ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2007, Vol. 36, No. 2

IMPROVING LOCAL SEARCH FOR THE TRAVELING SALESMAN PROBLEM

Alfonsas Misevičius, Armantas Ostreika, Antanas Šimaitis, Vilius Žilevičius

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

Abstract. The subject of this paper is the improving of local search for the traveling salesman problem (TSP). In particular, a so-called fast descent-random ascent (FDRA) strategy is proposed. The FDRA approach is based on the fast-modified 2-opt algorithm combined with certain perturbation (random ascent) procedures. The results obtained from the experiments demonstrate that the new improved local search strategy is better than the other local search algorithms. This approach may also be applied to other combinatorial optimization problems.

Keywords: traveling salesman problem, heuristics, local search, fast descent-random ascent strategy.

Introduction

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

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

The interpretation of n, $\boldsymbol{D}$ and p is as follows:

– n denotes the number of cities;

– $\boldsymbol{D}$ is the matrix that contains distances between all the pairs of cities;

– permutations are typically called tours and the pairs $(p(1),p(2))$, ..., $(p(i),p(i+1))$, ..., $(p(n),p(1))$ are referred to as edges; a particular element of the permutation $j = p(i)$ denotes city j to visit at step i.

Thus, solving the TSP means searching for the shortest closed tour in which every city is visited exactly once.

The TSP is a well-known representative example of combinatorial optimization. It is NP-hard [3] and still remains a great challenge for the researchers in this field. The TSP also serves as a good experimental basis for the investigation of various optimizations techniques. Since there are no efficient (polynomial time) exact algorithms for this problem, heuristic methods (like tour construction heuristics [1,18], (descent) local search algorithms [4,8,9], simulated annealing [15], tabu search [2,12], ant colony optimization [19], evolutionary (genetic) algorithms [11], etc.) are often applied. More exhaustive surveys of the heuristic algorithms for the TSP can be found in [4, 6, 7, 14, 17, 20].

In this paper, a new improved heuristic approach for the TSP based on the modified descent local search (2-opt) algorithm is discussed. The remaining part of the paper is organized as follows. Firstly, some basic definitions and preliminaries are given. Then, the improved local search algorithm and its variants are considered in more details. The results of the computational experiments are presented as well. The paper is completed with the concluding remarks.

1. Basic definitions and preliminaries

We start with some definitions and preliminaries. The basic definitions related to the traveling salesman problem and its solutions are as follows.

Definition 1. Hamming distance between two permutations (tours) p and p' is declared as $\rho(p,p') = n - |\Omega|$, where Ω is the set that consists of all possible pairs $(p(i),p((i \bmod n) + 1))$ $(i \in \{1, 2, ..., n\})$ such that $\exists j$:

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

or

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

(Briefly speaking, the Hamming distance between two tours is the number of edges that are contained in the first tour but not in the second tour.)

Definition 2. A neighbourhood function Θ: $\Pi \to 2^{\Pi}$ assigns for every p from Π a set $\Theta(p) \subseteq \Pi$ – the set of neighbouring solutions of p. The τ-edge-exchange neighbourhood Θ_τ ($2 \le \tau \le n$) is defined in the following way:
$\Theta_\tau(p) = \{p' \mid p' \in \Pi,\ \rho(p,p') = \tau\}$, where $p \in \Pi$ and ρ is the Hamming distance. The exploration of the neighbourhood Θ_τ takes $O(n^\tau)$ time. A special case of the neighbourhood Θ_τ is the neighbourhood Θ_2 (2-edge-exchange neighbourhood).

Definition 3. The solution (tour) $p' \in \Theta_\tau(p)$ 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. Formally, the move may be described as an operator ϕ: $\Pi \to \Pi$. A particular move example is the 2-edge-exchange move $\phi(p,i,j)$: $\Pi \times \mathrm{N} \times \mathrm{N} \to \Pi$, which gives $p' \in \Theta_2(p)$ such that $p'(i) = p(i)$, $p'(i+1) = p(j)$, $p'(j) = p(i+1)$, $p'((j \bmod n) + 1) = p((j \bmod n) + 1)$, where $1 \le i, j \le n \wedge 1 < j - i < n - 1$; in addition, if $j - i - 2 \ge 1$, then $p'(i + k + 1) = p(j - k)$ for every $k \in \{1, ..., j - i - 2\}$ (i.e. the corresponding elements of the permutation are replaced in the reversed order) (see Figure 1a). In the other words, two edges at the positions i and j are removed and two different edges are added (see Figure 1b). For this move, we will also use a compact notation ϕ_{ij}; then, $p' = p \oplus \phi_{ij}$ means that p' is obtained from p by applying $\phi(p,i,j)$. Similarly, higher order moves may be defined: 3-edge-exchange move, 4-edge-exchange move, ..., τ-edge-exchange move, ... The computational complexity of the τ-edge-exchange move is $O(n)$.

Definition 4. The solution (tour) $p^\bullet$ is 2-opt(imal) solution, i.e. it is locally optimal with respect to the neighbourhood Θ_2 if $z(p^\bullet) \le z(p)$ for any $p \in \Theta_2(p^\bullet)$.

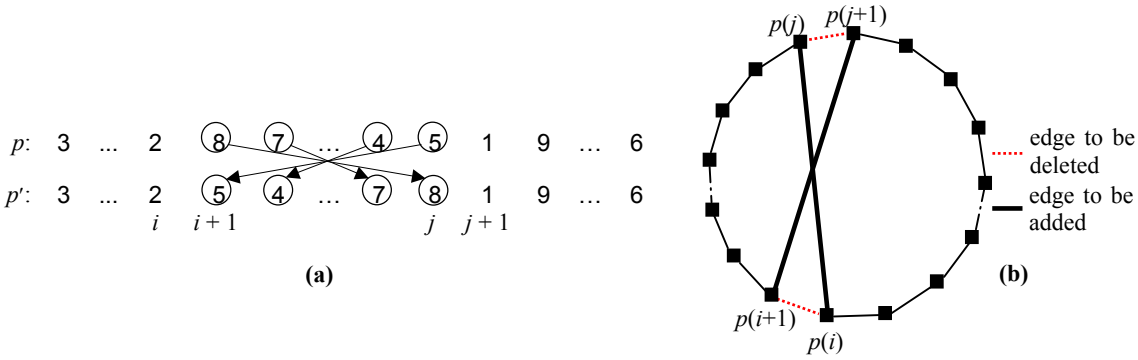

Figure 1. An example of the 2-edge-exchange move

The 2-opt solution may be achieved by the corresponding procedure (called 2-opt algorithm), which can be viewed as a sequence of the improving 2-edge-exchange (or simply 2-opt) moves. Usually, one starts from a randomly chosen initial solution. The initial solution may be constructed heuristically as well [1, 18]. The template of the 2-opt algorithm is presented in Figure 2. Similarly, τ-opt algorithm ($\tau > 2$) can be derived.

```
function 2-opt(p);
// input: p – initial (starting) solution; output: p• – resulting (locally optimal) solution
begin
  p• := p;
  repeat
    p := p•;
    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 ⊕ φij ) – z(p);
        if Δ < Δmin then begin Δmin := Δ; k := i; l := j end
      end;  // for
    if Δmin < 0 then p• := p ⊕ φkl   // move from the current solution to a new one
  until Δmin = 0;
  return p•
end.
```

Figure 2. Template of the 2-opt algorithm for the TSP

Before introducing the improved local search algorithm, let us describe the modified 2-opt procedure. In fact, there are two modifications. The first one is related to the restricted exploration of the neighborhood. So, instead of thorough scanning of the complete neighborhood, only the nearest neighbors of the current city are taken into consideration. The size of the neighbor list, i.e. the candidate list, *CL*, is controlled by the algorithm's user. This technique is not new [5,6]. It allows to speed up the neighborhood search process and reduce the computation time considerably without significant loss in the quality of the results. The neighbour list takes $O(mn)$ memory ($m = |\ CL\ |$) and its construction takes $O(n^2 \log_2 n)$ time (the construction takes place only once at the data processing phase).

The second modification aims also at minimizing of the run time of the algorithm. In this case, the run time reduction is due to limiting the number of the descending moves (descents). The limit of descents, λ, can be again flexibly tuned by the user. The resulting modification of the 2-opt procedure is called a "fast descent" (FD) (or "λ-descent"). It can be seen that "1-descent" is a very special case of "λ-descent". The template of the fast descent algorithm is shown in Figure 3. The complexity of the FD algorithm is $O(mn)$, where m is the fixed size of the candidate list *CL*.

```
function FastDescent(p,λ);
// input: p – initial (starting) solution, λ – number of descents (λ ≥ 1)
// output: p• – resulting (locally optimal) solution
// auxiliary variables: CL – candidate list (list of the nearest neighbours)
begin
  p• := p;
  number_of_moves := 0;
  for i := 1 to n do index[p[i]] := i;
  repeat
    p := p•;
    Δmin := 0;  // Δmin denotes the minimum difference in the objective function values
    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 ⊕ φij ) – z(p);
            if Δ < Δmin then begin Δmin := Δ; k := i; l := j end
        end // if
      end; // for
    if Δmin < 0 then begin
        number_of_moves := number_of_moves + 1;
        p• := p ⊕ φkl ;  // move from the current solution to a new one
        update index
    end // if
  until (number_of_moves = λ) or (Δmin = 0);
  return p•
end.
```

Figure 3. Template of the fast descent (λ-descent) algorithm for the TSP

2. An improved local search strategy: fast descent-random ascent

The underlying idea of our improved local search algorithm is to exploit good facets of both the deterministic search and stochastic search in an effective way. In particular, we combine the fast descent algorithm described above and special sort random perturbations. They are called as "random ascent".

Remind that, in the deterministic 2-opt algorithm, only the corresponding neighbors of the current solution are considered, and solely improving 2-edge-exchange moves are performed between these neighbors. We can extend the straightforward 2-opt local search if we tolerate some more moves time after time. This may be achieved by combination of the improving (descent) and random non-improving (ascent) moves in a proper manner. In particular, we can obtain the iterative process consisting of one or more tentative ascending moves (like random 3- or 4-edge-exchange moves) followed by the λ-descent procedure (i.e. λ improving 2-edge-exchange moves)[1].

[1] It is important that the structure of the given problem allows the effective (fast) implementation and execution of the random ascending moves. Fortunately, the TSP is just the case. For example, a random τ-edge-exchange move can be executed in time $O(n)$ with only a negligible effect to the overall complexity of the resulting algorithm.

So, let p be the current solution (tour). Then, if the random ascent (coupled with the λ-descent) results in a solution p'' that is better than the solution p' obtained by a single 2-opt move, the solution p'' replaces the current solution p and serves as a starting solution for the next iteration; otherwise the solution p is replaced by the solution $p' \in \Theta_2(p)$. Obviously, p'' does not necessarily belong to $\Theta_2(p)$. This procedure is continued until neither p' nor p'' is better than p. The above process is not the deterministic local search any more. We call it as a "fast descent-random ascent" (FDRA). The template of the basic version of the fast descent-random ascent algorithm is quite simple. It is given in Figure 4.

Our fast descent-random ascent approach is slightly different from the well-known iterated local search (ILS) method proposed by Lourenco, Martin, and Stützle [10]. Very generally, ILS may be thought of as a "high-level relay hybridization", where self-contained heuristics are executed in sequence (independently) (see also [21]). FDRA, on the contrary, belongs rather to a class of "low-level relay hybrids", in which one heuristic is embedded into other heuristic [21]. In our case, the random ascending moves are, in particular, embedded into the deterministic (2-opt) local search.

On the other hand, our approach appears to be quite similar to a so-called "forward-looking" strategy and its particular variant – "one-time chance" (for more details, see [13]).

```
function FastDescentRandomAscent(p,λ);
// input: p – initial (starting) solution, λ – number of descents (λ ≥ 1)
// output: p• – resulting solution
begin
  p• := p;
  repeat
    p° := p•;
    p' := FastDescent(p•,1);  // perform a single descending 2-edge-exchange move (starting from p•)
    p~ := RandomAscent(p');  // perform random ascending move (starting from p')
    p'' := FastDescent(p~,λ);  // perform λ descending 2-edge-exchange moves (starting from p~)
    if z(p'') < z(p') then p• := p'' else p• := p'
  until z(p°) = z(p•);
  return p•
end.
```

Figure 4. Template of the basic fast descent-random ascent algorithm for the TSP

We experimentally found that it is better to use more than one type of random ascending moves. The explanation is that applying several different kinds of perturbations adds more diversity to the search process and allows covering wider regions of the search space with potentially good solutions. In particular, we operate with two types of the random ascent: 1) simple pure random ascent based on an arbitrary τ-edge-exchange move, and 2) alternative random ascent based on a so-called nearest neighbour reconnection perturbation.

The first type of ascent basically consists of a special sort τ-edge-exchange move with no reversions. There is no need in the reversal moves, since they already take place during the fast descent (see Section 1). The move complexity, i.e. the value of the factor τ is relatively small. We used $\tau = 4$ (a double-bridge move) for smaller problems and $\tau = 8$ (a four-fold-bridge move) for larger problems.

The τ-edge-exchange move may also be viewed as a multiple block swap (MBS) perturbation. The MBS perturbation procedure iteratively selects two blocks, i.e. segments in the current tour and exchanges them (see Figure 5). The segments are selected in a random way.

Regarding the alternative random ascent, it utilizes a specific type of perturbation – the nearest neighbour reconnection (NNR), which has been proven to be quite effective within the iterated tabu search method [12]. In more details, the NNR perturbation consists of three main steps (see also Figure 6). Firstly, a sub-tour is obtained by choosing η cities starting from a random city. Secondly, the given sub-tour undergoes the nearest neighbour (NN) heuristic [18]. Finally, the resulting sequence of cities is pasted to the original tour to obtain a new feasible tour. The NNR perturbation cannot be easily undone by the subsequent 2-edge-exchange moves. On the other hand, the nearest neighbour reconnection does not increase the tour length substantially, since it incorporates clever tour reconstruction heuristic instead of a blind random move. These features make the NNR perturbation an almost ideal candidate for the role of the alternative random ascent procedure; at the same time, the search process becomes highly robust (see Section 3).

The sub-tour size is controlled by the corresponding parameter, η (the NNR perturbation strength). In our implementation of the NNR procedure, η is proportional to $\sqrt{n}$, where n is the problem size.

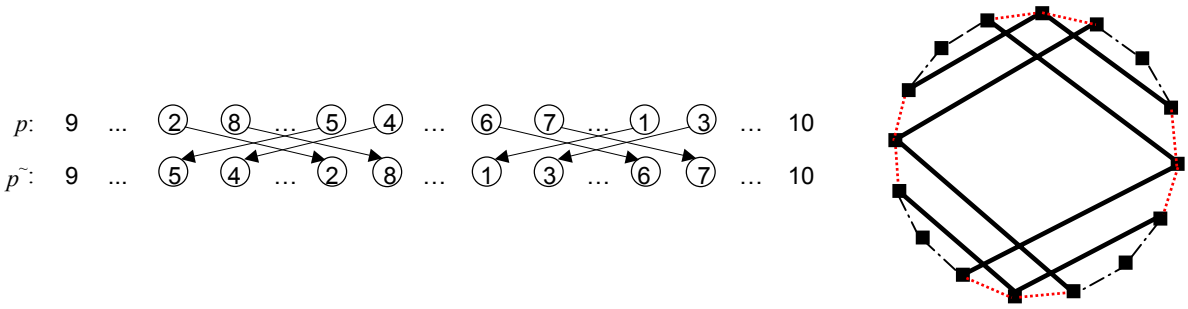

Figure 5. An example of the multiple block swap perturbation: the fourfold-bridge move

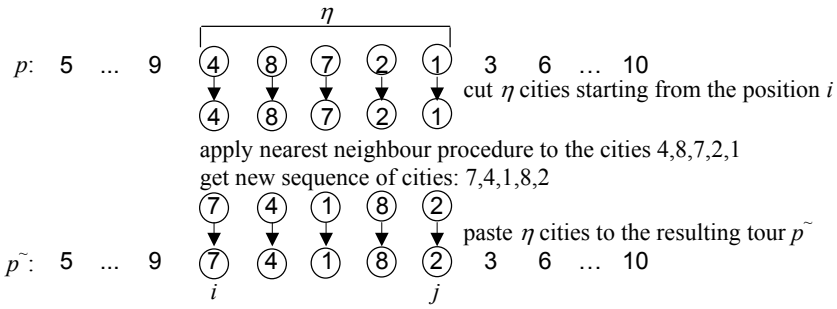

Figure 6. An example of the nearest neighbour reconnection perturbation

We can also maintain multiple random ascent trials. These trials (consisting of random ascents and λ-descents) are continued until a new better solution has been found or possibly some maximum number of trials, μ, has been reached. Here, μ is defined by the user (we used $\mu = 5$). The resulting algorithm is called as an "enhanced fast descent-random ascent" (EFDRA). It reminds rather a "more-time chance" [13] than the "one-time chance" strategy mentioned above. The template of the EFDRA algorithm is presented in Figure 7. The execution time of EFDRA is proportional to $\max\{m,\eta^2\}\cdot n\cdot K$, where m represents the candidate list size, η denotes the NNR perturbation strength, and K is some coefficient, which increases with the number of "chances" (i.e. trials of ascents and descents).

It is possible to further extend the EFDRA algorithm in a very gentle way. This new enhancement is entitled as EFDRA*. The template of EFDRA* is almost identical to the one of EFDRA, except that the call to the fast descent procedure ("FastDescent($p^{\sim},\lambda$)") is substituted by the call to the fast descent-random ascent (FDRA) procedure ("EnhancedFastDescentRandomAscent(p,λ,μ,τ,η)"). Only the call that follows the call to the alternative random ascent procedure (see Figure 7, Line 19 of the EFDRA template) is substituted. This is to avoid significant increasing in the run time of EFDRA*.

We may not limit ourselves with EFDRA*. Continuing in the above manner, it is easy to create a cascade of algorithms: EFDRA**, EFDRA***, and so on. Our most latest version of EFDRA is, in particular, EFDRA****. (The templates of EFDRA*…EFDRA**** are omitted for the sake of briefness.)

The following are the main control parameters for the algorithm EFDRA and its extensions: the candidate list size – m, the number of descents – λ, the number of trials – μ, the random move complexity – τ, and the NNR perturbation strength (sub-tour length) – η. Their values are as follows: $m = 10$, $\lambda = 30$, $\mu = 5$, $\tau = 4$ (for the smaller problems ($n \le 150$)) and $\tau = 8$ (for the larger problems ($n > 150$)), $\eta = 3\lfloor\sqrt{n}\rfloor$).

3. Results of computational experiments

To test the efficiency of the new proposed approach, a number of computational experiments have been carried out. In the experiments, the traveling salesman problem instances taken from the well-known electronic library of the TSP instances TSPLIB [16] were used.

The following are the performance measures of the algorithms: a) the average deviation of obtained 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 (i.e. the tour length) over W runs of the given algorithm, and z_{opt} is the provably optimal objective function value (the optimal tour lengths can be found in TSPLIB)); b) the number of solutions that are within 1% optimality (over W runs) – $C_{1\%}$; c) the number of the optimal solutions – C_{opt}.

```
function EnhancedFastDescentRandomAscent(p,λ,μ,τ,η);
// input: p – initial (starting) solution, λ – number of descents (λ ≥ 1), μ – number of trials of ascents (μ ≥ 1)
//        τ – random move complexity, η – alternative perturbation strength
// output: p• – resulting solution
begin
  p• := p;
  repeat // main cycle
    p° := p•; number_of_trials := 0;
    p▽ := FastDescent(p•,1); // perform a single descending 2-edge-exchange move (starting from p•)
    repeat // trials of random ascents
      p′ := p▽; number_of_trials := number_of_trials + 1;
      p˜ := RandomAscent(p′,τ); // perform random ascending move (starting from p′)
      p″ := FastDescent(p˜,λ); // perform λ 2-edge-exchange moves (starting from p˜)
      if z(p″) < z(p′)
          then p• := p″
          else begin
            p′ := p▽;
            p˜ := AlternativeRandomAscent(p′,η); // perform alternative perturbation
            p″ := FastDescent(p˜,λ); // perform λ 2-edge-exchange moves (starting from p˜)
            if z(p″) < z(p′) then p• := p″ else p• := p′
          end // else
    until (number_of_trials = μ) or (z(p•) < z(p▽))
  until z(p°) = z(p•);
  return p•
end.
```

Figure 7. Template of the enhanced fast descent-random ascent algorithm for the TSP

In the experiments conducted, five variants of the enhanced fast descent-random ascent algorithm (i.e. EFDRA, EFDRA*, EFDRA**, EFDRA***, and EFDRA****) were compared. In addition, four other heuristic algorithms were used in the comparison. They are as follows: a) the 2-opt algorithm (2-OPT); b) the 4-opt algorithm (4-OPT); c) the simulated annealing (SA) algorithm (coded by A. Misevicius); d) the fast iterated tabu search (FITS) algorithm [12]. In the case of 2-OPT, 500 repetitions are performed at every run, and only the best solution out of 500 repetitions is recorded as a result. The number of runs, W, is equal to 10 for all the algorithms, except 4-OPT, for which $W = 1$. All algorithms start from the improved initial solutions constructed by the nearest neighbour heuristic [18]. The algorithms require similar computation (CPU) time (except EFDRA, EFDRA*, EFDRA**, and EFDRA***, which consume less time, and 4-OPT, which needs much more time). 3 GHz Pentium computer was used in the experiments.

We can observe from Table 1 that the results are gradually improved (with respect to the performance measures used) as long as the number of "chances" (i.e. trials of ascents and descents) increases. This trend is especially evident for the algorithms EFDRA, EFDRA*, EFDRA**. Of course, EFDRA**** obviously outperforms all the remaining variants by consuming some more CPU time. So, we used EFDRA**** in further comparisons. The results of these comparisons are presented in Tables 2 and 3. The best results obtained are printed in bold face. (CPU times per one run are given for the algorithms 2-OPT, SA, FITS, and EFDRA****.)

Table 1. Comparison of the algorithms (Part I)

Instance	n	z_{opt}	$\bar{\delta}$, $C_{1\%}/C_{opt}$									
			EFDRA		EFDRA*		EFDRA**		EFDRA***		EFDRA****	
a280	280	2579	1.972,	1/0	0.145,	10/1	0.044,	10/7	**0**		**0**	
ch150	150	6528	2.551,	0/0	0.398,	7/1	0.267,	9/2	0.091,	10/6	**0**	
d198	198	15780	2.704,	0/0	0.112,	9/2	0.056,	10/3	0.028,	10/8	**0**	
fl417	417	11861	3.470,	0/0	0.288,	9/0	0.102,	**10**/0	0.059,	**10**/1	**0.028,**	**10/3**
gil262	262	2378	3.038,	0/0	0.384,	7/0	0.159,	9/1	0.029,	10/4	**0**	
kroa200	200	29368	3.097,	0/0	0.054,	10/3	0.004,	10/8	**0**		**0**	
lin318	318	42029	3.004,	0/0	0.773,	6/0	0.452,	7/0	0.196,	9/1	**0.094**,	**10/4**
rd400	400	15281	3.562,	0/0	0.777,	6/0	0.310,	8/0	0.213,	9/0	**0.112**,	**10/1**

Table 2. Comparison of the algorithms (Part II)

Instance	n	z_{opt}	$\bar{\delta}$, $C_{1\%}/C_{opt}$										CPU time (sec.)
			2-OPT		4-OPT		SA		FITS		EFDRA****		
a280	280	2579	2.338,	3/0	2.374,	0/0	0.314,	9/2	0.112,	10/8	**0**		*12.0*
att48	48	10628	0.010,	10/9	0.235,	1/0	0.436,	9/3	**0**		**0**		*0.06*
bayg29	29	1610	**0**		0.003,	1/0	0.031,	10/9	**0**		**0**		*0.02*
bays29	29	2020	**0**		0.396,	1/0	0.059,	10/8	**0**		**0**		*0.02*
berlin52	52	7542	0.059,	10/9	0.906,	1/0	**0**		**0**		**0**		*0.08*
bier127	127	118282	0.649,	0/0	1.598,	1/0	1.277,	2/0	0.023,	10/9	**0**		*2.5*
brazil58	58	25395	**0**		**0**		**0**		**0**		**0**		*0.09*
brg180	180	1950	**0**		**0**		9.077,	0/0	**0**		**0**		*2.3*
burma14	14	3323	**0**		**0**		**0**		**0**		**0**		*0.00*
ch130	130	6110	0.898,	5/1	0.953,	1/0	0.453,	8/1	0.027,	10/9	**0**		*3.4*
ch150	150	6528	0.200,	10/3	1.595,	0/0	0.695,	9/0	0.035,	10/9	**0**		*4.5*
d198	198	15780	0.637,	10/1	0.504,	1/0	0.181,	10/1	0.062,	10/9	**0**		*10.8*
d493	493	35002	2.026,	1/0	7.737,	0/0	0.737,	9/0	0.591,	7/**1**	**0.240**,	**10/1**	*65*
dantzig42	42	699	**0**		**0**		0.012,	10/9	**0**		**0**		*0.04*
eil51	51	426	0.069,	10/8	2.052,	0/0	0.093,	10/7	**0**		**0**		*0.06*
eil76	76	538	0.299,	10/2	1.626,	0/0	0.376,	9/3	**0**		**0**		*0.2*
eil101	101	629	0.974,	4/0	2.657,	0/0	0.493,	8/2	**0**		**0**		*1.4*
fl417	417	11861	0.880,	3/0	4.770,	0/0	1.098,	5/0	0.098,	10/1	**0.028,**	**10/3**	*45*
fri26	26	937	**0**		**0**		**0**		**0**		**0**		*0.02*
gil262	262	2378	1.656,	1/0	3.338,	0/0	0.436,	9/0	0.176,	10/1	**0**		*23*
gr17	17	2085	**0**		0.140,	1/0	**0**		**0**		**0**		*0.01*
gr21	21	2707	**0**		1.541,	0/0	**0**		**0**		**0**		*0.01*
gr24	24	1272	**0**		0.041,	1/0	**0**		**0**		**0**		*0.01*
gr48	48	5046	0.095,	10/7	0.950,	1/0	0.002,	10/9	**0**		**0**		*0.02*
gr96	96	55209	0.446,	10/3	0.406,	1/0	0.294,	10/3	**0**		**0**		*0.4*
gr120	120	6942	1.786,	2/0	2.579,	0/0	0.764,	5/0	0.090,	10/9	**0**		*2.7*
gr137	137	69853	0.819,	9/1	1.718,	0/0	0.879,	6/0	**0**		**0**		*3.0*
gr202	202	40160	2.053,	1/0	4.313,	0/0	0.513,	10/0	0.048,	10/8	**0**		*14*
gr229	229	134602	1.490,	3/0	2.496,	0/0	0.780,	8/0	0.148,	9/5	**0**		*18*
gr431	431	171414	3.111,	0/0	5.001,	0/0	1.106,	1/0	0.487,	6/0	**0.236**,	**10**/0	*60*
hk48	48	11461	**0**		0.113,	1/0	0.031,	10/6	**0**		**0**		*0.07*
kroa100	100	21282	0.043,	10/8	0.108,	1/0	0.229,	10/4	**0**		**0**		*0.5*
kroa150	150	26524	0.996,	9/1	1.727,	0/0	0.558,	8/0	**0**		**0**		*1.5*
kroa200	200	29368	0.604,	5/0	1.675,	0/0	0.625,	9/2	0.001,	10/9	**0**		*5.2*
krob100	100	22141	0.377,	10/2	2.527,	0/0	0.364,	6/0	**0**		**0**		*0.5*
krob150	150	26130	1.204,	6/1	1.572,	0/0	0.640,	4/1	0.019,	10/9	**0**		*2.5*
krob200	200	29437	1.918,	3/0	3.220,	0/0	0.616,	3/0	0.093,	10/7	**0**		*7.5*
kroc100	100	20749	0.536,	9/2	0.535,	1/0	0.197,	10/5	**0**		**0**		*1.5*
krod100	100	21294	1.483,	3/0	1.619,	0/0	0.247,	10/3	**0**		**0**		*1.4*
kroe100	100	22068	0.738,	9/0	2.035,	0/0	0.501,	8/1	**0**		**0**		*1.5*
lin105	105	14379	0.441,	10/0	2.324,	0/0	0.189,	10/4	**0**		**0**		*1.6*
lin318	318	42029	2.760,	0/0	3.212,	0/0	1.288,	2/0	0.343,	9/1	**0.094**,	**10/4**	*36*
pcb442	442	50778	3.597,	0/0	3.409,	0/0	0.929,	3/0	0.435,	9/0	**0.187**,	**10**/0	*60*
pr76	76	108159	0.239,	9/7	0.997,	1/0	0.017,	10/9	**0**		**0**		*0.2*
pr107	107	44303	0.088,	9/8	1.429,	0/0	0.003,	10/9	**0**		**0**		*0.6*
pr124	124	59030	0.733,	8/1	0.025,	1/0	0.125,	9/4	**0**		**0**		*1.0*
pr136	136	96772	3.008,	0/0	2.321,	0/0	0.552,	7/2	**0**		**0**		*1.1*
pr144	144	58537	0.336,	10/4	0.089,	1/0	0.404,	10/2	**0**		**0**		*1.4*
pr152	152	73682	0.434,	9/3	0.485,	1/0	0.297,	9/3	**0**		**0**		*1.6*
pr226	226	80369	1.136,	2/0	0.209,	1/0	0.547,	8/0	**0.001,**	**10/9**	**0.001,**	**10/9**	*5.3*
pr264	264	49135	0.536,	7/4	1.360,	0/0	0.099,	8/3	**0**		**0**		*8.0*
pr299	299	48191	1.779,	1/0	3.553,	0/0	0.612,	9/2	0.035,	10/8	**0**		*28*
pr439	439	107217	2.317,	0/0	3.965,	0/0	1.779,	0/0	0.345,	**10**/1	**0.085**,	**10/2**	*56*
rat99	99	1211	0.424,	10/2	1.007,	0/0	0.429,	7/4	**0**		**0**		*0.4*
rat195	195	2323	1.075,	5/0	2.960,	0/0	0.908,	5/1	0.004,	10/8	**0**		*6.3*
rd100	100	7910	0.921,	8/0	0.445,	1/0	0.739,	4/0	**0**		**0**		*0.7*
rd400	400	15281	3.204,	0/0	5.078,	0/0	0.777,	6/0	0.472,	9/**1**	**0.112**,	**10/1**	*52*
si175	175	21407	0.198,	10/3	0.162,	1/0	0.044,	10/8	**0**		**0**		*2.2*
st70	70	675	0.073,	10/7	0.489,	1/0	0.415,	9/2	**0**		**0**		*0.2*
swiss42	42	1273	**0**		0.514,	1/0	**0**		**0**		**0**		*0.05*
ts225	225	126643	0.782,	6/1	2.531,	0/0	1.360,	1/0	**0**		**0**		*4.8*
tsp225	225	3916	2.041,	0/0	4.882,	0/0	1.147,	1/0	0.230,10/	5	**0**		*4.9*
u159	159	42080	0.160,	10/4	0.669,	1/0	0.689,	7/0	**0**		**0**		*1.8*
ulysses16	16	6859	**0**		**0**		**0**		**0**		**0**		*0.01*
ulysses22	22	7013	**0**		**0**		**0**		**0**		**0**		*0.01*

Table 3. Cumulative characteristics of the algorithms

Characteristics \ Algorithms	2-OPT	4-OPT	SA	FITS	EFDRA****
Number of times that $\bar{\delta} = 0$	14	7	10	42	**57**
Cumulative average deviation	0.841	1.649	0.576	0.063	**0.015**
Maximum average deviation	3.597	7.737	9.077	0.591	**0.240**
Median	0.536	1.429	0.415	**0**	**0**
Standard deviation	0.950	1.640	1.148	0.137	**0.050**

From Tables 2 and 3, it can be seen that EFDRA**** is superior to other heuristic algorithms actually used in our experiments, especially the 2-opt and 4-opt algorithms. By the way, the 4-opt algorithm seems to be absolutely inefficient from both the solutions quality and computation time point of view; for example, for the 442-city instance pcb442, it took about 9 hours to get the solution that was 3.4 % above the optimal solution. It should be also noted that EFDRA**** appears to be better than the fast iterated tabu search (FITS) algorithm proposed in [12]). The difference in performance between EFDRA**** and FITS is even more clear for the larger TSP instances ($n > 300$).

4. Concluding remarks

In this paper, a new improved local search strategy called the fast descent-random ascent (FDRA) and its enhancements for the traveling salesman problem (TSP) are proposed. The fast descents are based on the modified 2-opt procedure, while random ascents are performed by using random moves and special type perturbations.

The basic FDRA algorithm and several variants of the enhanced FDRA (EFDRA) algorithm were examined on the numerous TSP instances taken from the TSP instance library – TSPLIB. The results from the experiments demonstrate that the EFDRA algorithm coupled with the proper random ascent (perturbation) procedures produces obviously better results then the other heuristic algorithms used in our experimentation. The EFDRA strategy should therefore be considered as one of the promising heuristic approaches capable of seeking optimal and near-optimal solutions in very reasonable computation times.

New modifications and further conceptual extensions of the proposed EFDRA strategy are worth examining, for example, using the recursive programming methodology, incorporating additional speeding up techniques (e.g. efficient data structures, "don't look bit" approach), implementing innovative perturbation operators, trying restart mechanisms, or hybridizing EFDRA with other metaheuristic approaches. It may also be worthy to apply the EFDRA strategy to other combinatorial optimization problems like quadratic assignment or graph partitioning problems.

References

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

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

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

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

[5] **D.S. Johnson, J.L. Bentley, L.A. McGeoch, E.E. Rothberg.** Near-optimal solutions to very large traveling salesman problems. *Monograph, to appear*.

[6] **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.

[7] **M. Jünger, G. Reinelt, G. Rinaldi.** The traveling salesman problem. *In M.Ball, T.Magnanti, C.L.Monma, G.Nemhauser* (*eds*.), *Handbook of Operations Research and Management Science: Networks, North-Holland, Amsterdam*, 1995, 225–330.

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

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

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

[11] **P.Merz, B.Freisleben.** Genetic local search for the TSP: new results. *Proceedings of* 1997 *IEEE International Conference on Evolutionary Computation* (ICEC'97) (*Indianapolis, USA*), IEEE, 1997, 159–164.

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

[13] **M. Mouhoub, Z. Wang.** Improving the ant colony optimization algorithm for the quadratic assignment problem. *Working Paper*, 2006.

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

[15] **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.

[16] **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/*].

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

[18] **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.

[19] **T. Stützle, M. Dorigo.** ACO algorithms for the traveling salesman problem. *In K.Miettinen et al*. (*eds*.), *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications, Wiley, Chichester*, 2001, 163–183.

[20] **T. Stützle, A. Grün, S. Linke, M. Rüttger.** A comparison of nature inspired heuristics on the traveling salesman problem. *In K.Deb et al*. (*eds*.), *Proceedings of the* 6*th International Conference on Parallel Problem Solving from Nature* (*Lecture Notes in Computer Science*, *Vol*.1917, *Springer, London*), 2000, 661–670.

[21] **E.G. Talbi.** A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 2002, *Vol*.8, 541–564.

Received May 2007.