EXPERIMENTS WITH LOCAL SEARCH HEURISTICS FOR THE TRAVELING SALESMAN PROBLEM

Alfonsas Misevicius\textsuperscript{1}, Andrius Blazinskas\textsuperscript{2}

\textsuperscript{1}Kaunas University of Technology, Department of Multimedia Engineering, Studentu st. 50–400a/410a, LT–5136, Kaunas, Lithuania, alfonsas.misevicius@ktu.lt

\textsuperscript{2}Kaunas University of Technology, Department of Multimedia Engineering, Studentu st. 67–504, LT–51392, Kaunas, Lithuania, andrius.blazinskas@ktu.lt

Abstract. In this paper, the experiments with local search heuristic algorithms for the traveling salesman problem (TSP) are described. Since the ordinary local search heuristics very seldom yield solutions of high quality, we investigate the enhanced local search algorithms using the extended neighbourhood structures. We also examine the performance of the local search heuristics in an iterated local search paradigm based on the deterministic descent-random ascent methodology. Our new heuristic algorithms are tested on the symmetric TSP instances taken from the publicly available electronic library of the TSP instances (TSPLIB). The results from the experiments demonstrate that our heuristics appear to be superior to traditional types of local search algorithms.

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

1 Introduction

The traveling salesman problem (TSP) \cite{2,7} can be formulated as follows. Given the matrix $D = (d_{ij})_{n \times n}$ and the set $\Pi_n$ of all possible permutations of the integers from 1 to $n$, find a permutation $p = (p(1), p(2), \ldots, p(n)) \in \Pi_n$ that minimizes the following function:

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

Each permutation corresponds to a tour through $n$ cities such that each city is visited exactly once. A particular element of the permutation, $j_i = p(i)$ ($i = 1, 2, \ldots, n$), denotes city $j_i$ to visit at step $i$. The pairs $(p(1), p(2)), \ldots, (p(i), p(i+1)), \ldots, (p(n), p(1))$ are called edges. The matrix $D$ contains distances between all the pairs of cities\textsuperscript{1}; the entries $d_{p(i)p(i \mod n + 1)}$ ($i = 1, 2, \ldots, n$) then denote the corresponding edge length and $z(p)$ is the total length of the tour $p$. So, solving the TSP means searching for the shortest tour so that every city is visited once and only once and the salesman returns back to his starting city at the end of the trip.

The traveling salesman problem is one of the most famous and probably the most intensively studied problems in combinatorial optimization. The TSP and its variants have various important practical applications \cite{13}. At the same time, the TSP is known to be NP-hard and is an excellent experimenting platform for the investigation of both exact (approximation) algorithms and heuristics (metaheuristics) \cite{2,7,10,11,12,19}.

In this paper, we describe the experiments with several local search (LS) heuristics for the TSP. The origins of the LS algorithms go back to \cite{5,15,16}; still, this class of heuristics is a quite active domain of research \cite{3,4,6,8,9,14,18,21}. The LS heuristics are distinguished for their simplicity and ease of implementation; but, sometimes, they also hide some features which can significantly improve the efficiency of the search. In this work, we demonstrate how such improvement is achieved by extending the traditional local search paradigm.

The remaining part of this paper is organized as follows. In Section 1, we outline the general local search framework. The proposed extensions of the basic LS framework are also described. In Section 2, we present the results of the computational experiments with the proposed heuristics on the benchmark problems taken from the TSP library (TSPLIB). Section 3 completes the paper with concluding remarks.

2 Local search heuristics for the traveling salesman problem

2.1 Preliminaries

The Hamming distance between two tours (permutations) $p$ and $p'$ is declared as

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

where $\Omega$ is the set that consists of all possible pairs $(p(i), p(i \mod n + 1))$ ($i = 1, 2, \ldots, n$) such that $\neg \exists j:$

\textsuperscript{1} Only symmetric TSP is considered here, i.e. $d_{ij} = d_{ji}, i = 1, 2, \ldots, n.$
\[(p(i), p(i \mod n + 1)) = \begin{cases} (p'(j), p'(j + 1)), & 1 \leq j < n \\ (p'(j), p'(1)), & j = n \end{cases} \quad \text{or} \quad \begin{cases} (p'(j), p'(j - 1)), & 1 < j \leq n \\ (p'(j), p'(n)), & j = 1 \end{cases}. \]  

(3)

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

A neighbourhood function \( \Theta: \Pi_n \rightarrow \Pi_n \) assigns for every \( p \) from \( \Pi_n \) a set \( \Theta(p) \subseteq \Pi_n \) — the set of neighbouring solutions of \( p \) (see Figure 1). An example of the neighbourhood function is the \( \tau \)-edge-exchange neighbourhood \( \Theta_\tau \) (\( 2 \leq \tau \leq n \)), which is defined in the following way:

\[
\Theta_\tau(p) = \{ p' \mid p' \in \Pi_n, \ p(p, p') = \tau \}.
\]

A special case of the neighbourhood \( \Theta_\tau \) is the neighbourhood \( \Theta_2 \) (2-edge-exchange neighbourhood), which is frequently used for the TSP.

Figure 1. Graphical interpretation of a hypothetical neighbourhood

The tour \( p' \in \Theta(p) \) can be obtained from \( p \) by an operation called a move, and \( p \) is said to move to \( p' \) when such an operation is performed. In case of the 2-edge-exchange neighbourhood, the move may be formally described as an operator \( \phi(p, i, j): \Pi \times \mathbb{N} \times \mathbb{N} \rightarrow \Pi \), which gives \( p' \in \Theta(p) \) such that \( p'(i) = p(i), \ p'(i + 1) = p(j), \ p'(j) = p(i + 1), \ p'(j \mod n + 1) = p(j \mod n + 1) \), where \( 1 \leq i, j \leq n \land 1 < j - i < n - 1 \); in addition, if \( j - i - 2 \geq 1 \), then \( p'(i + k + 1) = p(j - k) \) for every \( k \in \{1, ..., j - i - 2\} \). Roughly speaking, two edges at the positions \( i \) and \( j \) are removed and two different edges are added (see Figure 2).

For the 2-edge-exchange move, a shorter notation of the form \( \phi_{ij} \) may be used, such that \( p' = p \otimes \phi_{ij} \) means that \( p' \) is obtained from \( p \) by applying the move \( \phi(p, i, j) \).

Figure 2. Graphical illustration of the 2-edge-exchange move

The solution (tour) \( p^* \) is 2-optimal (imal) solution, i.e. it is locally optimal with respect to the neighbourhood \( \Theta_2 \) if \( z(p^*) \leq z(p) \) for any \( p \in \Theta_2(p^*) \), i.e. \( z(p^*) \leq z(p^* \otimes \phi_{ij}) \) for every \( i \in \{1, ..., n - 2\}, \ j \in \{i + 2, ..., n - 1 + \text{sign}(i - 1)\} \).

Figure 3. Graphical interpretation of the search process

Figure 4. Illustration of transformations of TSP tours during the local search process
The 2-opt solution may be achieved by the 2-opt procedure, which can also be viewed as a steepest descent (SD) algorithm using the neighbourhood $\Theta_2$, i.e. a sequence of the descending (improving) 2-edge-exchange moves (2-opt moves) starting from an arbitrary initial solution (see Figures 3, 4). (The initial solution may be generated randomly or constructed in a pre-determined way.) The description of the template of the 2-opt algorithm (steepest descent in $\Theta_2$) in a high-level pseudo-code form is given in Figure 5. In a similar way, higher order procedures may be described: 3-opt, 4-opt, ..., r-opt. However, in this case, the run-time complexity increases exponentially.

In the next section, we propose several extended local search strategies without substantially increasing computational complexity.

\begin{verbatim}
function steepest_descent_in_Theta2(p);
  // input: p – initial (starting) solution
  // output: $p^*$ – resulting solution (locally optimal solution in $\Theta_2$)

begin
  $p^* := p$;
  repeat
    $p := p'$;  // $\Delta_{\text{min}} := 0$; // $\Delta_{\text{min}}$ denotes the minimum difference in the objective function values
    for $i := 1$ to $n - 2$ do
      for $j := (i + 2)$ to $n - 1 + \text{sign}(i - 1)$ do begin
        $\Delta := z(p \otimes \phi_i) - z(p)$;  // $\Delta < \Delta_{\text{min}}$ then $\Delta_{\text{min}} := \Delta$; $k := i$; $l := j$
      end;  // for
    end;  // for

    if $\Delta_{\text{min}} < 0$ then $p^* := p \otimes \phi_{k}$  // move from the current solution to a new one
    until $\Delta_{\text{min}} = 0$;
  return $p^*$;
end.
\end{verbatim}

Figure 5. Pseudo-code of the steepest descent algorithm using the neighbourhood $\Theta_2$

### 2.2 Extensions of the local search

We can construct new more complex neighbourhood structures by allowing a composition of simpler neighbourhoods (like $\Theta_2$). This is like having some compounded neighbourhoods consisting of several (parallel) neighbourhoods (sub-neighbourhoods). The extended (compounded) 2-edge-exchange neighbourhoods may be formally defined as follows. Let $p$ be a feasible tour (permutation) from $\Pi_n$; also, let $\Theta_j(p)$ be the 2-edge-exchange neighbourhood as defined above in Section 2.1. Then, the extended neighbourhood $\Theta_k$ (denoted as $\Theta_{2k}$) can be described in the following way (also see Figure 6):

$$\Theta_{2k}(p) = \Theta_j(p) \cup \{p^* \mid p^* \in \Pi_n, p^* \neq p, \rho(p^*, p^*) = 2\};$$

where $p^* = \arg \min_{p^* \in \Theta_k(p)} z(p^*)$, $\Theta_{2k}(p) = \Theta_j(p) \setminus \{\arg \min_{p^* \in \Theta_k(p)} z(p^*)\}$.

Figure 6. Graphical interpretation of the neighbourhood $\Theta_{2k}$

Further, two new additional neighbourhoods ($\Theta_{20202}, \Theta_{20202}$) may be introduced (also see Figures 7, 8):

$$\Theta_{20202}(p) = \Theta_{202}(p) \cup \{p^\wedge \mid p^\wedge \in \Pi_n, p^\wedge \neq p, \rho(p^\wedge, 2p^\wedge) = 2\};$$

where $p^\wedge = \arg \min_{p^\wedge \in \Theta_{202}(p^*)} z(p^\wedge)$, $\Theta_{202}(p^*) = \Theta_{202}(p^*) \setminus \{\arg \min_{p^\wedge \in \Theta_{202}(p^*)} z(p^\wedge)\}$.

$$\Theta_{20202}(p) = \Theta_{202}(p) \cup \{p^\wedge \mid p^\wedge \in \Pi_n, p^\wedge \neq p, \rho(p^\wedge, 2p^\wedge) = 2\};$$
Similarly, the algorithms using the neighbourhood algorithms is proportional to \( O(n^2) \), we used a special type of perturbations — the so-called nearest neighbour reconnection (NNR) strategy. In this work, we applied the steepest descent procedure using the initial (starting) solution \( p \in \Theta_{2020} \) to the role of the improving (descending) algorithm; while, for the random perturbations (random ascents), we used a special type of perturbations — the so-called nearest neighbour reconnection (NNR) procedure (see [18]).

The pseudo-code of the steepest descent algorithm in the neighbourhood \( \Theta_{2020} \) is presented in Figure 9. Similarly, the algorithms using the neighbourhoods \( \Theta_{2020} \), \( \Theta_{2022} \) could be described. The complexity of these algorithms is proportional to \( O(n^2) \), where \( n \) is the problem size. These algorithms may be thought of as "forward-looking" strategies; they also conceptually resemble the "depth-first search" and "breadth-first search" strategies used for searching in tree or graph structures.

![Figure 7. Graphical interpretation of the neighbourhood \( \Theta_{2020} \)](image)

![Figure 8. Graphical interpretation of the neighbourhood \( \Theta_{2022} \)](image)

The pseudo-code of the steepest descent algorithm using the neighbourhood \( \Theta_{2020} \) is presented below.

```
function steepest_descent_in_\( \Theta_{2020}(p) \);
// input: \( p \) — initial (starting) solution
// output: \( p^* \) — resulting solution (locally optimal solution in \( \Theta_{2020} \))
begin
\( p^* := p \);
repeat
\( p := p^* \);
\( \Delta := \infty ; \Delta_{\text{min}} := \infty ; k := 0 ; f := 0 ; \)
for \( i := 1 \) to \( n-2 \) do
for \( j := i+2 \) to \( n-1 + \text{sign}(i-1) \) do
begin
\( \Delta := \Delta - \| \phi \| \); \( \Delta_{\text{min}} := \min \{\Delta, \Delta_{\text{min}}\} \); \( k_{\text{min}} := \min \{k, k_{\text{min}}\} \); \( f := f + 1 \);
end; // for
end; // for
\( p^* := \min \{\Delta, \Delta_{\text{min}}, \Delta_{\text{max}} \} \); \( k_{\text{min}} := \min \{k, k_{\text{min}}\} \); \( f := f + 1 \);
if \( z(p^*) < z(p) \) then \( p^* := p^* \) // move from the current solution to a new one
until \( z(p^*) \geq z(p) \);
return \( p^* \);
end.
```

![Figure 9. Pseudo-code of the steepest descent algorithm using the neighbourhood \( \Theta_{2020} \)](image)

The other way of extending the classical local search framework is to integrate (combine) the deterministic and stochastic search processes. This policy is known as an iterated local search (ILS) [17]. There are many different variations of the ILS approach depending on the actual deterministic improving algorithm and randomization (perturbation) technique. In this work, we applied the steepest descent procedure using the neighbourhood \( \Theta_{2020} \) in the role of the improving (descending) algorithm; while, for the random perturbations (random ascents), we used a special type of perturbations — the so-called nearest neighbour reconnection (NNR) procedure (see [18]).

\[ \Theta_{2022}(p) = \Theta_{2022}(p) . \]
The resulting algorithm (called the iterated steepest descent-random ascent (ISD-RA)) is not deterministic any more; however, it enables to significantly increase the search efficacy and leads to the superior-quality solutions (see Section 3). The NNR perturbations take $O(n^2)$ time, so the complexity of ISD-RA remains proportional to $O(n^2)$. The high-level pseudo-code of this algorithm is shown in Figure 10. The limit of run time of ISD-RA is predetermined by the number of iterations, $\lambda$, which, in turn, can be flexibly tuned by the user.

```
function iterated_steepest_descent_random_ascent(p, \lambda)
    // input: p - initial (starting) solution, \lambda - number of iterations (\lambda \geq 1)
    // output: p' - resulting solution
begin
    p' := steepest_descent(p); // perform steepest descent in a given neighbourhood (starting from p')
    p'' := p';
    current_number_of_iterations := 0;
repeat
    current_number_of_iterations := current_number_of_iterations + 1;
    p' := random_ascend(p''); // perform random ascending perturbation (starting from p''
    p' := steepest_descent(p'); // perform steepest descent in a given neighbourhood (starting from p')
    if z(p') < z(p'') then p'' := p
until current_number_of_iterations = \lambda;
return p';
end.
```

Figure 10. Pseudo-code of the iterated steepest descent-random ascent algorithm

3 Results of computational experiments

We have tested our local search algorithms on the benchmark problems taken from the publicly available electronic library of the TSP – TSPLIB [20]. The experiments were carried out on a personal computer with an Intel Pentium IV 3 GHz single-core processor.

The following heuristics were used in the experiments: 2-opt; 3-opt; 4-opt; steepest descent in $\Theta_2$ (SD-$\Theta_2$); steepest descent in $\Theta_{202}$ (SD-$\Theta_{202}$); steepest descent in $\Theta_{2020}$ (SD-$\Theta_{2020}$); iterated steepest descent-random ascent (ISD-RA). In order to allow fair comparison, the experimentation was designed as follows. Let $W$ be the pre-defined number of runs. At every run, the algorithm is applied to a given instance, each time starting from a new random initial solution. The current run is interrupted as soon as the local optimum is found (or the maximum number of iterations is performed). The next run is then started, and so on. The process stops when $W$ runs have been carried out. The best solution obtained during these runs serves as a resulting solution of the algorithm. This is repeated for each examined instance. In particular, we used 1 run of 4-opt, 100 runs of 3-opt, and 10000 runs of 2-opt. Similarly, we applied 5000 runs of SD-$\Theta_2$, and 2500 runs of SD-$\Theta_{202}$ and SD-$\Theta_{2020}$. In the last three cases, the number of runs is smaller than in 2-opt; this is due to the fact that SD-$\Theta_2$, SD-$\Theta_{202}$, and SD-$\Theta_{2020}$ consume more computation (CPU) time. The run time of ISD-RA is controlled by the number of iterations, $\lambda$. We used $\lambda = 2000$. In this way, all the algorithms utilize approximately similar CPU times (note that we experimented with the problems, where the number of cities is more or less equal to 100).

The results of these experiments are presented in Table 1 (additionally, in Figure 11, we graphically illustrate the obtained optimal tours for several TSP instances). Note that, in Table 1, $\delta$ denotes the relative deviation of the solutions from the provably optimal solution; it is defined by the following formula: $\delta = 100(\bar{z} - z^\circ)/z^\circ\ [\%]$, where $\bar{z}$ is the obtained value of the objective function (tour length) and $z^\circ$ denotes the provably optimal objective function value (these values can be found in TSPLIB [20]).

The results from Table 1 show that the new proposed neighbourhoods $\Theta_{202}$, $\Theta_{2020}$, $\Theta_{2020}$ are superior to the ordinary 2-edge-exchange neighbourhood $\Theta_2$. Also, it could be seen that the neighbourhood $\Theta_{2020}$ ("breadth-first search") is preferable to the neighbourhood $\Theta_{2020}$ ("depth-first search") (this is despite the fact that the sizes of the neighbourhoods $\Theta_{2020}$ and $\Theta_{2020}$ are identical). Finally, it is easy to observe that the steepest descent in the neighbourhood $\Theta_{2020}$ combined with the random ascending perturbations is clearly better than all remaining algorithms without random perturbations. It should be stressed that the results of ISD-RA may possibly be improved even more by incorporating more elaborated perturbation procedures.

The other criterion of the efficiency of the algorithms is so-called time-to-target plots [1]. In this case, for any given target value of the objective function (target solution) and the time to obtain this value, the time-to-target plot shows the probability that the target value will be obtained. So, for a given target value, the run time of the algorithm to achieve this value is recorded. This is repeated multiple times and the recorded times are then
sorted. With the $i$th time, a probability $P_i = \frac{i \cdot 0.5}{m}$ ($i = 1, 2, \ldots$) is associated, where $m$ is the number of trials (we used $m = 30$).

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

<table>
<thead>
<tr>
<th>Instance name</th>
<th>$z^0$</th>
<th>2-opt</th>
<th>3-opt</th>
<th>4-opt</th>
<th>$SD -_2 2_2 \Theta$</th>
<th>$SD -_2 2_2 \Theta$</th>
<th>$SD -_2 2_2 \Theta$</th>
<th>ISD-RA</th>
<th>Average CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bier127</td>
<td>118282</td>
<td>0.649</td>
<td>0.853</td>
<td>1.598</td>
<td>0.517</td>
<td>0.340</td>
<td>0.331</td>
<td>0.000</td>
<td>3.7</td>
</tr>
<tr>
<td>ch130</td>
<td>6110</td>
<td>0.898</td>
<td>0.977</td>
<td>0.953</td>
<td>0.418</td>
<td>0.401</td>
<td>0.356</td>
<td>0.016</td>
<td>3.9</td>
</tr>
<tr>
<td>ell101</td>
<td>629</td>
<td>1.455</td>
<td>1.934</td>
<td>2.657</td>
<td>1.119</td>
<td>0.996</td>
<td>0.855</td>
<td>0.007</td>
<td>3.1</td>
</tr>
<tr>
<td>gr96</td>
<td>55209</td>
<td>0.446</td>
<td>0.397</td>
<td>0.406</td>
<td>0.321</td>
<td>0.311</td>
<td>0.302</td>
<td>0.000</td>
<td>2.9</td>
</tr>
<tr>
<td>gr120</td>
<td>6942</td>
<td>1.786</td>
<td>1.468</td>
<td>2.579</td>
<td>1.199</td>
<td>0.939</td>
<td>0.843</td>
<td>0.034</td>
<td>3.6</td>
</tr>
<tr>
<td>gr137</td>
<td>69853</td>
<td>0.819</td>
<td>0.888</td>
<td>1.718</td>
<td>0.768</td>
<td>0.779</td>
<td>0.600</td>
<td>0.011</td>
<td>4.5</td>
</tr>
<tr>
<td>kroa100</td>
<td>21282</td>
<td>0.043</td>
<td>0.045</td>
<td>0.108</td>
<td>0.040</td>
<td>0.035</td>
<td>0.035</td>
<td>0.000</td>
<td>3.0</td>
</tr>
<tr>
<td>krob100</td>
<td>22141</td>
<td>0.377</td>
<td>0.575</td>
<td>2.527</td>
<td>0.387</td>
<td>0.352</td>
<td>0.347</td>
<td>0.000</td>
<td>3.0</td>
</tr>
<tr>
<td>kroc100</td>
<td>20749</td>
<td>0.536</td>
<td>0.539</td>
<td>0.535</td>
<td>0.403</td>
<td>0.375</td>
<td>0.335</td>
<td>0.000</td>
<td>3.0</td>
</tr>
<tr>
<td>krod100</td>
<td>21294</td>
<td>1.483</td>
<td>1.548</td>
<td>1.619</td>
<td>1.002</td>
<td>0.940</td>
<td>0.904</td>
<td>0.000</td>
<td>3.0</td>
</tr>
<tr>
<td>kroe100</td>
<td>22068</td>
<td>0.738</td>
<td>1.392</td>
<td>2.035</td>
<td>0.830</td>
<td>0.694</td>
<td>0.657</td>
<td>0.000</td>
<td>3.0</td>
</tr>
<tr>
<td>lin105</td>
<td>14379</td>
<td>0.441</td>
<td>0.437</td>
<td>2.324</td>
<td>0.438</td>
<td>0.416</td>
<td>0.403</td>
<td>0.000</td>
<td>3.2</td>
</tr>
<tr>
<td>pr107</td>
<td>44303</td>
<td>0.088</td>
<td>0.183</td>
<td>1.429</td>
<td>0.095</td>
<td>0.086</td>
<td>0.071</td>
<td>0.010</td>
<td>3.2</td>
</tr>
<tr>
<td>pr124</td>
<td>59030</td>
<td>0.733</td>
<td>0.223</td>
<td>0.025</td>
<td>0.450</td>
<td>0.412</td>
<td>0.342</td>
<td>0.000</td>
<td>3.6</td>
</tr>
<tr>
<td>rat99</td>
<td>1211</td>
<td>0.424</td>
<td>0.401</td>
<td>1.007</td>
<td>0.365</td>
<td>0.329</td>
<td>0.296</td>
<td>0.000</td>
<td>3.0</td>
</tr>
<tr>
<td>rd100</td>
<td>7910</td>
<td>0.921</td>
<td>0.682</td>
<td>0.445</td>
<td>0.537</td>
<td>0.462</td>
<td>0.391</td>
<td>0.009</td>
<td>3.0</td>
</tr>
<tr>
<td>Average</td>
<td>0.740</td>
<td>0.784</td>
<td>1.373</td>
<td>0.556</td>
<td>0.492</td>
<td>0.442</td>
<td>0.005</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

In Figure 11, we present the time-to-target plots for two best algorithms, $SD -_2 2_2 \Theta$ and ISD-RA. The instances examined are kroa100 and krob100 and the target values are 21282 and 22141, respectively. These values are exactly equal to the provably optimal objective function values (see TSPLIB [20]).

The performance improvement factor, $PIF$, of ISD-RA to $SD -_2 2_2 \Theta$ can formally be defined by the formula: $PIF = \frac{t_{0.5}(SD -_2 2_2 \Theta)}{t_{0.5}(ISD - RA)}$, where $t_{0.5}$ denotes the time needed to obtain the given target solution with probability 0.5. From Figure 12, it could be viewed that the performance improvement factor of ISD-RA to $SD -_2 2_2 \Theta$ is equal to approximately 15.5 ($\frac{3.0}{0.2}$) and 11.1 ($\frac{5.5}{0.5}$) for the instances kroa100 and krob100.
respectively. We observed similar performance improvement also for remaining TSPLIB instances used in the experimentation.

4 Concluding remarks

The local search (LS) heuristics remain popular among researchers due to their reasonably good efficiency and easy implementation. In this work, several new modifications of the local search heuristics for solving the traveling salesman problem are proposed. These heuristics are based on the use of the extended neighbourhoods, which enable to seek high quality solutions without significantly increasing computational complexity. The efficiency of the search can be improved even more by integrating these extended heuristics into the iterated local search (ILS) framework. The computational experiments confirm the promising performance of the resulting ILS algorithm (iterated steepest descent-random ascent (ISD-RA)) from both the solution quality and computation time point of view.

Our proposed strategies for extending the neighbourhoods and enhancing the local search heuristics are of quite general character. They appear to be problem-independent, so may be applicable for other combinatorial type optimization problems. Also, there is still a potential for constructing further new neighbourhood structures following the proposed approach. In addition, the improved local search procedures can be used as powerful subroutines within more complex metaheuristic methods (like genetic or evolutionary algorithms). This might be one of the hopeful future research directions.

References