European Society of Computational Methods in Sciences and Engineering (ESCMSE)

Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM)
vol. 2, no. 3-4, 2007, pp. 129-137
ISSN 1790–8140

Parallel Evolutionary Algorithm for the Traveling Salesman Problem[1]

Wojciech Bożejko[2]

Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland

Mieczysław Wodecki[3]

Institute of Computer Science,
University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland

Received 7 March, 2006; accepted in revised form 10 March, 2007

Abstract: In this paper we present a evolutionary algorithm for solving traveling salesman problem (TSP). It deals with testing feasible solutions which are local minima. This method is based on the following observation: if there are the same elements in some positions in several permutations, which are local minima, then these elements are in the same position in the optimal solution. Computational experiments on the benchmark instances from the TSP-Library [9] are presented and compared with the results yielded by the best algorithms discussed in the literature. These results show that the algorithm proposed allows us to obtain the best known results for the benchmarks in a short time.

Keywords: Traveling Salesman Problem, Parallel algorithm, Metaheuristics, Optimization

Mathematics Subject Classification: 90B80

1 Introduction

The *traveling salesman problem* (TSP) in its simplest form involves finding the shortest closed tour between a number of cities. The distances between cites are non-negative, symmetric and satisfy the triangle inequality. The TSP belongs to a class of the NP-hard problems. No optimal search algorithm, that runs in a polynomial time, has been discovered so far. Classical heuristic algorithms (tabu search, simulated annealing and genetic algorithm) converge to a certain local minimum – diversification of the search process is difficult and its efficiency is poor.

The TSP belongs to a class of the NP-hard problems, however a solution of such a problem is usually made using heuristic approach that converges to a locally optimal solution (see. [3], [8]).

[1]Published electronically October 27, 2007
[2]Corresponding author. E-mail: wojciech.bozejko@pwr.wroc.pl
[3]E-mail: mwd@ii.uni.wroc.pl

A very popular and effective approach for escaping such local optimum are the following methods: tabu search [4], simulated annealing [6], neural networks [1], [5] and evolution strategies [7],[11].

In this paper we present the evolutionary method to find a good solution to TSP. It consists in testing feasible solutions which constitute local minima. By means of computer simulations on benchmarks taken from the TSPLIB [9] we have obtained very promising results using a cluster of personal computers and the MPI library.

2 Definition of problem

The classical traveling salesman problem (TSP) is defined on an undirected graph $G = (V, E)$, where $V = \{1, 2, ..., n\}$ is a vertex (cites) set and $E = \{\{i, j\} : \ i \neq j, \ i, j \in V\}$ is an edge set. A non-negative cost (distance) matrix $C = (c_{i,j})$ is defined on E. The matrix C is symmetric ($c_{i,j} = c_{j,i}, \ i, j \in V$) and satisfies the triangle inequality ($c_{i,j} + c_{j,k} \geq c_{i,k}$, for all $i, j, k \in V$). The problem deals with finding a minimum length Hamiltonian cycle (a tour that passes through each city exactly once, and returns to the starting city) on a G.

Each feasible solution of the TSP (a cycle including all the nodes of G) is a permutation of elements of the set V. Let

$$L(\delta) = \sum_{j=1}^{n-1} c_{\delta(j),\delta(j+1)} + c_{\delta(n),\delta(1)}, \tag{1}$$

be a length of the traveling salesman's tour

$$(\delta(1), \delta(2), \ ... \ , \delta(n-1), \delta(n), \delta(1)), \ \delta \in \Pi, \tag{2}$$

where Π is a set of all permutations of elements of the set V.

Here we present a method of the algorithm's construction for solving the TSP consisting in determining and testing of local minima.

The basic idea is to start with an initial population (any subset of the solution space). Next, for each element of the population, the local optimization algorithm is applied (e.g. 2-exchange Lin-Kerninghan algorithm, the so-called "2-opt") to determine a local minimum. In this way we obtain a set of permutations – local minima. If there is an element which is in the same position in several permutation then it is fixed in this position in the permutation, and other positions and elements of permutations are still free. A new population (a set of permutations) is generated by drawing free elements in free positions (because in fixed positions are fixed elements). After determining a set of local minima (for the new population) we can increase the number of fixed elements. In every iteration "the oldest" fixed elements are set as free – to prevent finishing the algorithm's work after executing a number of iterations.

The proposed method is especially helpful in solving large-size instances of very difficult discrete optimization problems with irregular goal functions.

3 The Evolutionary Method

To solve the traveling salesman problem we propose an evolutionary algorithm which examines local minima of the cost function L. To determine the local minimum a local search algorithm is used. We apply the following notation:

π^*	:	sub-optimal permutation determined by the algorithm,
η	:	number of elements in the population,
P^i	:	population in the iteration i of the algorithm, $P^i = \{\pi_1, \pi_2, \ldots, \pi_\eta\}$,
$LocalOpt(\pi)$	:	local optimization algorithm to determine local minimum, where π is a starting solution,
LM^i	:	a set of local minima in iteration i, $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$, $\hat{\pi}_j = LocalOpt(\pi_j)$, $\pi_j \in P^i$, $j = 1, 2, \ldots, \eta$,
FS^i	:	a set of fixed elements and position in permutations of population P^i,
$FixSet(LM^i, FS^i)$	:	a procedure which determines a set of fixed elements and positions in the next iteration of evolutionary algorithm, $FS^{i+1} = FixSet(LM^i, FS^i)$,
$NewPop(FS^{i+1})$	:	a procedure which generates a new population in the next iteration of algorithm, $P^{i+1} = NewPop(FS^{i+1})$,
$\mathcal{N}(x)$	:	neighborhood of the solution x,
F	:	fitness function; equals to the length of the traveling salesman's tour L,
$nrtasks$	:	number of tasks used in the parallel version of the algorithm.

In any permutation $\pi \in P^i$ positions and elements which belong to the set FS^i (in iteration i) we call *fixed*, whereas other elements and positions we call *free*.

The algorithm's working begins with creating an initial population P^0 (and it can be created randomly). We set a sub-optimal solution π^* as the best element of the population P^0. A new population of iteration $i+1$ (a set P^{i+1}) is generated as follows: for current population P^i a set of local minima LM^i is determined (for each element $\pi \in P^i$ executing procedure $LocalOpt(\pi)$). Elements which are in the same positions in local minima are established (procedure $FixSet(LM^i, FS^i)$), and a set of fixed elements and positions FS^{i+1} is generated. Each permutation of a new population P^{i+1} includes fixed elements (in fixed positions) from the set FS^{i+1}. Free elements are randomly drawn in the remaining free positions of the permutation. If permutation $\beta \in LM^i$ exists and $F(\beta) < F(\pi^*)$, then the permutation π^* is set to β. The algorithm finishes its working after generating a fixed number of generations.

The general structure of the evolutionary algorithm for the permutation optimization problem is given below.

Algorithm 1. Evolutionary Algorithm (EA_TSP)

Initialization: a random creation of an initial population $P^0 \leftarrow \{\pi_1, \pi_2, \ldots, \pi_\eta\}$;
$\pi^* \leftarrow$ the best element of the population P^0;
a set of fixed elements $FS^0 \leftarrow \emptyset$; $i \leftarrow 0$;

repeat

Determine a set of local minima $LM^i \leftarrow \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$, where
$\hat{\pi}_j \leftarrow LocalOpt(\pi_j)$, $\pi_j \in P^i$;
for $j \leftarrow 1$ **to** η **do** **if** $F(\hat{\pi}_j) < F(\pi^*)$ **then** $\pi^* \leftarrow \hat{\pi}_j$;
Determine a set $FS^{i+1} \leftarrow FixSet(LM^i, FS^i)$ **and**

generate a new population $P^{i+1} \leftarrow NewPop(FS^{i+1})$;
$i \leftarrow i+1$;
until not a Stop Criterion is fulfilled.

Procedures *LocalOpt*, *FixSet* and *NewPop* are described in further parts of this paper.

3.1 Local optimization (*LocalOpt* procedure)

A fast method based on a local improvement is applied to determine local minima. The method begins with an initial solution π^0. In every iteration for the current solution π^i the neighborhood $\mathcal{N}(\pi^i)$ is determined. The $\mathcal{N}(\pi^i)$ is a subset of the set of feasible solutions. The neighborhood is generated by moves that are fixed transformations of solution π^i into another permutation from set of the feasible solution Π. Next, from the neighborhood the best element π^{i+1} is chosen, which is the current solution in the next iteration.

Algorithm 2. Neighborhood Search (NS)
Select a starting point x;
$x_{best} \leftarrow x$;
repeat
choose an element y from neighborhood $\mathcal{N}(x)$
according to a given criterion based on the
goal function's value $F(y)$;
$x \leftarrow y$;
if $F(y) < F(x_{best})$ **then**
$x_{best} \leftarrow y$;
until some termination condition is satisfied.

A crucial component of the local search algorithm constitutes the definition of the neighborhood function in combination with the representation of a solution. It is obvious that the choice of a good neighborhood is one of the key elements of the neighborhood search method's efficiency.

Traditionally a neighborhood of the solution π is a search space which can be defined as a set of new solutions that can be obtained from π by exactly one move (a single perturbation of π). During the iterative process the current solution of the algorithm "moves" through the solution space Π from neighbor to neighbor. A move is evaluated by comparing the goal function's value of the current solution to every single one of its neighbor.

The evolution of the solution π^i, $i = 1, 2, ...$, draws a trajectory in the search space Π. There exist many criteria for selecting the next solution π^{i+1} in the neighborhood of π^i. If the current solution is not worse then π^i, i.e. $F(\pi^{i+1}) \leq F(\pi^i)$, so such a strategy is usually called a steepest descent strategy. The main weakness of the descent algorithm is its inability to escape from local minima (all elements in the neighborhood $\mathcal{N}(\pi^i)$ are worse than π^i).
For any iteration of the local search algorithm, a subset of moves applicable to it is defined. This subset of moves generates a subset of solutions – the neighborhood. Each move transforms a permutation (a current solution) into another permutation from Π.

We use *2-exchange* moves to generate the neighborhood. It is a move on a tour T which removes two edges from T and adds two edges to T to obtain a new tour. The 2-opt neighborhood of a tour T consists of all tours that can be obtained from T by 2-*exchange* move. Let

$$\pi = (\pi(1), ..., \pi(i-1), \pi(i), \pi(i+1), ..., \pi(j-1), \pi(j), \pi(j+1), \pi(j+2), ..., \pi(n)) \qquad (3)$$

be a Hamilton cycle (the edge $(\pi(n), \pi(1))$ also belongs to a cycle). By exchange of the pair of edges $(\pi(i), \pi(i+1))$ and $(\pi(j), \pi(j+1))$ to $(\pi(i), \pi(j))$ and $(\pi(i+1), \pi(j+1))$ we obtain a new

permutation (Hamilton cycle)

$$\pi' = (\pi(1), ...\pi(i-1), \pi(i), \pi(j), \pi(j-1), ..., \pi(i+2), \pi(i+1), \pi(j+1), \pi(j+2), ..., \pi(n)). \quad (4)$$

Permutation π' is a Hamilton cycle, if $j - i \geq 3$. Generating π' we change a subsequence $\pi(i+1), \pi(i+2), \ ... \ , \pi(j-1), \pi(j)$ from the permutation π into the subsequence in an inverse order, that is $\pi(j), \pi(j-1), \ ... \ , \pi(i+2), \pi(i+1)$.

3.2 A set of fixed elements and positions (*FixSet* procedure)

A set FS^i (in iteration i) includes quadruples (a, l, α, φ), where a is an element of a set V ($a \in V$), l is a position in permutation ($1 \leq l \leq n$) and α, φ are attributes of a pair (a, l). The parameter α means "*adaptation*" and decides on inserting to the set, while φ – "*age*" – decides on deleting from the set. A $FixSet(LM^i, FS^i)$ procedure is invoked, in which the following operations are executed: (a) changing of the age of each element (φ parameter), (b) deleting the oldest elements, (c) inserting the new elements.

There are two functions of acceptance $\Gamma(i)$ and $\Phi(i)$ connected with the inserting and the deleting operations, respectively. Both of them can be determined experimentally in such a way, that the results of the algorithm are the best possible (by trying a lot of values). The values of them for the algorithm which is proposed here are presented in the 'Computational experiments' section of this paper.

3.3 Element's age modification

In each iteration of the algorithm the age of each element which belongs to FS^i is increased by 1, that is

$$\forall (a, l, \alpha, \varphi) \in FS^{i+1}, \quad (5)$$

$$FS^{i+1} \leftarrow FS^{i+1} \backslash \{(a, l, \alpha, \varphi)\} \cup \{(a, l, \alpha, \varphi + 1)\}. \quad (6)$$

The age parameter makes it possible to delete an element from the set FS^i.
Each fixed element is free after some number of iterations and can be fixed again in any free position.

3.4 Elements insertion

Let $P^i = \{\pi_1, \pi_2, \ldots, \pi_\eta\}$ be a population of η elements in an iteration i. For each permutation $\pi_j \in P^i$, applying the local search algorithm (*LocalOpt*(π_j) procedure) a set of local minima $LM^i = \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$ is determined. For any permutation

$$\hat{\pi}_j = (\hat{\pi}_j(1), \hat{\pi}_j(2), \ ... \ , \hat{\pi}_j(n)), \quad j = 1, 2, \ ... \ \eta, \quad (7)$$

let

$$nr(a, l) \geq \left| \{\hat{\pi}_j \in LM^i : \quad \hat{\pi}_j(l) = a\} \right|. \quad (8)$$

It is a number of permutations from the set LM^i, in which element a is in the position l.
If $a \in V$ is a free element and

$$\alpha = \frac{nr(a, l)}{\eta} \geq \Phi(i), \quad (9)$$

then the element a is fixed in the position l; $\varphi = 1$ and the quadruple (a, l, α, φ) is inserted to the set of fixed element and positions, that is

$$FS^{i+1} \leftarrow FS^{i+1} \cup \{(a, l, \alpha, \varphi)\}. \quad (10)$$

Acceptance function Φ should be defined so that

$$\forall i, \ \ 0 < \Phi(i) \leq 1. \quad (11)$$

3.5 Elements deletion

To test many local minima each fixed element is released after executing some number of iterations. Let

$$ES = \{(a, l, \alpha, \varphi) \in FS^{i+1} : \ \frac{\alpha}{\varphi} \leq \Gamma(i)\}. \tag{12}$$

It is a set of some elements and positions which are fixed in all permutations of the population P^i.

If $ES \neq \emptyset$, then elements of this set are deleted from FS^{i+1}, that is

$$FS^{i+1} \leftarrow FS^{i+1} \backslash ES, \tag{13}$$

otherwise (when $ES = \emptyset$), let $\delta = (a', l', \alpha', \varphi') \in FS^{i+1}$ be such that

$$\frac{\alpha'}{\varphi'} = \min\{\frac{\alpha}{\varphi} : \ (a, l, \alpha, \varphi) \in FS^{i+1}\}. \tag{14}$$

The element δ is deleted from the set FS^{i+1}, that is

$$FS^{i+1} \leftarrow FS^{i+1} \backslash \{\delta\}. \tag{15}$$

Function $\Gamma(i)$ should be defined in such a way that each element of the set FS^i is deleted after executing some number of iterations.

3.6 A new population (*NewPop* procedure)

If a quadruple $(a, l, \alpha, \varphi) \in FS^{i+1}$ then in each permutation of a new population P^{i+1} there is an element a in a position l. Therefore, to generate a new population P^{i+1} randomly drawn free elements are inserted in remaining free positions of the elements of population P^i.

Algorithm 3. New Population (*NewPop*(FS_{i+1}))

$P^{i+1} \leftarrow \emptyset$;
Determine a set of free elements
$FE \leftarrow \{a \in V : \ \neg \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$
and a set of free positions
$FP \leftarrow \{l : \ \neg \exists\, (a, l, \alpha, \varphi) \in FS^{i+1}\}$;
for $j \leftarrow$ **to** η **do** $\{$*Inserting fixed elements*$\}$
  for every $(a, l, \alpha, \varphi) \in FS^{i+1}$ **do**
    $\pi_j(l) \leftarrow a$;
  end for;
  $W \leftarrow FE$;
  $\{$*Inserting free elements*$\}$
  for $s \leftarrow 1$ **to** n **do**
    if $s \in FP$ **then** $\pi_j(s) \leftarrow w$, **where**
      $w \leftarrow random(W)$ **and** $W \leftarrow W \backslash \{w\}$;
  end for;
  $P_{i+1} \leftarrow P_{i+1} \cup \{\pi_j\}$.
end for;

Function *random* generates an element of the set W from the uniform distribution. Computational complexity of the algorithm is $O(\eta \cdot n)$.

4 Parallel Evolutionary Algorithm

For the parallel version of the evolutionary algorithm, two models of parallelization have been proposed.

Independent model. In this model the processes execute independent evolutionary algorithms (working on different subpopulations) with different parameters of fixing elements in positions. At the end, the best solution of each subpopulation is collected and the best solution of the whole algorithm is chosen. The advantage of this model is the diversification of the solutions space search process.

Cooperative model. This model works in the same way as a sequential algorithm with a big population composed of subpopulations of each process. Processes are connected with independent evolutionary algorithms and different subpopulations. The same parameters of fixing elements in positions are used in every process. In every iteration the average number $nr(a,l)$ of permutations (for all subpopulations) in which there is an element a in the position l is computed.

The general structure of the parallel evolutionary algorithm can be described as follows:

Algorithm 4. Parallel Evolutionary Algorithm (ParEA_TSP)

Initialization:
$P_p^0 \leftarrow \{\pi_1, \pi_2, \ldots, \pi_\eta\}$;
$\pi_p^* \leftarrow$ the best element of the population P_p^0;
$i \leftarrow 0$;
$FS_p^0 \leftarrow \emptyset$; p is the number of process $p = 1, 2, ..., nrtasks$;
parfor $p = 1..nrtasks$
For each process p determine sets of local minima
$LM_p^i \leftarrow \{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_\eta\}$, where $\hat{\pi}_j \leftarrow LocalOpt(\pi_j), \pi_j \in P^i$;
for $j \leftarrow 1$ **to** η **do** **if** $F(\hat{\pi}_j) < F(\pi_p^*)$ **then** $\pi_p^* \leftarrow \hat{\pi}_j$;
repeat
Determine sets $FS_p^{i+1} \leftarrow FixSet(LM_p^i, FS_p^i)$;
Compute $nr_p(a,l)$ – numbers of permutations from the set LM_p^i,
where there is an element a in the position l;
if $model = cooperative$ **then**
Exchange nr_p between processes, compute the average values
and store them in every process instead of the original nr_p values;
end if;
Generate new populations $P_p^{i+1} \leftarrow NewPop(FS_p^i)$; $i \leftarrow i + 1$;
until not a Stop Criterion is fulfilled.

Parallel version of the evolutionary algorithm was implemented in C++ language with the MPI library (MPICH) and it was tested on the cluster of 13 computers with Intel Celeron 2.4GHz processors and 240MB RAM memory, connected by Ethernet 100mbit/s. The time of calculations was the same for both sequential and parallel versions of the algorithm.

5 Computational experiments

The algorithm starts with a feasible solution $\pi_j \in P^i$, and it tries to improve this solution making small changes to it. Values of functions Γ and Φ were set to $\Gamma(i) = 0.15$ and $\Phi(i) = 0.6$. Size of the population was set to 100 individuals per processor. The algorithm stopped after 50 iterations.

In the paper [2] there are comparative results of algorithm *Meta-RaPS* and other 23 algorithms well known in literature. The fastest of these algorithms is *Meta-RaPS*. Table 1 and 2 include

Table 1: Percentage deviations from optima or best known solutions and time for tests taken from the TSPLIB. Both EA_TSP and ParEA_TSP have the same times of working.

problem	**Meta-RaPS_TSP**		**EA_TSP**		**ParEA_TSP**
	%diff	time (s)	%diff	time (s)	%diff
lin105	0.00	20	0.00	7	0.00
pr107	0.00	139	0.00	8	0.00
pr124	0.00	49	0.07	10	0.00
bier127	0.90	48	0.58	12	0.51
pr136	0.39	73	0.47	15	0.43
pr152	0.00	178	0.04	19	0.00
KroA200	1.07	190	0.86	32	0.69
KroB200	1.26	134	0.91	31	0.80
pr226	0.23	357	0.27	39	0.19
pr264	1.58	824	0.23	51	0.00
pr299	2.01	766	1.35	123	1.04
pr439	3.29	2265	2.83	314	2.28
pr1002	6.04	7032	4.13	897	3.87
Average	**1.29**	**944.8**	**0.90**	**119.8**	**0.75**

comparisons of calculations between *EA_TSP* and *Meta-RaPS* algorithms. *Meta-RaPS* algorithm was executed on the AMD 900 MHz Athlon PC, which has 3722 MIPS [10]. Celeron 2.4 GHz, on which our algorithms was executed, has 4238 MIPS [10], so it is 1.13 times faster in the integer calculations.

Table 2: Percentage deviations from optima or best known solutions for tests taken from the TSPLIB.

Problem	#cites	%Deviation to the optimal		
		Meta-RaPS_CI	**EA_TSP**	**ParEA_TSP**
Mpr21	21	0.00	0.00	0.00
Mgr48	48	0.02	0.03	0.02
Mpr76	76	1.00	0.81	0.76
KroA	100	0.50	0.04	0.00
KroB	100	0.97	0.66	0.51
KroC	100	0.85	0.48	0.27
KroD	100	1.45	1.23	0.92
KroE	100	1.30	0.29	0.02
Meil101	101	5.23	1.42	1.23
Average		**1.26**	**0.55**	**0.41**

As we can see the *EA_TSP* and its parallel version is much more faster (in average 6.93 times faster after converting the speed of AMD 900 and Celeron 2.4 processors). The average percentage deviations to optima (or the best known solutions) are much lower in *EA_TSP* than in *Meta-RaPS* [2].

6 Conclusions

We have discussed a new approach to the Traveling Salesman Problem (TSP) based on the evolutionary algorithm. The usage of the population with fixed features of local optima makes the performance of the method much better than the iterative improvement approaches. It allows the algorithm to visit the more promising areas of the solutions space and to achieve very good solutions in a much shorter time, accelerating the convergence of the algorithm. Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that the proposed algorithm provides better results than attained by the leading approaches.

In the future work we would like to examine the algorithm for other strongly NP-hard problems, i.e. Quadratic Assignment Problem and single- and multi-machine scheduling problems.

References

[1] Altinel, K., Aras, N., Oommen, J., Fast, efficient and accurate solutions to the hamiltonian path problem using neural approaches, Computers and Operations Research 27 (5), 461-494 (2000).

[2] G.W. DePuy, R.J. Morga, G.E. Whitehouse, Meta-RaPS: a simple and effective approach for solving the traveling salesman problem, Transportation Research Part E **41**, 115-130 (2005).

[3] Johnson, D.S., McGeoch, L.A., Experimental analysis of heuristics for STSP. In: Gutin, G., Punnen, (Eds.), The Traveling Salesman Problem and its Variations. Kluwer Academic Publishers, 369-443 (2002).

[4] J. Knox, Tabu search performance on the symmetric traveling salesman problem, Comp. & Oper. Res., 867-876 (1994).

[5] K.-S. Leung, H.-D. Jin, Z.-B. Xu , An expanding self-organizing neural network for the traveling salesman problem, Neurocomputing **62**, 267-292 (2004).

[6] C.C. Lo, C.C. Hus, Annealing framework with learning memory, IEEE Transaction on System, Man, Cybernetics, Part A **28** (5), 1-13 (1998).

[7] H. Muhlembein, M. Gorges-Schleuter, O. Kramer, Evolution algorithm ic combinatorial optimization, Parallel Computing, 65-85 (1988).

[8] G. Reinelt, The traveling salesman: computational Solutions for TSP applications, Berlin: Springer (1994).

[9] Reinelt, G., Bixby, W., TSPLIB-a library of traveling salesman and related problem instances. Available from <http://www.iwr.uni-heidelberg.de/groups/comopt /software/TSPLIB95/tsp/> (1995).

[10] Sisoft Sandra, http://www.sisoftware.net/

[11] C.-F. Tsai, C.-W Tsai, C.-C. Tseng, A new hybrid heuristic approach for solving large traveling salesman problem, Information Sciences **166**, 67-81 (2004).