component (feature) to this TS procedure. The idea is to use an alternative intensification mechanism under certain circumstances, for example, if the current difference in the objective function values is negative. In this case, the basic TS procedure is temporally "interrupted" in order to apply 2-opt (or other fast LS) procedure. The reason is to prevent accidental miss of local optima and to intensify search at the moments of decreasing of the objective function. This alternative intensification is omitted if it already took place within the last r iterations. Here, r can be related to the current tabu list size h: $r = \omega h$, where ω is the alternative intensification frequency factor. The template of the 2-opt procedure is presented in Figure 5.

```
function $TS(\pi,n,\tau,\theta)$; // the tabu search procedure for the TSP //
  // input: $\pi$ – the current permutation (tour); $n$ – the problem size; $\tau$ – the number of iterations; //
  //        $\theta$ – the size of the portions of the neighbourhood $\Theta_2$; $i, j$ – the current move indices; //
  //        $h$ – the current tabu list size; $\omega$ – the alternative intensification frequency factor //
  // output: $\pi^\bullet$ – the best solution found //
  $\pi^\bullet := \pi$; $T := 0$; $c := 1$; $c' := 1$; $n' := if(i = 1, n-1, n)$ ; $r := \omega h$; $improved :=$ "FALSE";
  while $(c \le \tau)$ or $improved =$ "TRUE" then begin // main cycle //
    $\Delta z_{min} := \infty$;
    for $w := 1$ to $\theta$ do begin // find the best non-tabu move in the part of $\Theta_2(\pi)$ //
      $i := if(j < n', i, if(i < n-2, i+1, 1))$ ; $n' := if(i = 1, n-1, n)$ ; $j := if(j < n', j+1, i+2)$ ;
      $\Delta z := z(\pi \oplus m_{ij}) - z(\pi)$;
      $tabu := if(t_{ij} \ge c,$ "TRUE", "FALSE"); $aspired := if(z(\pi) + \Delta z < z(\pi^\bullet),$ "TRUE", "FALSE");
      if $((\Delta z < \Delta z_{min})$ and not $tabu)$ or $aspired$ then begin $\Delta z_{min} := \Delta z$; $u := i$; $v := j$
end
    end; // for //
    $improved := if(\Delta z_{min} < 0,$ "TRUE", "FALSE");
    if $\Delta z_{min} < \infty$ then begin
      $\pi := \pi \oplus m_{uv}$ ; // replace the current permutation (tour) by the new one //
      $t_{uv} := c + h$ // make the corresponding move tabu //
    end; // if //
    if $improved =$ "TRUE" and $(c - c' \ge r)$ then begin $\pi := TWO\_OPT(\pi,n)$; $c' := c$ end;
    if $z(\pi) < z(\pi^\bullet)$ then $\pi^\bullet := \pi$; // save the best so far permutation (tour) //
    $c := c + 1$
  end; // while //
  return $\pi^\bullet$
end.
```

Figure 4. Pseudo-code of the tabu search procedure for the TSP

```
function $TWO\text{-}OPT(\pi,n)$; // 2-opt based on the steepest descent (best improvement) for the TSP //
  // input/output: $\pi$ – the initial/locally optimal permutation (tour); $n$ – the problem size //
  $k := 1$; $l := 2$; $n' := n-1$;
  repeat // main cycle //
    $\Delta z_{min} := 0$; // $\Delta z_{min}$ – the minimum difference of the objective function values //
    for $w := 1$ to $|\Theta_2|$ do begin
      $k := if(l < n', k, if(k < n-2, k+1, 1))$ ; $n' := if(k = 1, n-1, n)$ ; $l := if(l < n', l+1, k+2)$ ;
      $\Delta := z(\pi \oplus m_{kl}) - z(\pi)$;
      if $\Delta < \Delta z_{min}$ then begin $\Delta z_{min} := \Delta$; $u := k$; $v := l$ end
    end; // for //
    if $\Delta z_{min} < 0$ then $\pi := \pi \oplus m_{uv}$ // replace the current permutation (tour) by the new one //
  until $\Delta z_{min} \ge 0$;
  return $\pi$
end.
```

Figure 5. Pseudo-code of the 2-opt algorithm for the TSP

2.3. Reconstruction (diversification)

For the solution reconstruction, we propose a procedure entitled as a randomized greedy reconnect. The procedure consists of three main steps (see also Figure 6): 1) "ruin" (destruction) of the current solution; 2) local improvement of the part (segment) of the solution; 3) rebuilding the solution. In more details, the current solution is disintegrated in a random way, so that a part of the solution is obtained. Further, a greedy local improvement based on the nearest neighbour (NN) algorithm [33] is applied to this segment. Finally, the locally optimized segment is copied back to its original position, i.e. the tour is reconnected (recreated). The only control parameter for the greedy reconnect procedure is the reconstruction level μ, i.e. the actual length of the tour segment to be processed by NN. We let the parameter μ vary in some interval $[\mu_a, \mu_b] \subseteq [3, n]$; here, the values of μ_a, μ_b can be related to the problem size n, i.e. $\mu_a = \max(3, \lfloor \xi_1 n \rfloor)$ and $\mu_b = \max(3, \lfloor \xi_2 n \rfloor)$, where ξ_1, ξ_2 ($0 < \xi_1 \leq \xi_2 \leq 1$) are user-defined coefficients. Figures 7, 8 show the templates of the corresponding procedures in an algorithmic language form.

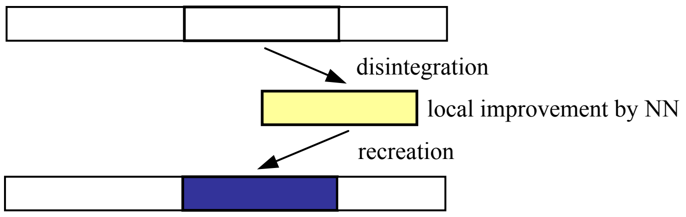

Figure 6. The graphical interpretation of the reconstruction process

```
function RGR($\pi$,$\mu$); // the randomized greedy reconnect procedure for the TSP //
  // input/output: $\pi$ – the current/reconstructed permutation (tour), $\mu$ – the reconstruction level //
  choose a position $i$ ($1 \leq i \leq n$) in $\pi$ at random;
  starting at the position $i$, copy $\mu$ items from $\pi$ to $\pi_{tmp}$
  (if $i + \mu - 1 > n$, the copying is done in a wrap-around fashion);
  $\pi'_{tmp} := NN(\pi_{tmp})$; // apply the nearest neighbour heuristic to $\pi_{tmp}$, get $\pi'_{tmp}$ //
  copy the contents of $\pi'_{tmp}$ back to $\pi$;
  return $\pi$
end.
```

Figure 7. Pseudo-code of the randomized greedy reconnect procedure for the TSP

```
function NN($\pi$); // the nearest neighbour heuristic for the TSP //
  // input: $\pi$ – the current permutation (tour); output: $\pi_{new}$ – the resulting permutation //
  select a random city $j$ from $\pi$;
  $\pi_{new}(1) := j$;
  for $i := 2$ to $|\pi|$ do begin // continue while there are any unvisited cities //
    find the nearest (to $j$) unvisited city $k$ in $\pi$;
    $\pi_{new}(i) := k$;
    $j := k$
  end; // for //
  return $\pi_{new}$
end.
```

Figure 8. Pseudo-code of the nearest neighbour heuristic for the TSP

In addition to the greedy reconnect procedure described, an alternative reconstruction mechanism may be applied. It takes place after the situation of stagnation of the search is detected. Stagnation is said to reveal itself if the best so far solution remains unchanged for a quite long time. Figure 9 depicts an example of a typical situation. In such situations, we simply make "cold restarts", i.e. generate new solutions from scratch.

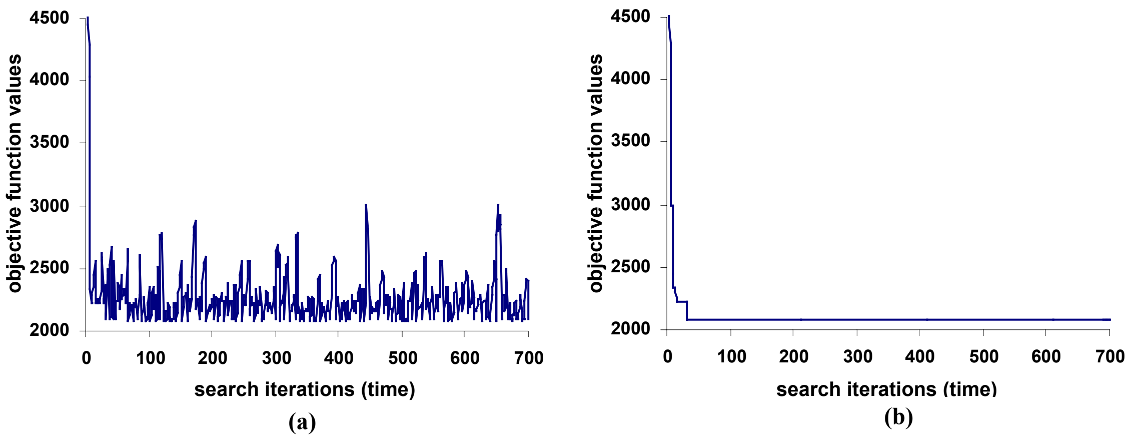

Figure 9. Illustration of the stagnation situation:
(a) cost (objective function value) vs time (search iteration); (b) the BSF solution cost vs time.
Note. As a data source, the TSP instance gr24 from TSPLIB [31] was used

2.4. Candidate acceptance

In our implementation of the ITS algorithm, we have chosen the exploitation (see Section 1.2) as a candidate acceptance strategy. This means that the only actually best solutions (BSF-solutions) are candidates for the subsequent reconstruction. The reason was that we found that the exploitation strategy yielded better results than the alternative exploration (WYA) strategy. Speaking in an algorithmic language, the candidate solution at the qth iteration of ITS is defined according to the formula $\pi^{(q)} = if(z(\pi^{\bullet}) < z(\pi^{(q-1)}), \pi^{\bullet}, \pi^{(q-1)})$, where $\pi^{(q)}$ is the candidate solution at the current iteration, $\pi^{(q-1)}$ is the candidate solution before the last execution of the improvement (TS) procedure (the BSF solution), and $\pi^{\bullet}$ is the solution obtained by the improvement procedure.

```
function $ITS(\pi,n,Q,\tau,\theta,h,\omega,\xi_1,\xi_2)$;  // the iterated tabu search procedure for the TSP  //
  // input: $\pi$ – the current (initial) permutation (tour); $n$ – the problem size;  //
  //        $Q$ – the total number of iterations; $\tau$ – the number of iterations for the TS procedure;  //
  //        $\theta$ – the size of the portions of the neighbourhood $\Theta_2$; $h$ – the current tabu list size;  //
  //        $\omega$ – the alternative intensification frequency factor; $\xi_1$, $\xi_2$ – the reconstruction factors  //
  // output: $\pi^*$ – the best permutation found  //
  $\pi^{\bullet} := TS(\pi,n,\tau,\theta)$;  // improve the current (initial) solution preliminary  //
  $\pi := \pi^{\bullet}$; $\pi^* := \pi^{\bullet}$; $i := 1$; $j := 2$; $n' := n-1$;
  $\mu_a := max(3,\lfloor\xi_1 n\rfloor)$; $\mu_b := max(3,\lfloor\xi_2 n\rfloor)$; $\mu := \mu_a-1$;
  for $q := 1$ to $Q$ do begin  // main cycle  //
    if stagnation condition is met
      then begin generate new solution $\pi^{\sim}$; $\mu := \mu_a-1$ end
      else begin
        $\pi := if(z(\pi^{\bullet}) < z(\pi), \pi^{\bullet}, \pi)$;  // choose the candidate for the reconstruction  //
        $\mu := if(\mu < \mu_b, \mu+1, \mu_a)$;  // update the reconstruction level  //
        $\pi^{\sim} := RGR(\pi,\mu)$  // apply randomized greedy reconnect procedure to $\pi^*$, get $\pi^{\sim}$  //
      end;  // else  //
    $\pi^{\bullet} := TS(\pi^{\sim},n,\tau,\theta)$;  // try to improve the reconstructed solution  //
    if $z(\pi^{\bullet}) < z(\pi^*)$ then begin
      $\pi^* := \pi^{\bullet}$;  // save the best so far permutation  //
      $\mu := \mu_a-1$  // reset the reconstruction level //
    end  // if  //
  end;  // for  //
  return $\pi^*$
end.
```

Figure 10. Pseudo-code of the iterated tabu search procedure for the TSP

The template of the resulting algorithm is presented in Figure 10. The default values of the control parameters for the ITS algorithm are: $Q = Cn$ ($C = 1 \div 3$); $\tau = 0.6n$; $\theta = 0.15n$; $h = 0.25n$; $\omega = 0.08$; $\xi_1 = 0.30$; $\xi_2 = 0.35$.

3. Computational experiments

In order to evaluate the efficiency of the proposed algorithm, some experiments have been carried out. The well-known set of the test data taken from the library of the TSP instances TSPLIB [31] were used. The algorithms used in the experimentation are as follows:

1) the multi-start 2-opt (M-2-OPT) algorithm;
2) the 4-opt (4-OPT) algorithm;
3) the variant of a simulated annealing (SA) algorithm;
4) the variant of a standard (pure) tabu search (TS) algorithm;
5) the iterated tabu search (ITS) algorithm.

All the algorithms are coded by the author. The programming language Free Pascal is used.

The performance measures of the algorithms are: a) the average deviation of solutions from a provably optimal solution – $\bar{\delta}$ ($\bar{\delta} = 100(\bar{z} - z_{\text{opt}})/z_{\text{opt}}$ [%]), where $\bar{z}$ is the average objective function value (tour length) over 10 runs (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 [31])); b) the number of solutions that are within 1% optimality (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. Firstly, it can be seen that the 2-opt based multi-start appears better then the straightforward 4-opt in many cases. In turn, SA works better than the multi-2-opt. Unexpectedly, SA seems even to be better than the pure tabu search. Regarding the iterated tabu search, it clearly outperforms both the standard tabu search (see Figure 11) and all the other algorithms tested with respect to the performance measures used (first of all, the average deviation). Note that the efficiency of ITS can be improved even more by increasing the number of iterations or tuning the values of other control parameters. The enhancements of the performance by modifying/extending the components of ITS are also possible (see Section 4).

Some results of other known TSP-heuristics can be cited for the sake of more objectivity. In Knox's paper [18], a version of the standard tabu search based algorithm was proposed. The author reports few results for small TSP instances ($n \le 75$). The solutions are very close to optimal, however the CPU times are quite large (for example, more than 600 seconds are needed for an instance with 50 cities).

In [15], Johnson 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 ≤100. The other efficient implementation of Lin-Kernighan algorithm yielded solutions that were from 0.24% to 3.04% of optimal solutions on instances of size 48 to 226. In [8], very few results are given; for example, for the instance kroa100, the average quality of the solutions is 0.00% – this took 11 seconds.

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 experimentation)

Instance	***n***	z_{opt}	$\bar{\delta}$ [, $C_{1\%}/C_{\text{opt}}$,] time									
			M-2-OPT		**4-OPT**		**SA**		**TS**		**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**	*60*
d493	493	35002	5.55,	0/0, *170*	—		0.64,	10/0, *320*	2.76,	0/0, *900*	**0.28**,	10/1, *900*
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*

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 experimentation)

Instance	n	z_{opt}	$\bar{\delta}$ [, $C_{1\%}/C_{\text{opt}}$,] time				
			M-2-OPT	4-OPT	SA	TS	ITS
fl417	417	11861	2.19, 0/0, *100*	—	1.26, 4/0, *230*	1.92, 3/0, *550*	**0.05**, 10/5, *540*
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, *350*
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** *320*
gr229	229	134602	4.10, 10/0, *9.0*	—	0.61, 9/0, *47*	2.50, 1/0, *78*	**0.01**, 10/9, *380*
gr431	431	171414	5.34, 10/0, *110*	—	0.46, 10/0, *240*	3.86, 0/0, *730*	**0.23**, 10/1, *630*
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*
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** *32*
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** *24*
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*	**0.29**, 10/3, *180*
linhp318	318	41345	6.34, 0/0, *45*	—	2.50, 0/0, *115*	5.67, 0/0, *280*	**0.22**, 10/4, *170*
pcb442	442	50778	7.07, 0/0, *120*	—	0.55, 10/0, *250*	1.58, 0/0, *690*	**0.37**, 10/0, *510*
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** *16*
pr226	226	80369	1.23, 0/0, *9.5*	—	0.37, 10/0, *47*	2.22, 6/0, *66*	**0** *65*
pr264	264	49135	4.89, 0/0, *16*	—	0.05, 10/8, *66*	1.92, 3/1, *125*	**0** *21*
pr299	299	48191	5.04, 0/0, *30*	—	0.15, 10/1, *90*	4.22, 0/0, *200*	**0.01**, 10/7, *290*
pr439	439	107217	5.44, 0/0, *120*	—	1.85, 2/0, *240*	5.22, 0/0, *720*	**0.18**, 10/2, *740*
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*
rd400	400	15281	6.62, 0/0, *95*	—	0.49, 10/0, *220*	4.07, 0/0, *700*	**0.22**, 10/2, *560*
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** *30*
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*

In a more recent work [10], the tabu search like algorithm (called a complete local search) could 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).

4. Concluding remarks

The goal of this paper was rather testing the new optimization framework than creation of a highly refined algorithm for the traveling salesman problem. The iterated tabu search algorithm based on the improvement (intensification) and reconstruction (diversification) paradigm is proposed. The main idea is to try to achieve more efficiency by coupling the standard tabu search with the proper reconstruction (diversification) mechanism. The results obtained from the experiments with the TSP instances (especially, the smaller TSP instances) confirm that the performance of traditional TS is improved considerably if the pure TS acts as an effective local improvement procedure within the ITS paradigm.

Regarding possible extensions of ITS, the following directions of improvement may be proposed:

1) using the reactive tabu search (instead of the straightforward tabu search) as a probably more efficient local improvement procedure;
2) implementing other, more elaborated and robust reconstruction (diversification) operators within ITS; 3) trying new efficient "cold restart" techniques;
4) maintaining "the history" (the (long-term) memory) of the locally optimal solutions;
5) incorporating the limited runs of ITS into other meta-heuristics, for example, genetic (memetic) algorithms.

In addition to that, it would be interesting using other more "tailored" algorithms (for example, the Lin-Kernighan heuristic) in the role of an intensification procedure.

The results of ITS for the TSP are promising. So, it may be worthy applying the ITS-based approach to other difficult combinatorial optimization problems. This could by a subject of the future work.

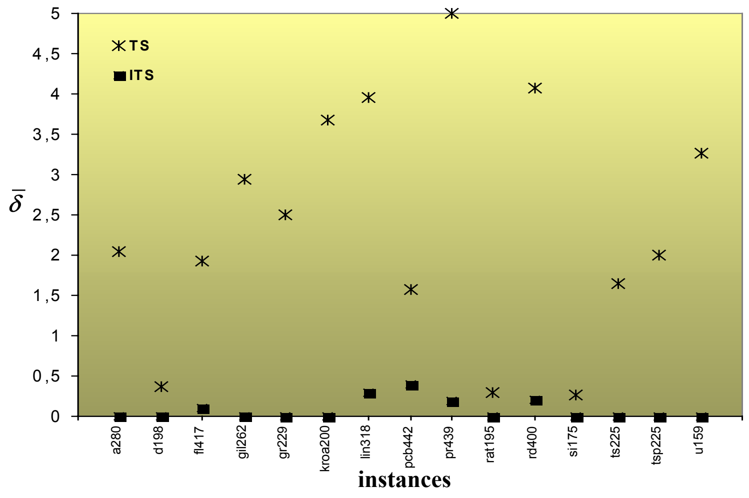

Figure 11. Illustration of results of the standard and iterated tabu search algorithms

References

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

[2] **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.

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

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

[5] **N. Christofides.** Worst-case analysis of a new heuristic for the traveling salesman problem. *Tech. Report* 388, *Carnegie Mellon University, Pittsburgh, USA*, 1976.

[6] **G.A. Croes.** A method for solving traveling-salesman problems. *Operations Research*, 1958, *Vol.*6, 791 - 812.

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

[8] **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.

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

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

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

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

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

[14] **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.

[15] **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.

[16] **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.

[17] **S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi.** Optimization by simulated annealing. *Science*, 1983, *Vol.*220, 671 - 680.

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

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

[20] **S. Lin.** Computer solutions of the traveling salesman problem. *Bell System Tech. Journal*, 1965, *Vol.*44, 2245 - 2269.

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

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

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

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

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

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

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

[28] **I.H. Osman, J.P. Kelly.** Meta-heuristics: an overview. *In I.H.Osman, J.P.Kelly (eds.), Meta-Heuristics: Theory and Applications, Kluwer, Norwell*, 1996, 1 - 21.

[29] **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.

[30] **C.R. Reeves.** Modern heuristic techniques. *In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, G.D. Smith (eds.). Modern Heuristic Search Methods, Wiley, Chichester*, 1996, 1–25.

[31] **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/`.]

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

[33] **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.