ISSN 1392–124X (print), ISSN 2335–884X (online) INFORMATION TECHNOLOGY AND CONTROL, 2013, Vol. 42, No. 3

Modified Local Search Heuristics for the Symmetric Traveling Salesman Problem

Alfonsas Misevičius[1], Andrius Blažinskas[2], Antanas Lenkevičius[3]

Kaunas University of Technology, Department of Multimedia Engineering,
Studentų st. 50-400a/416a, LT-51368 Kaunas, Lithuania
e-mail: [1]alfonsas.misevicius@ktu.lt, [2]andrius.blazinskas@ktu.lt, [3]antanas.lenkevicius@ktu.lt

crossref http://dx.doi.org/10.5755/j01.itc.42.3.1301

Abstract. In this paper, we investigate some modified local search (LS) heuristics for the solution of symmetric traveling salesman problem (TSP). These modifications are mainly due to the use of extended neighborhood structures. In addition, we are concerned with several new sets of the moves (transitions of solutions) based on the extended configurations of edge exchanges. We are also examining the performance of these extensions being used in an iterated local search (ILS) paradigm. The results from the experiments with the benchmark TSP instances from the TSP library (TSPLIB) demonstrate that the introduced improvements enable to seek solutions of higher quality without substantially increasing computational complexity.

Keywords: artificial intelligence; heuristics; local search; combinatorial optimization; traveling salesman problem.

1. Introduction

The traveling salesman problem (TSP) is stated as follows. Given the distance matrix $\boldsymbol{D} = (d_{ij})_{n\times n}$ and the set Π_n of all possible permutations of the integers from 1 to n, find a permutation $p \in \Pi_n$ that minimizes the following objective function:

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

Each permutation can be viewed as a tour through n cities such that each city is visited exactly once. The element of the permutation, $c_i = p(i)$, denotes city c_i to visit at step i, where $i = 1, 2, \ldots, n$. The pairs $(p(1), p(2))$, ..., $(p(i), p(i+1))$, ..., $(p(n), p(1))$ are called edges. The distances between cities are stored in the matrix $\boldsymbol{D}$ with elements d_{ij} [1]; then the entry $d_{c_i, c_{i \bmod n+1}}$ ($i = 1, 2, ..., n$) denotes the length of the ith edge and $z(p)$ is the total length of the tour p. Thus, solving the TSP means searching for the tour of minimal total length so that every city is visited only once and the salesman returns back to the starting city at the end of the trip.

The traveling salesman problem has been intensively studied during more than past 50 years and is one of the central problems in combinatorial optimization [2, 7, 10]. The TSP and its variants have important practical applications in various areas (like vehicle routing, clustering, job-shop scheduling) [13]. On the other hand, the TSP is a suitable platform for both theoretical and experimental investigations of intelligent computer-based methods, including exact and heuristic/metaheuristic algorithms [11, 12].

Among heuristic algorithms, local search (LS)-based (also known as neighborhood search) algorithms have been shown to be quite effective. Many of them are based on edge or chain exchange mechanisms (like 2-opt, 3-opt or generalized r-opt procedures [14]) and remain popular due to their ease of implementation. The most efficient local search approaches originate from the widely approved Lin-Kernighan (LK) algorithm [15], which may be viewed as a dynamic r-opt procedure. However, the Lin-Kernighan algorithm and its enhanced variants [3, 8, 9, 24] require a quite considerable amount of sophistication regarding the data structures and programming techniques, which makes them hard to implement and replicate. To address these issues, the researchers have considered the simpler versions of LK-heuristic [17] or modified edge/chain exchange algorithms (such as guided local search [29] or so-called Or-opt heuristics [4]). Also, other LS-based methodological modifications have been proposed, for example, local search with search space smoothing [6], iterated/combined local search [19, 28], ejection chain/stem-and-cycle methods [5, 25].

[1] Only symmetric problems are considered in this paper, i.e. $d_{ij} = d_{ji}$, $i = 1, 2, \ldots, n$.

In this paper, we are concerned with how the improved results can be achieved by incorporating some more new modifications into the traditional local search frame. The paper is structured as follows. In Section 1, we are giving preliminary definitions and outlining the general local search paradigm in the context of symmetric traveling salesman problems (STSPs). The new modified local search approaches for the STSP are detailed in Section 2. In Section 3, we present the results of the computational experiments with the proposed heuristics on the benchmark problems taken from the traveling salesman problem library — TSPLIB. The paper is completed with concluding remarks.

2. Preliminaries

The Hamming distance between two TSP tours p and p' can be defined as

$$\rho(p, p') = |\Omega|; \qquad (2)$$

where Ω is the set that consists of all possible pairs of cities (edges) $(p(i), p(i \bmod n + 1))$ $(i = 1, 2, ..., n)$ such that $\neg \exists j$:

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

$$(p(i), p(i \bmod n+1)) = \begin{cases} (p'(j), p'(j-1)), 1 < j \le n \\ (p'(j), p'(n)), \ j = 1 \end{cases}. \qquad (3)$$

Roughly speaking, the Hamming distance is the number of edges that are contained in one tour, but not in the other.

A neighborhood function Θ: $\Pi_n \to 2^{\Pi_n}$ assigns for every p from Π_n a set $\Theta(p) \subseteq \Pi_n$ — the set of neighboring solutions of p. An example of the neighborhood function is the 2-(edge-)exchange neighborhood Θ_2, which is frequently used for the TSP. The neighborhood Θ_2 can be formally described in the following way: $\Theta_2(p) = \{p' \mid p' \in \Pi_n, \ \rho(p, p') = 2\}$, where ρ denotes the Hamming distance as defined by formulas (2), (3). Each tour, p', of $\Theta_2(p)$ may be obtained from p by an operation that is referred to as a 2-(edge-)exchange move (solution transition). The 2-edge-exchange move can mathematically be formulated as a mapping Φ: $\Pi_n \times \mathrm{N} \times \mathrm{N} \to \Pi_n$, which gives for every tour $p \in \Pi_n$ a neighboring tour $p' \in \Theta_2(p) \subseteq \Pi_n$ such that $p'(i) = p(i)$, $p'(i+1) = p(j)$, $p'(j) = p(i+1)$, $p'(j \bmod n + 1) = p(j \bmod n + 1)$, where $1 \le i, j \le n \wedge 1 < j - i < n - 1$; in addition, if $j - i - 2 \ge 1$ then $p'(i + k + 1) = p(j - k)$ for $k = 1, ..., j - i - 2$. Briefly speaking, the pair of existing edges $(p(i), p(i+1))$, $(p(j), p(j+1))$ are removed from the tour and two new different edges $(p(i), p(j))$, $(p(i+1), p(j+1))$ are added (see Figure 1). A more compact form of notation, for example, $p \boxplus move(i, j)$ (or $\odot(p, i, j)$), may be applicable for this type of moves. Note that the 2-exchange move is symmetric as $p' = p \boxplus move(i, j) \to p = p' \boxplus move(i, j)$.

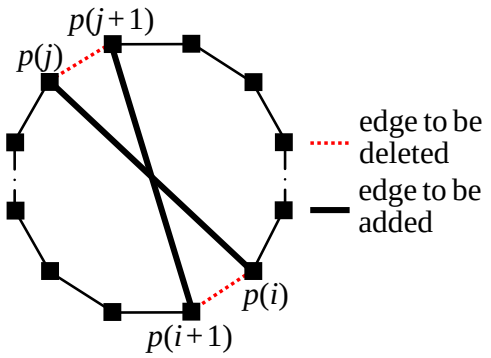

Figure 1. Illustrative example of the 2-edge-exchange move

The solution (tour) $p^\bullet$ is locally optimal with respect to the neighborhood Θ_2 (also referred to as a 2-opt(imal) solution) if $z(p^\bullet) \le z(p)$ for any $p \in \Theta_2(p^\bullet)$, i.e. $z(p^\bullet) \le z(p^\bullet \boxplus move(i, j))$ for $i = 1, ..., n - 2$, $j = i + 2, ..., n - 1 + \mathrm{sign}(i - 1)$. A 2-opt solution may be found by starting from an arbitrary initial solution and trying to iteratively apply 2-exchange moves (2-opt moves) until no possible move yields a better value of the objective function (z) (i.e., negative value of $z(p \boxplus move(i, j)) - z(p)$, where p denotes the currently processed solution). This iterative process is generally known as a 2-opt algorithm (procedure) (i.e. local search algorithm using the neighborhood Θ_2 and descending 2-exchange moves). There are two strategies: a) a greedy-first-accept (GFA) strategy selects the first neighbor $p' \in \Theta_2(p)$ for which $z(p') < z(p)$; b) a greedy-best-accept (GBA) strategy selects a neighbor $p' \in \Theta_2(p)$ such that $p' = \arg\min_{p \in \Theta_2(p)} z(p)$. The detailed formalized pseudo-code of the GBA-based 2-opt algorithm is shown in Figure 2 (see also [22]).

The run time of the 2-opt algorithm can be reduced by introducing the following modifications (see also [22]):

- using the restricted exploration of the 2–exchange neighborhood;
- applying the limited number of the descending 2-exchange moves.

In the first case, the neighborhood exploration is confined to the nearest cities of the current city. The nearest-neighbor list of the cities, i.e. the candidate list, CL, takes $O(Kn)$ memory (where $K = |CL|$) and its construction takes $O(n^2 \log_2 n)$ time (the construction takes place only once before initializing the algorithm). The size of the candidate list is arbitrarily chosen by the algorithm's user/analyst. (We used $K = 8$.)

In the second case, the number of descending 2-exchange moves, λ, is also controlled by the user. (We used $\lambda = 30$.)

The resulting user-controlled procedure can be called a "fast λ-descent" or a "fast local search" (FLS). The pseudo-code of the GBA-based fast local search algorithm is given in Figure 3.

```
function local_search_in_Θ2(p);
// input: p – initial (starting) solution (tour); output:
p – resulting solution (locally optimal solution with
respect to Θ2)
begin
  repeat
    Δmin := 0; //Δmin denotes the minimum
difference in the objective function values
    for i := 1 to n – 2 do
      for j := i + 2 to n – 1 + sign(i – 1) do
begin
        Δ := z(p ⊞ move(i, j)) – z(p); if
Δ < Δmin then begin Δmin := Δ; k := i; l := j
end
      end; // for
    if Δmin < 0 then p := p ⊞ move(k, l) //
move from the current solution to a new one
  until Δmin = 0;
  return p
end.
```

Figure 2. Pseudo-code of the greedy-best-accept-based local search algorithm using the neighborhood Θ_2

```
function fast_local_search_in_Θ2(p, λ);
//input: p – starting solution (tour), λ – (maximum)
number of descents (λ ≥ 1); output: p – resulting
solution
//auxiliary variables: CL – candidate list (nearest-
neighbor list of the cities)
begin
  number_of_moves := 0; for i := 1 to n
do index[p[i]] := i;
  repeat
    Δmin := 0;
    for u := 1 to n do
      for v := 1 to |CL| do begin
        i := min(u,index[CL[p[u],v]]);
j := max(u,index[CL[p[u],v]]);
        if (i + 2 ≤ j) and
(j ≤ n – 1 + sign(i – 1) then begin
          Δ := z(p ⊞ move(i, j)) – z(p); if
Δ < Δmin then begin Δmin := Δ; k := i; l := j
end
        end // if
      end; // for
    if Δmin < 0 then begin
p := p ⊞ move(k, l); update index;
number_of_moves := number_of_moves + 1 end //
if
  until (number_of_moves = λ) or (Δmin = 0);
  return p
end.
```

Figure 3. Pseudo-code of the greedy-best-accept-based fast local search algorithm using the neighborhood Θ_2

3. Modifications of the local search heuristics for the symmetric TSP

3.1. Extended neighborhoods

Our idea is to allow a composition of the simpler neighborhoods (like Θ_2) and construction of more complex neighborhood topologies with the compounded neighborhoods consisting of several sub-neighborhoods.

So, let p be a feasible tour (permutation) from Π_n; also, let $z(p)$ and $\Theta_2(p)$ denote, respectively, the objective function value and the 2-edge-exchange neighborhood for the tour p as defined above in the previous sections. Then, the extended (compounded) 2-edge-exchange neighborhood (denoted as $\Theta_{2\oplus 2}$) is formally described as follows:

$$\Theta_{2\oplus 2}(p) = \Theta_2(p) \cup \{p^{\wedge} \mid p^{\wedge} \in \Pi_n, \; p^{\wedge} \neq p, \rho(p^{\wedge}, p^{\blacksquare}) = 2\}, \quad (4)$$

where ρ denotes the Hamming distance and $p^{\blacksquare} = \arg\min_{p^{-} \in \Theta_2^{-}(p)} z(p^{-})$, $\Theta_2^{-}(p) = \Theta_2(p) \setminus \{\arg\min_{\dot{p} \in \Theta_2(p)} z(\dot{p})\}$.

The graphical view of the neighborhood $\Theta_{2\oplus 2}$ is depicted in Figure 4.

Further, the extended neighborhood $\Theta_{2\oplus 2\oplus 2}$ can be introduced (see also Figure 5):

$$\Theta_{2\oplus 2\oplus 2}(p) = \Theta_{2\oplus 2}(p) \cup \{p^{\cdot\wedge\wedge} \mid p^{\cdot\wedge\wedge} \in \Pi_n, \; p^{\cdot\wedge\wedge} \neq p, \rho(p^{\cdot\wedge\wedge}, p^{\cdot\blacksquare}) = 2\}, \quad (5)$$

where $p^{\cdot\blacksquare} = \arg\min_{p^{\cdot\cdot-} \in \Theta_2^{--}(p)} z(p^{\cdot\cdot-})$, $\Theta_2^{--}(p) = \Theta_2^{-}(p) \setminus \{\arg\min_{\ddot{p} \in \Theta_2^{-}(p)} z(\ddot{p})\}$. Very similarly, the neighborhood $\Theta_{2\oplus 2\oplus 2\oplus 2}$ (see Figure 6) and higher order neighborhoods may be defined.

Note that searching in such kind of neighborhoods conceptually resembles the "breadth-first search" policies used, for example, for searching in graph structures. The other type of extended neighborhoods may be introduced so that "walking" in these neighborhoods would have looked like the "depth-first search" (an alternative to the breadth-first search). An example of such a neighborhood is the neighborhood $\Theta_{2\otimes 2\otimes 2}$ which can mathematically be described in the following way (see also Figure 7):

$$\Theta_{2\otimes 2\otimes 2}(p) = \Theta_{2\otimes 2}(p) \cup \{p^{\cdot\wedge\wedge} \mid p^{\cdot\wedge\wedge} \in \Pi_n, \; p^{\cdot\wedge\wedge} \neq p, \rho(p^{\cdot\wedge\wedge}, p^{\cdot\blacksquare}) = 2\}, \quad (6)$$

where $p^{\cdot\blacksquare} = \arg\min_{p^{\cdot\cdot-} \in \Theta_2^{-}(p^{\blacksquare})} z(p^{\cdot\cdot-})$, $\Theta_2^{-}(p^{\blacksquare}) = \Theta_2(p^{\blacksquare}) \setminus \{\arg\min_{\ddot{p} \in \Theta_2(p^{\blacksquare})} z(\ddot{p})\}$, $p^{\blacksquare} = \arg\min_{p^{-} \in \Theta_2^{-}(p)} z(p^{-})$, $\Theta_2^{-}(p) = \Theta_2(p) \setminus \{\arg\min_{\dot{p} \in \Theta_2(p)} z(\dot{p})\}$.

(The neighborhood $\Theta_{2\otimes 2}$ is, in fact, the same as the

neighborhood $\Theta_{2\oplus 2}$ ($\Theta_{2\otimes 2}(p) \equiv \Theta_{2\oplus 2}(p) \ \ \forall\, p \in \Pi_n$).) Similarly to formula (6), the neighborhood $\Theta_{2\otimes 2\otimes 2\otimes 2}$ (see Figure 8) and the other higher order neighborhoods may be derived. In addition, many different combined variants are possible: $\Theta_{2\oplus 2\otimes 2}$, $\Theta_{2\otimes 2\oplus 2}$, $\Theta_{2\oplus 2\otimes 2\oplus 2}$, $\Theta_{2\otimes 2\otimes 2\oplus 2}$, and so on. As long as the number of the sub-neighborhoods covered remains constant, the time complexity of the neighborhood exploration is proportional to $O(n^2)$, where n is the problem size. In the case of the fast local search, the complexity reduces to $O(Kn)$, where K is the candidate list size.

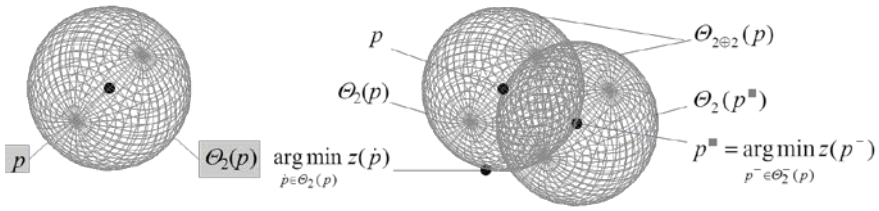

Figure 4. Graphical representation of the neighborhoods Θ_2 and $\Theta_{2\oplus 2}$

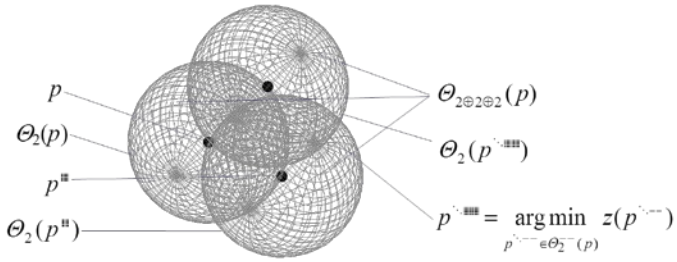

Figure 5. Graphical representation of the neighborhood $\Theta_{2\oplus 2\oplus 2}$

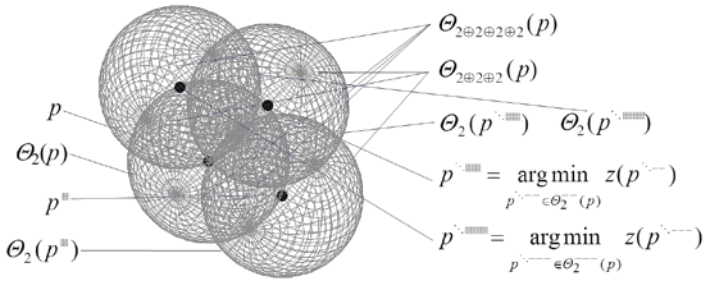

Figure 6. Graphical representation of the neighborhood $\Theta_{2\oplus 2\oplus 2\oplus 2}$

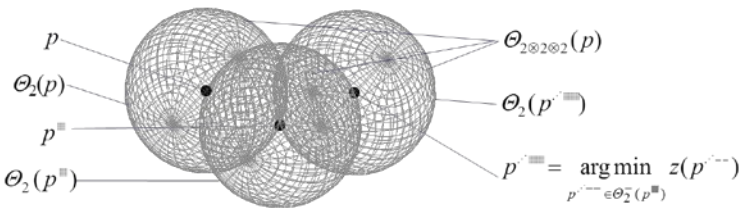

Figure 7. Graphical representation of the neighborhood $\Theta_{2\otimes 2\otimes 2}$

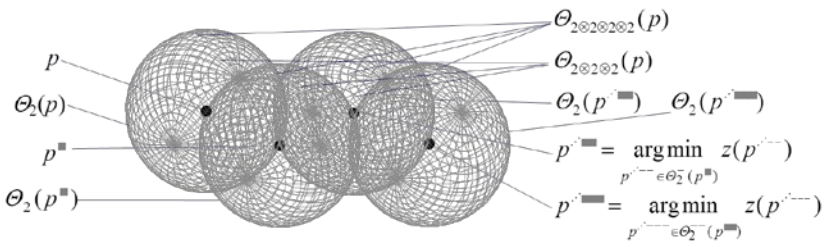

Figure 8. Graphical representation of the neighborhood $\Theta_{2\otimes 2\otimes 2\otimes 2}$

In Figure 9, we present the pseudo-code of the 2-opt local search algorithm using the neighborhood $\Theta_{2\oplus 2}$ ($\Theta_{2\otimes 2}$). Similarly, the algorithm for the fast local search in the neighborhood $\Theta_{2\oplus 2}$ ($\Theta_{2\otimes 2}$), as well as the algorithms for searching in the neighborhoods $\Theta_{2\oplus 2\oplus 2}$, $\Theta_{2\oplus 2\oplus 2\oplus 2}$, $\Theta_{2\otimes 2\otimes 2}$, $\Theta_{2\otimes 2\otimes 2\otimes 2}$, etc. can be described.

```
function local_search_in_Θ2⊕2(p);
// input: p – initial (starting) solution (tour); output:
p – resulting solution (locally optimal solution with
respect to Θ2⊕2 (Θ2⊗2))
begin
  repeat
    Δmin(1) := ∞; Δmin(2) := ∞; k(1) := 0;
l(1) := 0;
    for i := 1 to n – 2 do
      for j := i + 2 to n – 1 + sign(i – 1) do
begin
        Δ := z(p ⊞ move(i, j)) – z(p);
        if Δ < Δmin(1) then begin
Δmin(2) := Δmin(1); k(2) := k(1); l(2) := l(1);
Δmin(1) := Δ; k(1) := i; l(1) := j end
        else if Δ < Δmin(2) then begin
Δmin(2) := Δ; k(2) := i; l(2) := j end
    end; // for

p' := argmin(z(p ⊞ move(k(1), l(1))), z(local_search_i
n_Θ2(p ⊞ move(k(2), l(2)))));
    if z(p') < z(p) then p := p' //replace the
current solution by a new one
  until z(p') ≥ z(p);
  return p
end.
```

Figure 9. Pseudo-code of the greedy-best-accept-based local search algorithm using the neighborhood $\Theta_{2\oplus 2}$ ($\Theta_{2\otimes 2}$)

3.2. Extended moves

The set of the extended (modified) moves consists of the following subsets:

- subset of the 3-(edge-)exchange moves,
- subset of the 4-(edge-)exchange moves,
- subset of the 5-(edge-)exchange moves,
- subset of the 6-(edge-)exchange moves.

Each of the above subsets is rather a restricted set of moves built in a specific way than the set containing all possible move configurations. The subsets are created as described below.

The subset of the 3-exchange moves (or 3-exchange subset), denoted by $\Xi^{\backslash//}$, is constructed in such a way that each move is formed by removing three existing ("old") edges and adding three different ("new") edges. In particular, the separate edge $(p(i), p(i+1))$ and two adjoining (neighboring) edges $(p(j-1), p(j))$, $(p(j), p(j+1))$ are removed and the three new edges $(p(i), p(j))$, $(p(j), p(i+1))$, $(p(j-1), p(j+1))$ are added so that the feasibility of the tour is preserved (here, $j > i+1$) (see Figure 10).

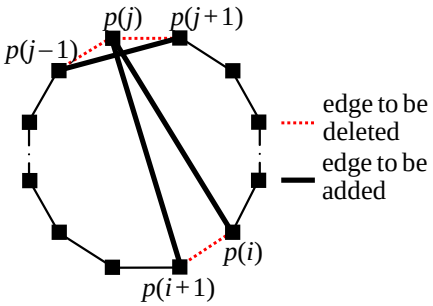

Figure 10. Illustrative example of the 3-edge-exchange move

The 4-exchange subset contains two other subsets denoted as $\Xi^{\backslash//}$ and $\Xi^{\backslash\backslash//}$. In both cases, the moves are constructed by removing four old edges and adding four new edges[2]. In the first case, the separate edge $(p(i), p(i+1))$ and three adjoining edges $(p(j-2), p(j-1))$, $(p(j-1), p(j))$, $(p(j), p(j+1))$ are deleted and the four new edges are added (here, $j > i+2$). There are 5 different variants for adding new edges (also see Figure 11(a)): 1) $(p(i), p(j))$, $(p(j), p(j-1))$, $(p(j-1), p(i+1))$, $(p(j-2), p(j+1))$; 2) $(p(i), p(j))$, $(p(j), p(j-2))$, $(p(i+1), p(j-1))$, $(p(j-1), p(j+1))$; 3) $(p(i), p(j-1))$, $(p(j-1), p(j))$, $(p(j), p(j-2))$, $(p(i+1), p(j+1))$; 4) $(p(i), p(j-1))$, $(p(j-1), p(j))$, $(p(j), p(i+1))$, $(p(j-2)), p(j+1))$; 5) $(p(i), p(j-2))$, $(p(i+1), p(j))$, $(p(j), p(j-1))$, $(p(j-1)), p(j+1))$.

In the second case, the two pairs of neighboring edges $(p(i-1), p(i))$, $(p(i), p(i+1))$ and $(p(j-1), p(j))$, $(p(j), p(j+1))$ are removed and the four new edges are added (here, $j > i+1$). There are 3 different variants for inserting new edges in this particular situation (also see Figure 11(b)): 1) $(p(i-1), p(j))$, $(p(j), p(i+1))$, $(p(j-1), p(i))$, $(p(i), p(j+1))$; 2) $(p(i-1), p(j))$, $(p(j), p(i))$, $(p(i), p(j-1))$, $(p(i+1), p(j+1))$; 3) $(p(i-1), p(j-1))$, $(p(i+1), p(j))$, $(p(j), p(i))$, $(p(i), p(j+1))$.

Regarding the 5-exchange subset ($\Xi^{\backslash\backslash///}$), the moves are formed by firstly deleting and then reconnecting five edges. In particular, the two adjoining edges $(p(i-1), p(i))$, $(p(i), p(i+1))$ and then three other adjoining edges $(p(j-2), p(j-1))$, $(p(j-1), p(j))$, $(p(j), p(j+1))$ are removed; after that, five edges are inserted (here, $j > i+2$). We have 17 different ways to reconnect the newly inserted edges (see Figure 12).

Finally, the two pairs of adjoining edges $(p(i-2), p(i-1))$, $(p(i-1), p(i))$, $(p(i), p(i+1))$ and $(p(j-2), p(j-1))$, $(p(j-1), p(j))$, $(p(j), p(j+1))$ are removed and the six new edges are added (here, $j > i+2$). This results in the 6-exchange subset, $\Xi^{\backslash\backslash////}$. In this case, the total number of the different configurations of edges increases to 106 (see Appendix, Figure A1).

[2] Some of edges may be simply "refreshed" (instead of inserting entirely new edges).

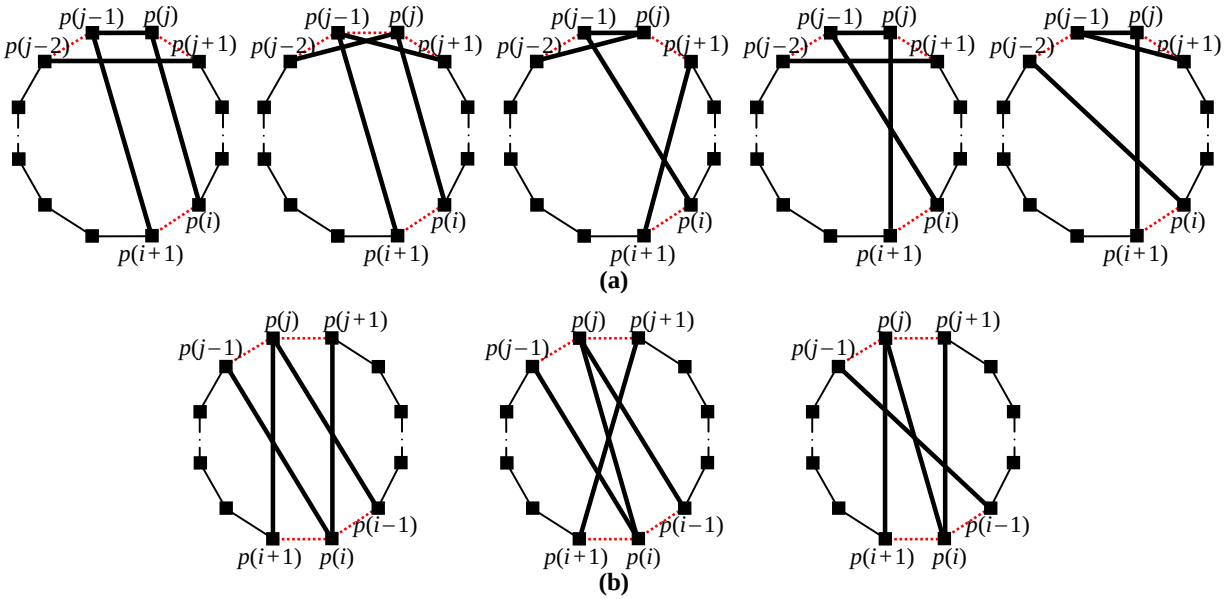

Figure 11. Illustrative examples of the different possible variants of 4-edge-exchange moves

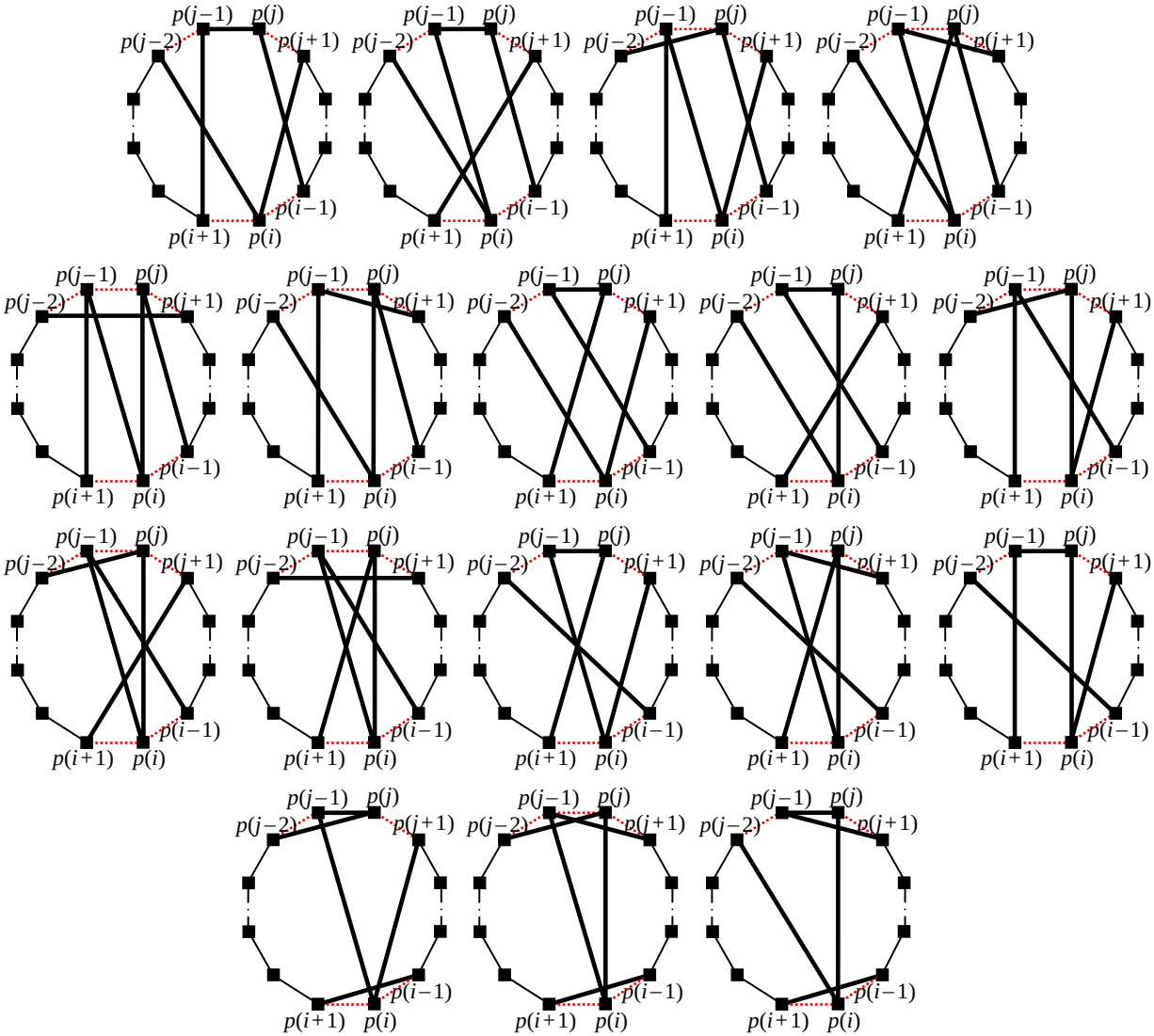

Figure 12. Illustrative examples of the different possible variants of 5-edge-exchange moves

Of course, we can take the union of all the subsets of moves ($\Xi^{\backslash//} \cup \Xi^{\backslash///} \cup \Xi^{\backslash\backslash//} \cup \Xi^{\backslash\backslash///} \cup \Xi^{\backslash\backslash\backslash///}$), including the 2-exchange moves; then the resulting search strategy would have resembled a very modified 6-opt like procedure.

There is a slight increase in the run time due to the testing of all possible configurations of the extended (modified) moves. The algorithm's framework itself remains, however, almost unchanged, except that the "$p \boxplus \text{move}(i, j)$"-type operations are turning to the operations of the type "$p \boxplus \text{move}(i, j, config^*)$". Here, $config^*$ is denoting the move configuration such that the move causes a tour of shortest length among all move configurations; that is, $config^* = \arg\min_{config=1,2,...TNF} \{z(p \boxplus \text{move}(i, j, config))\}$, where *config* denotes the current move configuration and *TNF* is the total number of configurations, which is actually equal to 133 (1 + 1 + 8 + 17 + 106), including the 2-exchange move.

The neighborhood exploration complexity is still $O(n^2)$ for the 2-opt local search and $O(Kn)$ for the fast local search.

3.3. Incorporating stochastic perturbations

Yet another way of enhancing the local search framework is to integrate (combine) the rigorous (greedy descent-based) and stochastic search processes. The resulting approach (known as an iterated local search (ILS) [16]) is not deterministic any more; however, it can lead to higher quality solutions. The example of the ILS algorithm is presented in a high-level pseudo-code form in Figure 13. Many variations of ILS are possible depending on the deterministic (descent) local search algorithm and stochastic perturbation technique at hand. In this work, we tried the greedy descent-based fast local search heuristics in the role of the descending algorithm; while, for the role of the perturbation mechanism, we used a special type of procedure — the so-called nearest-neighbor reconnection (NNR) procedure (see [22]). The NNR procedure is quite identical to the well-known nearest-neighbor (NN) algorithm [27], except that only a relatively small fraction (usually about 20-30%) of cities is involved in this procedure and the remaining larger part of the tour is not affected (a fraction is selected randomly from the tour).

```
function iterated_local_search(p, μ);
// input: p – initial (starting) solution (tour), μ – number of iterations (μ ≥ 1); output: p• – resulting solution
begin
  p′ := local_search_in_Θ2(p); // perform local search in a given neighborhood (starting from the initial solution p)
  p• := p′;
  current_number_of_iterations := 0;
  repeat

current_number_of_iterations := current_number_of_iterations + 1;
    p˜ := random_perturbation(p•); // perform random perturbation procedure (starting from p•)
    p′ := local_search_in_Θ2(p˜); // perform local search in a given neighborhood (starting from p˜)
    if z(p′) < z(p•) then p• := p′
  until current_number_of_iterations = μ;
  return p•
end.
```

Figure 13. Pseudo-code of the iterated local search algorithm

The limit of the run time of our ILS algorithms is predetermined by the number of iterations (μ) (see Figure 13), which can be flexibly tuned by the user.

The NNR-based perturbations take less than $O(n^2)$ time, so these perturbations do not significantly increase the overall computational complexity.

4. Results of computational experiments

We have examined our local search heuristic algorithms on the benchmark problem instances taken from the traveling salesman problem library — TSPLIB [26]. In the computational experiments conducted, the size of the instances varies between 96 and 1000 cities. The experiments were performed on a personal computer with an Intel Pentium IV 3 GHz single-core processor.

We used the following LS heuristics in our experimentation: 1) standard (2-opt) local search using the neighborhood Θ_2 (denoted as LS$\langle\langle\Theta_2\rangle\rangle$); 2) fast local search using the neighborhood Θ_2 (FLS$\langle\langle\Theta_2\rangle\rangle$); 3) iterated fast local search using the neighborhood Θ_2 (IFLS$\langle\langle\Theta_2\rangle\rangle$); 4) iterated fast local search using the neighborhood $\Theta_{2\oplus2\oplus2\oplus2}$ (IFLS$\langle\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\rangle$); 5) iterated fast local search using the neighborhood $\Theta_{2\otimes2\otimes2\otimes2}$ (IFLS$\langle\langle\ \Theta_{2\otimes2\otimes2\otimes2}\ \rangle\rangle$); 6) iterated extended fast local search using the neighborhood $\Theta_{2\oplus2\oplus2\oplus2}$ and the set of extended moves $\Xi^{\backslash//} \cup \Xi^{\backslash///} \cup \Xi^{\backslash\backslash//} \cup \Xi^{\backslash\backslash///} \cup \Xi^{\backslash\backslash\backslash///}$ (IEFLS$\langle\langle\ \Theta_{2\oplus2\oplus2\oplus2}\ \rangle\rangle$); 7) iterated extended fast local search using the neighborhood $\Theta_{2\otimes2\otimes2\otimes2}$ and the same set of extended moves (IEFLS$\langle\langle\Theta_{2\otimes2\otimes2\otimes2}\rangle\rangle$).

The first part of the experimentation was designed as follows. Let *R* be the pre-defined number of restarts, i.e. single applications of the algorithm to a given instance. At every restart, the algorithm starts from a new random initial solution. The current restart is interrupted as soon as the local optimum is found (or the maximum number of iterations is performed — which is the case of the iterated local search). The next restart is then started, and so on. The process stops when *R* restarts have been carried out. The best solution obtained during these restarts serves as a

resulting solution of the algorithm. This is repeated for each examined instance.

So, we used 10000 restarts of LS$\langle\!\langle\Theta_2\rangle\!\rangle$ and 100000 restarts of FLS$\langle\!\langle\Theta_2\rangle\!\rangle$. The computation time of the iterated local search algorithms (IFLS, IEFLS) is controlled by the number of iterations (μ). We used μ = 20000 for IFLS$\langle\!\langle\Theta_2\rangle\!\rangle$, μ = 5000 for IFLS $\langle\!\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\!\rangle$ and IFLS$\langle\!\langle\Theta_{2\otimes2\otimes2\otimes2}\rangle\!\rangle$, and μ = 500 for IEFLS$\langle\!\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\!\rangle$ and IEFLS$\langle\!\langle\Theta_{2\otimes2\otimes2\otimes2}\rangle\!\rangle$. In this way, all the algorithms consume more or less the same amount of computation (CPU) time.

The results of the experiments are presented in Table 1. Note that, in Table 1, "$\bar{\delta}$" is to denote the relative deviation of the obtained solutions from the provably optimal solution; $\bar{\delta}$ is calculated by the formula: $\bar{\delta} = 100(\bar{z} - z^{\diamond})/z^{\diamond}$ [%] , where $\bar{z}$ is the obtained value of the objective function (tour length) and $z^{\diamond}$ denotes the provably optimal objective function value (these values can be found in TSPLIB).

In addition, we have compared our best variant (IEFLS$\langle\!\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\!\rangle$) with other heuristic algorithms. In the experiments, the following seven heuristic algorithms (including IEFLS) have been tried: 1) 3-opt (LS$\langle\!\langle\Theta_3\rangle\!\rangle$); 2) simulated annealing (SA) [20]; 3) tabu search (TS) [23]; 4) genetic algorithm (GA) [21]; 5) Helsgaun's implementation of the Lin-Kernighan algorithm (HLK) [8]; 6) greedy randomized adaptive search procedure (GRASP) [18]; 7) IEFLS$\langle\!\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\!\rangle$. The run time for IEFLS$\langle\!\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\!\rangle$ remained the same as given in Table 1. The execution time for other algorithms was adjusted so that all the algorithms used approximately similar CPU time. The results of the comparison are presented in Table 2.

We may also be interested in the investigation of run time performance (run time distributions) instead of the solution quality. In this case, the so-called "time to target" methodology [1] can be applicable. For a given value of the objective function (target value), the algorithm is repeated multiple times and the run times of the algorithm to achieve this value are recorded and sorted. With the ith sorted time, a probability $P_i = \frac{i-0.5}{W}$ is associated, where i (i = 1, 2, ..., W) denotes the current run (trial) number and W is the total number of runs. The probabilities P_i are visualized by so-called time-to-target plots which show the probability that the target value will be achieved.

In Figure 14, we present the time-to-target plots for the algorithms LS$\langle\!\langle\Theta_2\rangle\!\rangle$, FLS$\langle\!\langle\Theta_2\rangle\!\rangle$, IFLS$\langle\!\langle\Theta_2\rangle\!\rangle$, and IEFLS$\langle\!\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\!\rangle$. The instance examined is gr96 and the target value is equal to 55485, which is 0.5% above the provably optimal solution value (55209). (We used 30 runs (W = 30), where each run consists of 10000 restarts of LS, 100000 restarts of FLS, 20000 iterations of IFLS, and 500 iterations of IEFLS.)

The performance improvement factor, *PIF*, of one algorithm (say, $\mathfrak{I}_1$) to another one (say, $\mathfrak{I}_2$) may be introduced by using the formula: $PIF = \frac{t_{0.5}(\mathfrak{I}_2)}{t_{0.5}(\mathfrak{I}_1)}$; here, $t_{0.5}$ denotes the time needed to obtain the given target value with the probability 0.5. In Table 3, we present the approximate values of the performance improvement factors for the algorithms LS$\langle\!\langle\Theta_2\rangle\!\rangle$, FLS$\langle\!\langle\Theta_2\rangle\!\rangle$, IFLS$\langle\!\langle\Theta_2\rangle\!\rangle$, and IEFLS$\langle\!\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\!\rangle$. In particular, the performance improvement factors PIF_1,

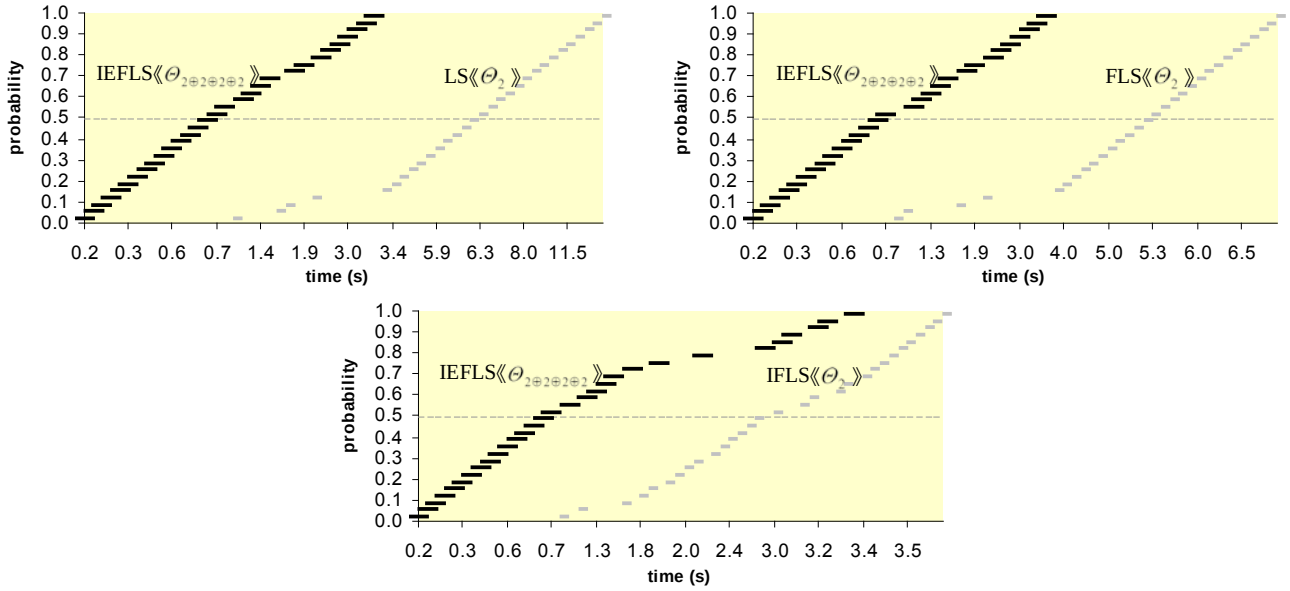

Figure 14. Time-to-target plots for the TSPLIB instance gr96

Table 1. Results of the experiments with the local search heuristics on TSPLIB instances

Instance name‡	$z^{\Diamond}$	$\bar{\delta}$							Averaged CPU time (s)
		LS $\langle\langle \Theta_2 \rangle\rangle$	FLS $\langle\langle \Theta_2 \rangle\rangle$	IFLS $\langle\langle \Theta_2 \rangle\rangle$	IFLS $\langle\langle \Theta_{2\otimes 2\otimes 2\otimes 2} \rangle\rangle$	IFLS $\langle\langle \Theta_{2\oplus 2\oplus 2\oplus 2} \rangle\rangle$	IEFLS $\langle\langle \Theta_{2\otimes 2\otimes 2\otimes 2} \rangle\rangle$	IEFLS $\langle\langle \Theta_{2\oplus 2\oplus 2\oplus 2} \rangle\rangle$	
bier127	118282	0.849	0.517	0.163	0.077	0.058	0.026	**0.000**	*2.6*
ch130	6110	0.999	0.699	0.552	0.251	0.170	0.063	**0.010**	*2.7*
d657	48912	5.732	2.650	1.335	0.629	0.256	0.230	**0.158**	*110*
dsj1000a	18659688	9.937	7.069	3.739	1.994	0.571	0.410	**0.334**	*500*
eil101	629	1.655	0.802	0.285	0.072	0.034	0.019	**0.004**	*2.1*
fl417	11861	2.597	1.403	0.656	0.251	0.088	0.055	**0.045**	*39*
gil262	2378	1.099	0.795	0.332	0.117	0.056	0.020	**0.012**	*8.8*
gr96	55209	0.997	0.724	0.301	0.056	**0.000**	**0.000**	**0.000**	*1.9*
gr120	6942	1.968	0.998	0.321	0.121	0.091	**0.022**	0.023	*2.4*
gr137	69853	0.824	0.768	0.057	0.042	0.046	0.054	**0.011**	*3.0*
gr229	134602	0.911	0.714	0.071	0.057	0.007	0.004	**0.002**	*8.5*
gr431	171414	3.045	1.789	0.660	0.359	0.202	0.067	**0.056**	*42*
kroa100	21282	0.073	0.060	0.053	0.050	**0.000**	**0.000**	**0.000**	*2.0*
krob100	22141	0.379	0.287	0.059	0.047	0.019	0.005	**0.000**	*2.0*
kroc100	20749	0.546	0.455	0.368	0.061	0.066	0.008	**0.000**	*2.0*
krod100	21294	1.538	1.202	0.575	0.274	0.199	0.093	**0.000**	*2.0*
kroe100	22068	0.983	1.302	0.493	0.283	0.253	0.142	**0.000**	*2.0*
lin105	14379	0.642	0.439	0.048	0.021	0.001	**0.000**	**0.000**	*2.1*
lin318	42029	1.202	0.801	0.390	0.121	0.064	0.049	**0.036**	*18*
pcb442	50778	5.185	3.004	0.738	0.409	**0.123**	0.134	0.128	*46*
pr107	44303	0.093	0.097	0.265	0.043	0.025	0.019	**0.006**	*2.2*
pr124	59030	0.953	0.650	0.244	0.135	0.098	0.047	**0.000**	*2.4*
pr264	49135	0.785	0.578	0.239	0.077	0.048	**0.010**	0.011	*9.0*
pr439	107217	4.462	2.922	0.668	0.411	0.260	**0.067**	0.074	*44*
rat99	1211	0.614	0.465	0.067	0.044	0.029	0.007	**0.000**	*2.0*
rat783	8806	7.472	3.159	1.634	0.702	0.404	**0.236**	0.239	*160*
rd100	7910	0.961	0.873	0.600	0.110	0.076	0.044	**0.001**	*2.0*
rd400	15281	1.543	1.322	0.734	0.156	0.089	**0.042**	0.044	*34*
si535	48450	6.489	3.030	1.134	0.655	0.174	0.157	**0.139**	*69*
tsp225	3916	0.505	0.413	0.100	0.010	0.003	0.002	**0.000**	*8.3*
	Average:	2.168	1.333	0.563	0.255	0.117	0.068	**0.044**	

‡ the numeral in the instance name indicates the size of the problem, i.e. the number of cities.

Table 2. Results of the comparison of different heuristic algorithms on TSPLIB instances

Instance name	$z^{\Diamond}$	$\overline{\delta}$							Averaged CPU time (s)
		LS $\langle\langle\Theta_3\rangle\rangle$	SA	TS	GA	HLK	GRASP	IEFLS $\langle\langle\Theta_{2\oplus2\oplus2\oplus2}\rangle\rangle$	
bier127	118282	0.752	0.133	0.029	0.034	**0.000**	**0.000**	**0.000**	*2.6*
ch130	6110	0.809	0.312	0.048	0.149	**0.000**	**0.000**	0.010	*2.7*
d657	48912	5.031	2.025	1.685	0.793	**0.000**	**0.000**	0.158	*110*
dsj1000a	18659688	9.565	3.002	4.013	2.032	0.030	**0.008**	0.334	*500*
eil101	629	0.850	0.127	0.016	0.023	**0.000**	**0.000**	0.004	*2.1*
fl417	11861	2.433	0.458	0.264	0.342	0.048	**0.000**	0.045	*39*
gil262	2378	1.003	0.158	0.340	0.172	**0.000**	**0.000**	0.012	*8.8*
gr96	55209	0.574	0.001	**0.000**	**0.000**	**0.000**	n/a	**0.000**	*1.9*
gr120	6942	1.443	0.181	0.089	0.110	**0.000**	n/a	0.023	*2.4*
gr137	69853	0.768	0.038	**0.000**	0.123	**0.000**	n/a	0.011	*3.0*
gr229	134602	0.751	0.148	0.096	0.132	0.006	n/a	**0.002**	*8.5*
gr431	171414	2.959	0.294	0.576	0.247	**0.043**	n/a	0.056	*42*
kroa100	21282	0.070	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*2.0*
krob100	22141	0.362	0.097	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*2.0*
kroc100	20749	0.535	0.169	0.012	0.073	**0.000**	**0.000**	**0.000**	*2.0*
krod100	21294	1.288	0.207	0.013	0.280	**0.000**	**0.000**	**0.000**	*2.0*
kroe100	22068	0.853	0.301	0.024	0.135	0.002	**0.000**	**0.000**	*2.0*
lin105	14379	0.526	0.083	0.032	0.019	**0.000**	**0.000**	**0.000**	*2.1*
lin318	42029	1.198	0.285	0.620	0.216	0.071	n/a	**0.036**	*18*
pcb442	50778	5.001	1.043	0.756	1.799	0.001	**0.000**	0.128	*46*
pr107	44303	0.087	0.071	0.005	0.035	**0.000**	**0.000**	0.006	*2.2*
pr124	59030	0.842	0.217	0.004	0.121	**0.000**	**0.000**	**0.000**	*2.4*
pr264	49135	0.771	0.145	0.053	0.125	**0.000**	**0.000**	0.011	*9.0*
pr439	107217	4.300	0.862	0.443	0.254	0.001	**0.000**	0.074	*44*
rat99	1211	0.511	0.090	0.056	0.004	**0.000**	**0.000**	**0.000**	*2.0*
rat783	8806	7.180	1.192	2.682	0.877	**0.000**	**0.000**	0.239	*160*
rd100	7910	0.846	0.255	0.407	0.303	**0.000**	**0.000**	0.001	*2.0*
rd400	15281	1.522	0.422	0.839	0.248	**0.000**	**0.000**	0.044	*34*
si535	48450	6.353	1.813	1.959	0.343	**0.005**	n/a	0.139	*69*
tsp225	3916	0.489	0.282	0.256	0.147	**0.000**	**0.000**	**0.000**	*8.3*
	Average:	1.989	0.480	0.511	0.305	0.007	**0.000**	0.044	

PIF_2, PIF_3 are given, where $PIF_1 = \dfrac{t_{0.5}(\text{LS})}{t_{0.5}(\text{IEFLS})}$, $PIF_2 = \dfrac{t_{0.5}(\text{FLS})}{t_{0.5}(\text{IEFLS})}$, $PIF_3 = \dfrac{t_{0.5}(\text{IFLS})}{t_{0.5}(\text{IEFLS})}$ (here, the target values are 0.5% and 1.0% above the corresponding optimal solution values for $n < 300$ and $n > 300$, respectively). From Table 3, it could be seen that the performance improvement factors of IEFLS to LS, IEFLS to FLS, and IEFLS to IFLS, averaged over 30 instances, are roughly equal to 6.6, 4.9, and 2.7,

respectively. For example, for the instance gr96, we obtained the following factors:

$$PIF_1 = \frac{t_{0.5}(\text{LS})}{t_{0.5}(\text{IEFLS})} \approx \frac{6.3}{0.7} = 9,$$

$$PIF_2 = \frac{t_{0.5}(\text{FLS})}{t_{0.5}(\text{IEFLS})} \approx \frac{5.3}{0.7} \approx 7.6,$$

$$PIF_3 = \frac{t_{0.5}(\text{IFLS})}{t_{0.5}(\text{IEFLS})} \approx \frac{3}{0.7} \approx 4.3$$

(this is seen also in Figure 14).

Table 3. Performance improvement factors for the local search heuristics on TSPLIB instances

Instance name	PIF_1	PIF_2	PIF_3	Instance name	PIF_1	PIF_2	PIF_3
bier127	6.5	4.1	1.9	**krod100**	7.2	4.6	3.3
ch130	8.2	6.3	2.9	**kroe100**	6.3	4.4	3.0
d657	5.0	3.7	1.7	**lin105**	9.4	6.9	3.1
dsj1000a	3.9	2.8	1.5	**lin318**	6.1	4.6	2.4
eil101	8.6	5.9	4.8	**pcb442**	5.1	3.8	1.7
fl417	4.5	3.8	2.1	**pr107**	7.7	5.8	3.2
gil262	4.8	3.9	2.6	**pr124**	7.5	5.9	2.8
gr96	9.0	7.6	4.3	**pr264**	5.8	4.9	2.5
gr120	8.4	6.1	5.0	**pr439**	5.5	4.3	2.0
gr137	7.8	3.9	2.2	**rat99**	6.3	5.9	2.9
gr229	5.6	4.5	2.5	**rat783**	5.2	3.9	1.8
gr431	4.9	3.8	1.9	**rd100**	9.5	7.1	3.5
kroa100	6.9	5.0	2.7	**rd400**	5.6	4.5	2.2
krob100	7.3	4.9	2.9	**si535**	5.5	4.3	2.1
kroc100	7.1	4.5	3.2	**tsp225**	6.3	5.8	2.9
Average:	----------				6.6	4.9	2.7

5. Concluding remarks

In this paper, we have proposed several modified local search heuristic algorithms for solving the well-known combinatorial optimization problem, the traveling salesman problem. In particular, we were concerned with the extended neighborhood structures and some new rules for the move (edge exchange) generation. We have also examined the performance of these extensions being integrated in the iterated local search paradigm, which is based on the combination of deterministic and stochastic search processes.

Our new heuristics are computationally tested on the benchmark TSP instances taken from the publicly available library of the TSP instances (TSPLIB). The results from the experiments show that applying the new developed neighborhood structures as well as the extended (modified) move generation mechanisms appears to be apparently superior to the traditional 2-opt-based heuristics from both solution quality and run time performance point of view. It has also been learned that, by integrating these modifications into the iterated local search framework, the performance is increased even more significantly. One more interesting observation is that the "breadth-first" like search appears to be preferable to the "depth-first" strategy at least for the TSP instances examined.

The results of the comparison of our new modified local search algorithm and other heuristic algorithms demonstrate that our approach is inferior to the best state-of-the-art heuristics for larger-sized symmetric TSP instances. However, our algorithm compares favourably with the elaborated modern heuristic procedures for smaller-sized problems.

Regarding the future work, it is worth to increase the performance of the proposed LS modifications using advanced data structures. It might also be worthy to try our LS heuristics on even larger-sized TSPLIB instances, or possibly other types of the TSP (like the asymmetric TSP). The introduced strategies for extending the neighborhoods and integrating the descending local search and stochastic perturbations

seem to be of rather general character, so they may be applicable for a wider class of combinatorial optimization problems. In addition, our heuristics could be used as effective sub-procedures within other artificial intelligence methods (like genetic/evolutionary algorithms or swarm optimization techniques).

References

[1] **R. M. Aiex, M. G. C. Resende, C. C. Ribeiro.** Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 2002, Vol. 8, 343–373.

[2] **D. L. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook.** *The traveling salesman problem: A computational study*. Princeton University Press: Princeton, 2007.

[3] **D. L. Applegate, W. J. Cook, A. Rohe.** Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 2003, Vol. 15, 82–92.

[4] **G. Babin, S. Deneault, G. Laporte.** Improvements to the Or-opt heuristic for the symmetric travelling salesman problem. *Journal of the Operational Research Society*, 2007, Vol. 58, 402–407.

[5] **D. Gamboa, C. Rego, F. Glover.** Implementation analysis of efficient heuristic algorithms for the traveling salesman problem. *Computers & Operations Research*, 2006, Vol. 33, 1154–1172.

[6] **J. Gu, X. Huang.** Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP). *IEEE Transactions on Systems, Man, and Cybernetics*, 1994, Vol. 24, 728–735.

[7] **G. Gutin, A. P. Punnen (eds.).** *The Traveling Salesman Problem and Its Variations*. Kluwer: Dordrecht, 2002.

[8] **K. Helsgaun.** An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 2000, Vol. 126, 106–130.

[9] **D. S. Johnson.** Local optimization and the traveling salesman problem. In M.Paterson (ed.). In: *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Proceedings. Lecture Notes in Computer Science*, Vol. 443, Springer, Heidelberg, 1990, pp. 446–461.

[10] **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, pp. 215–310.

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

[12] **G. Laporte.** A concise guide to the traveling salesman problem. *Journal of the Operational Research Society*, 2010, Vol. 61, 35–40.

[13] **J. K. Lenstra, A. H. G. Rinnooy Kan.** Some simple applications of the travelling salesman problem. *Operational Research Quarterly*, 1975, Vol. 26, 717–733.

[14] **S. Lin.** Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 1965, Vol. 44, 2245–2269.

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

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

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

[18] **Y. Marinakis, A. Migdalas, P. M. Pardalos.** Multiple phase neighborhood Search–GRASP based on Lagrangean relaxation, random backtracking Lin-Kernighan and path relinking for the TSP. *Journal of Combinatorial Optimization*, 2009, Vol. 17, 134–156.

[19] **P. Merz, J. Huhse.** An iterated local search approach for finding provably good solutions for very large TSP instances. In G.Rudolph et al. (eds.). In: *Parallel Problem Solving from Nature — PPSN X, 10th International Conference Dortmund, Germany, Proceedings. Lecture Notes in Computer Science*, Vol. 5199, Springer, Berlin-Heidelberg, 2008, pp. 929–939.

[20] **A. Misevicius.** Simulated annealing algorithm for the solution of the traveling salesman problem. In: A. Targamadzė, R. Butleris, R. Butkienė (eds.), *Proceedings of the 14th International Conference on Information and Software Technologies, IT-2008*, Technologija, Kaunas, 2008, pp. 19–24.

[21] **A. Misevicius, A. Blazinskas, J. Blonskis, V. Buksnaitis**. Genetic algorithms for the traveling salesman problem: negative and positive aspects (in Lithuanian). *Informacijos mokslai (Information Sciences)*, 2009, Vol. 50, 173–180.

[22] **A. Misevicius, A. Ostreika, A. Simaitis, V. Zilevicius.** Improving local search for the traveling salesman problem. *Information Technology and Control*, 2007, Vol. 36, 187–195.

[23] **A. Misevicius, J. Smolinskas, A. Tomkevicius.** Iterated tabu search for the traveling salesman problem: new results. *Information Technology and Control*, 2005, Vol. 34, 327–337.

[24] **H. D. Nguyen, I. Yoshihara, K. Yamamori, M. Yasunaga.** A new three-level tree data structure for representing TSP tours in the Lin-Kernighan heuristic. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2007, Vol. E90-A, 2187–2193.

[25] **C. Rego, D. Gamboa, F. Glover, C. Osterman.** Traveling salesman problem heuristics: leading methods, implementations and latest advances. *European Journal of Operational Research*, 2011, Vol. 211, 427–441.

[26] **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.]

[27] **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.

[28] **H.-K. Tsai, J.-M. Yang, C.-Y. Kao.** Solving traveling salesman problems by combining global and local search mechanisms. In D.B.Fogel (ed.), *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, IEEE Press, New York, 2002, pp. 1290−1295.

[29] **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.

Received February 2012.

Appendix

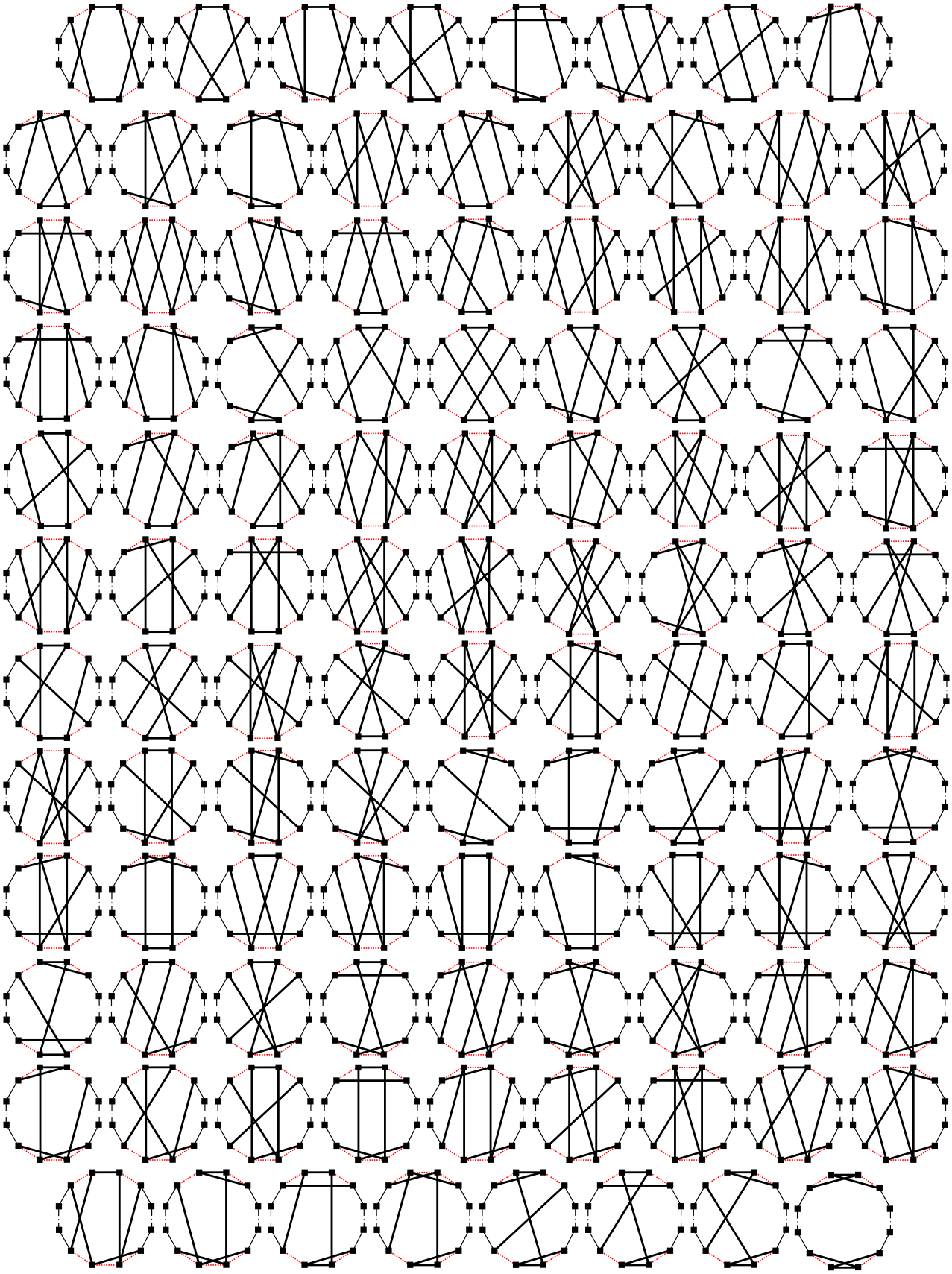

Figure A1. Illustrative examples of the different possible variants of 6-edge-exchange moves