5

A Modified Discrete Particle Swarm Optimization Algorithm for the Generalized Traveling Salesman Problem

Mehmet Fatih Tasgetiren[1], Yun-Chia Liang[2], Quan-Ke Pan[3] and P. N. Suganthan[4]
[1]*Department of Operations Management and Business Statistics, Sultan Qaboos University Muscat,*
[2]*Department of Industrial Engineering and Management, Yuan Ze University,*
[3]*College of Computer Science, Liaocheng University, Liaocheng,*
[4]*School of Electrical and Electronic Engineering, Nanyang Technological University,*
[1]*Sultanate of Oman*
[2]*Taiwan, R.O.C*
[3]*P.R. China*
[4]*Singapore*

1. Introduction

A variant of the traveling salesman problem (**TSP**) is known as the generalized traveling salesman problem (**GTSP**), where a tour does not necessarily visit all the nodes since the set N of nodes is divided into m sets or clusters, $N_1,..,N_m$ with $N_1 \cup ..\cup N_m = N$ and $N_j \cap N_k = \phi$ if $j \neq k$. The objective is to find a minimum tour length containing at least a node from each cluster N_j. Several applications of the **GTSP** can be found in postal routing [1], computer file processing [2], order picking in warehouses [3], process planning for rotational parts [4], and the routing of clients through welfare agencies [5]. Furthermore, many other combinatorial optimization problems can be reduced to the **GTSP** problem [1]. **TSP** is *NP*-Hard and hence the **GTSP** is *NP*-hard because if the set N of nodes is partitioned into $|N|$ subsets with each containing one node, it results in a **TSP**.

Regarding the literature for the **GTSP**, it was first addressed in [2, 5, 6]. Exact algorithms can be found in Laporte *et al*. [7, 8], Laporte & Nobert [9], Fischetti *et al*. [10, 11], and others in [12, 13]. On the other hand, several worthy heuristic approaches are applied to the **GTSP**. Noon [3] presented several heuristics for the **GTSP** among which the most promising one is an adaptation of the well-known nearest-neighbor heuristic for the **TSP**. Similar adaptations of the farthest-insertion, nearest-insertion, and cheapest-insertion heuristics are proposed in Fischetti *et al*. [11]. **GI3** (Generalized Initilialization, Insertion, and Improvement) is one of the most sophisticated heuristics, which is developed by Renaud & Boctor [14]. **GI3** is a generalization of the **I3** heuristic presented in Renaud *et al*. [15]. The application of the metaheuristic algorithms specifically to the **GTSP** is very rare in the litearture. A random

Source: Travelling Salesman Problem, Book edited by: Federico Greco, ISBN 978-953-7619-10-7, pp. 202, September 2008, I-Tech, Vienna, Austria

key genetic algorithm (**RKGA**) is proposed by Snyder & Daskin [16], which ignited the metaheuristic research on the **GTSP**. In the **RKGA**, random key representation is used and solutions generated by the **RKGA** are improved by using two local search heuristics namely, 2-opt and "swap" procedures. Note that their "swap" procedure provides a speed-up method in the search process. It is basically concerned with removing a node *j* from a tour, and inserting all possible nodes *k*'s from the corresponding cluster in an edge (u, v) in a tour (i.e., between the node u and the node v) with a modified nearest-neighbor criterion. They have been separately implemented by embedding them in the *level-I improvement* and *level-II improvement* procedures.

For each individual in the population, they store the original (pre-improvement) cost and the final cost after improvements have been made. When a new individual is created, they compare its pre-improvement cost to the pre-improvement cost of the individual at position $p \times N$ in the previous (sorted) population, where $p \in [0,1]$ is a parameter of the algorithm and $p = 0.05$ in Snyder & Daskin [16]. These two improvement procedures are implemented as follows:

1. If the new solution is worse than the pre-improvement cost of this individual, the *level-I improvement* is used by applying one 2-opt exchange and one "swap" procedure (assuming a profitable one can be found) and store the resulting individual.
2. On the other hand, if the new solution is better, the *level-II improvement* is used by applying *2-opt* until no profitable *2-opt* can be found, then applying "*swap*" procedures until no profitable swaps can be found, and repeat until no improvements have been made in a given pass.

The RKGA focuses on designing the local search to spend more time on improving solutions that seem promising in comparison to previous solutions and to spend less time on the others. In both *level-I* and *level-II* improvement, a ''*first-improving*'' strategy is employed where the first move of a given type improving the objective value is implemented, rather than searching for the best such move before choosing one. Thereafter, Tasgetiren *et al.* [17, 18, 19] presented a discrete particle swarm optimization algorithm a genetic algorithm (**GA**) and an iterated greedy algorithm, respectively whereas Silberholz & Golden proposed another genetic algorithm in [20] which is denoted as **mrOXGA**.

The **GSTP** may deal with either symmetric where the distance from node *j* to node *k* is the same as the distance from *k* to *j* or asymmetric distances where the distance from node *j* to node *k* is not the same as the distance from *k* to *j*. In this paper, meta-heuristics are presented to solve the **GTSP** on a standard set of benchmark instances with symmetric distances.

Particle swarm Optimization (**PSO**) is one of the most recent evolutionary meta-heuristic methods, which receives growing interest from the researchers. It is based on the metaphor of social interaction and communication such as bird flocking and fish schooling. **PSO** was first introduced to optimize various continuous nonlinear functions by Eberhart & Kennedy [21]. Distinctly different from other evolutionary-type methods such as **GA** and **ES**, **PSO** algorithms maintain the members of the entire population through the search procedure. In a PSO algorithm, each individual is called a *particle*, and each particle moves around in the multi-dimensional search space with a velocity constantly updated by the particle's own experience, the experience of the particle's neighbors, or the experience of the whole swarm. That is, the search information is socially shared among particles to direct the population towards the best position in the search space. The comprehensive surveys of the **PSO** algorithms and applications can be found in Kennedy *et al.* [22] and Clerc [23].

In this paper, a **DPSO** algorithm is presented to solve the **GTSP** on a standard set of benchmark instances with symmetric distances. Furthermore, the **DPSO** algorithm is hybridized with local search improvement heuristics to intensify the search process; hence to further improve the solution quality.
The remaining chapter is organized as follows. Section 2 introduces the **DPSO** algorithm and its basic components. Section 3 presents the computational results on benchmark problems. Finally, Section 4 summarizes the concluding remarks.

2. Discrete particle swarm optimization algorithm

In the standard **PSO** algorithm, all particles have their position, velocity, and fitness values. Particles fly through the *m*-dimensional space by learning from the historical information emerged from the swarm population. For this reason, particles are inclined to fly towards better search area over the course of evolution. Let *NP* denote the swarm size represented as $x^k = \left[x_1^k, x_2^k, ..., x_{NP}^k\right]$. Then each particle in the swarm population has the following attributes: A current position represented as $x_i^k = \left[x_{i1}^k, x_{i2}^k, .., x_{im}^k\right]$; a current velocity represented as $v_i^k = \left[v_{i1}^k, v_{i2}^k, .., v_{im}^k\right]$; a current personal best position represented as $p_i^k = \left[p_{i1}^k, p_{i2}^k, ..., p_{im}^k\right]$; and a current global best position represented as $g^k = \left[g_1^k, g_2^k, ..., g_m^k\right]$. Assuming that the function f is to be minimized, the current velocity of the *jth* dimension of the *ith* particle is updated as follows.

$$v_{ij}^k = w^{k-1} v_{ij}^{k-1} + c_1 r_1 \left(p_{ij}^{k-1} - x_{ij}^{k-1}\right) + c_2 r_2 \left(g_j^{k-1} - x_{ij}^{k-1}\right) \tag{1}$$

where w^k is the inertia weight which is a parameter to control the impact of the previous velocities on the current velocity; c_1 and c_2 are acceleration coefficients and r_1 and r_2 are uniform random numbers between [0,1]. The current position of the *jth* dimension of the *ith* particle at the generation k is updated using the previous position and current velocity of the particle as follows:

$$x_{ij}^k = x_{ij}^{k-1} + v_{ij}^k \tag{2}$$

The personal best position of each particle is updated using

$$p_i^k = \begin{cases} p_i^{k-1} & if \quad f\left(x_i^k\right) \geq f\left(p_i^{k-1}\right) \\ x_i^k & if \quad f\left(x_i^k\right) < f\left(p_i^{k-1}\right) \end{cases} \tag{3}$$

Finally, the global best position found so far in the swarm population is obtained for $1 \leq i \leq NP$ as

$$g^k = \begin{cases} \arg\min_{p_i^k} f\left(P_i^k\right) & if \quad \min f\left(p_i^k\right) < f\left(g^{k-1}\right) \\ g^{k-1} & else \end{cases} \tag{4}$$

Standard **PSO** equations cannot be used to generate binary/discrete values since positions are real-valued. Pan *et al.* [24, 25, 26] have presented a **DPSO** optimization algorithm to tackle the binary/discrete spaces, where particles are updated as follows:

$$x_i^k = c_2 \oplus CR\left(c_1 \oplus CR\left(w \oplus F_\rho\left(x_i^{k-1}\right), p_i^{k-1}\right), g^{k-1}\right) \quad (5)$$

The update equation (5) consists of three components: The first component is $a_i^k = w \oplus F_\rho\left(x_i^{k-1}\right)$, which represents the velocity of the particle. In the component $a_i^k = w \oplus F_\rho\left(x_i^{k-1}\right)$, F_ρ represents the mutation or perturbation operator with the mutation strength of ρ and the mutation probability of w. In other words, a uniform random number r is generated between 0 and 1. If r is less than w then the mutation operator is applied to generate a perturbed particle by $a_i^k = F_\rho\left(x_i^{k-1}\right)$, otherwise current particle is mutated as $a_i^k = insert\left(x_i^{k-1}\right)$. In addition, the mutation strength d is the degree of perturbation, i.e., single insert move or double insert move or some constructive heuristics generating distinct solutions and so on. In this paper, we employ the destruction and construction (**DC**) procedure of the IG algorithm in the mutation phase.

The second component is $b_i^k = c_1 \oplus CR\left(a_i^k, p_i^{k-1}\right)$, which is the "*cognition*" part of the particle representing the private thinking of the particle itself. In the component $b_i^k = c_1 \oplus CR\left(b_i^k, p_i^{k-1}\right)$, CR represents the crossover operator with the probability of c_1. Note that a_i^k and p_i^{k-1} will be the first and second parents for the crossover operator, respectively. It results either in $b_i^k = F_d\left(a_i^k, p_i^{k-1}\right)$ or in $b_i^k = a_i^k$ depending on the choice of a uniform random number.

The third component is $x_i^k = c_2 \oplus CR\left(b_i^k, g^k\right)$, which is the "*social*" part of the particle representing the collaboration among particles. In the component $x_i^k = c_2 \oplus CR\left(b_i^k, g^{k-1}\right)$, CR represents the crossover operator with the probability of c_2. Note that b_i^k and g^{k-1} will be the first and second parents for the crossover operator, respectively. It results either in $x_i^k = CR\left(b_i^k, g^{k-1}\right)$ or in $x_i^k = b_i^k$ depending on the choice of a uniform random number. The basic idea behind the **DPSO** algorithm is to provide information exchange amongst the population members, personal best solutions and the global best solution.

However, combining the particle with both personal best and then global best solution through crossover operator may cause a particle losing some genetic information. Instead, we propose a modification to our **DPSO** algorithm in this paper utilizing either the "social" or "cognitive" genetic information during the particle update process. It is achieved as follows:

$$x_i^k = \begin{cases} CR(a_i^k, p_i^{k-1}) & if \quad r < c_1 \\ CR(a_i^k, g^{k-1}) & else \end{cases} \quad (6)$$

In other words, after mutation operator, the particle is updated by recombining the temporary mutated individual with either the personal best or global best solution depending on a search directing probability of c_1. For the **DPSO** algorithm, the *gbest* (global neighborhood) model of Kennedy *et al*. [22] was followed. The pseudo code of the **DPSO** algorithm with the local search is given in Fig. 1.

```
Procedure DPSO
Initialize parameters
Initialize particles of population
Evaluate particles of population
Apply local search to population individuals                    %Optional
While (not termination) Do
    Find personal best
    Find global best
    Update particles of population
    Evaluate particles of population
    Apply local search to population individuals                %Optional
Endwhile
Return Global best
Endprocedure
```

Fig. 1. Generic Outline of DPSO Algorithm with Local Search.

2.1 Solution representation

We employ a path representation for the **GTSP** in this paper. In the path representation, each consecutive node is listed in order. An advantage of this representation is due to its simplicity in objective function evaluation since the total cost of a path can easily be calculated by summing the costs (distances) of each pair of adjacent nodes. However, a distadvantge of this representation is due to the fact that there is no quarantee that a randomly selected solution will be a valid **GTSP** tour because there is no quarantee that each cluster is represented exactly once in the path without some repair procedures. In order to handle the decision of which node should be chosen from a given cluster in the **GTSP** solution, we include both cluster and tour information in solutions. In other words, a **GTSP** solution consists of both an array of permutation of clusters (n_j) and an array of nodes (π_j) to be visited in m dimensions/clusters. In this way, each solution is guaranteed to be a GTSP solution. The solution representation together with the necessary distance information for calculating the objective function value $F(x)$ of the solution x is illustrated in Table 1 where $d_{\pi_j\pi_{j+1}}$ shows the distance from node π_j to node π_{j+1}. The initial solution is constructed in such a way that first a permutation of clusters is determined randomly, then since each cluster contains one or more nodes, a tour is established by randomly choosing a single node from each corresponding cluster. By including cluster information in solution representation, which node must be visited in a tour can be determined easily with either a random selection or a systematic way. For example, in the pair (n_j, π_j), n_j stands for the cluster in the j^{th} dimension whereas π_j represents the node to be visited from cluster n_j.

	j	1	2	...	m-1	m	1
x	n_j	n_1	n_2	...	n_{m-1}	n_m	n_1
	π_j	π_1	π_2	...	π_{m-1}	π_m	π_1
	$d_{\pi_j\pi_{j+1}}$	$d_{\pi_1\pi_2}$	$d_{\pi_2\pi_3}$	...	$d_{\pi_{m-1}\pi_m}$	$d_{\pi_m\pi_1}$	
$F(x)=$	$\sum_{j=1}^{m} d_{\pi_j\pi_{j+1}} + d_{\pi_m\pi_1} =$	$d_{\pi_1\pi_2} +$	$d_{\pi_2\pi_3} +$	...	$d_{\pi_{m-1}\pi_m} +$	$d_{\pi_m\pi_1}$	

Table 1. Solution Representation

As illustrated in Table 1, the objective function value of a solution x is the total tour length and given by

$$F(x)=\sum_{j=1}^{m-1} d_{\pi_j\pi_{j+1}} + d_{\pi_m\pi_1} \tag{1}$$

For example, consider a **GTSP** instance of **11EIL51** from **TSPLIB** library [27], which has fifty one nodes divided into eleven clusters. So the clusters are $N_1=\{19,40,41\}$, $N_2=\{3,20,35,36\}$, $N_3=\{24,43\}$, $N_4=\{33,39\}$, $N_5=\{11,12,27,32,46,47,51\}$, $N_6=\{2,16,21,29,34,50\}$, $N_7=\{8,22,26,28,31\}$, $N_8=\{13,14,18,25\}$, $N_9=\{4,15,17,37,42,44,45\}$, $N_{10}=\{1,6,7,23,48\}$, and $N_{11}=\{5,9,10,30,38,49\}$. Table 2 illustrates a random **GTSP** solution with the distance information $d_{\pi_j\pi_{j+1}}$ and the objective function $F(x)$ for the instance **11EIL51**. In addition, the whole distance matrix and other detailed information about the instance **11EIL51** can be found in http://www.ntu.edu.sg/home/EPNSugan.

	j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*1*
x	n_j	10	5	7	2	6	11	4	9	1	8	3	10
	π_j	1	51	22	20	50	10	33	44	41	25	24	1
	$d_{\pi_j\pi_{j+1}}$	$d_{1,51}$	$d_{51,22}$	$d_{22,20}$	$d_{20,50}$	$d_{50,10}$	$d_{10,33}$	$d_{33,44}$	$d_{44,41}$	$d_{41,25}$	$d_{25,24}$	$d_{24,1}$	
$F(x)$	201	14	21	15	21	17	12	17	20	21	14	29	

Table 2. **GTSP** Solution for Instance **11EIL51**

As to the construction of the initial random solution as mentioned before, first a random permutation of clusters is established; then a corresponding node is randomly chosen from each cluster to establish the tour. To be more specific, for example, in Table 2, $n_2=5$ refers to the cluster N_5, and the corresponding node $\pi_2=51$ refers to the node 51 chosen randomly from the cluster N_5.

2.2 NEH heuristic

Due to the availability of the insertion methods that we have already proposed in [17, 18, 19], it is possible to apply the **NEH** heuristic of Nawaz *et al.* [28] to the **GTSP**. Without considering cluster information for simplicity, the **NEH** heuristic for the **GTSP** can be summarized as follows:

1. Determine an initial tour of nodes. Let this tour be x.
2. The first two nodes (that is, π_1 and π_2) are chosen and two possible partial tours of these two nodes are evaluated. Note that since a tour must be Hamiltanion cycle, partial tours will be evaluated with the first node being the last node, too. As an example, partial tours, (π_1,π_2,π_1) and (π_2,π_1,π_2) are evaluated.
3. Repeat the following steps until all nodes are inserted. In the kth step, node π_k at position k is taken and tentatively inserted into all the possible k positions of the partial tour that are already partially completed. Select these k tentative partial tours

that results in the minimum objective function value or a cost function suitably predefined.

To picture out how the **NEH** heuristic can be adopted for the **GTSP**, consider a solution with five nodes as $x = \{3,1,4,2,5\}$. The following example illustrates the implementation of the **NEH** heuristic for the **GTSP**:

1. Current solution is $x = \{3,1,4,2,5\}$.
2. Evaluate the first two nodes as follows: $\{3,1,3\}$ and $\{1,3,1\}$. Assume that the first partial tour has a better objective function value than the second one. So the current partial tour will be $\{3,1\}$.
3. Insertions:
   - Insert node 4 into three possible positions of the current partial tour as follows: $\{4,3,1,4\}$, $\{3,4,1,3\}$ and $\{3,1,4,3\}$. Assume that the best objective function value is with the partial tour $\{3,4,1,3\}$. So the current partial tour will be $\{3,4,1\}$.
   - Next, insert node 2 into four possible positions of the current partial tour as follows: $\{2,3,4,1,2\}$, $\{3,2,4,1,3\}$, $\{3,4,2,1,3\}$ and $\{3,4,1,2,3\}$. Assume that the best objective function value is with the partial tour $\{3,2,4,1,3\}$. So the current partial tour will be $\{3,2,4,1\}$.
   - Finally, insert node 5 into five possible positions of the current partial tour as follows: $\{5,3,2,4,1,5\}$, $\{3,5,2,4,1,3\}$, $\{3,2,5,4,1,3\}$, $\{3,2,4,5,1,3\}$ and $\{3,2,4,1,5,3\}$. Assume that the best objective function value is with the partial tour $\{3,2,4,5,1,3\}$. So the final complete tour will be $x = \{3,2,4,5,1\}$.

2.3 Destruction and construction procedure

We employ the destruction and construction (**DC**) procedure of the iterated greedy (**IG**) algorithm in [29] in the **DPSO** algorithm. In the destruction step, a given number d of nodes, randomly chosen and without repetition, are removed from the solution. This results in two partial solutions. The first one with the size d of nodes is called x^R and includes the removed nodes in the order where they are removed. The second one with the size $m-d$ of nodes is the original one without the removed nodes, which is called x^D. It should be pointed out that we consider each corresponding cluster when the destruction and construction procedures are carried out in order to keep the feasibility of the **GTSP** tour. Note that the perturbation scheme is embedded in the destruction phase where p nodes from x^R are randomly chosen without repetition and they are replaced by some other nodes from the corresponding clusters.

The construction phase requires a constructive heuristic procedure. We employ the **NEH** heuristic described in the previous section. In order to reinsert the set x^R into the destructed solution x^D in a greedy manner, the first node π_1^R in x^R is inserted into all possible $m-d+1$ positions in the destructed solution x^D generating $m-d+1$ partial solutions. Among these $m-d+1$ partial solutions including node π_1^R, the best partial solution with the minimum tour length is chosen and kept for the next iteration. Then the

second node π_2^R in x^R is considered and so on until x^R is empty or a final solution is obtained. Hence x^D is again of size m.

To figure out how **DC** can be adopted for the **GTSP**, consider a solution with five nodes as $x = \{3,1,4,2,5\}$. Again, we do not consider cluster information for simplicity:

1. Current solution is $x = \{3,1,4,2,5\}$.
2. Remove nodes 1 and 5 randomly from the current solution to establish two partial solutions as $x^D = \{3,4,2\}$ and $x^R = \{1,5\}$.
3. Insert node 1 into **four** possible positions of the current partial tour $x^D = \{3,4,2\}$ as follows: $\{1,3,4,2,1\}$, $\{3,1,4,2,3\}$, $\{3,4,1,2,3\}$ and $\{3,4,2,1,3\}$. Assume that the best objective function value is with the partial tour $\{3,4,1,2,3\}$. So the current partial tour will be $x^D = \{3,4,1,2\}$.
4. Next, insert node 5 into **five** possible positions of the current partial tour $x^D = \{3,4,1,2\}$ as follows: $\{5,3,4,1,2,5\}$, $\{3,5,4,1,2,3\}$, $\{3,4,5,1,2,3\}$, $\{3,4,1,5,2,3\}$ and $\{3,4,1,2,5,3\}$. Assume that the best objective function value is with the final tour $\{5,3,4,1,2,5\}$. So the final complete tour will be $x = \{5,3,4,1,2\}$.

In order to highlight the difference between the **NEH** insertion and the one proposed in by Rosenkrantz *et al.* [30], we give the same example as follows:

1. Current solution is $x = \{3,1,4,2,5\}$
2. Revove nodes 1 and 5 randomly from the current solution to establish two partial solutions as $x^D = \{3,4,2\}$ and $x^R = \{1,5\}$.
3. Insert node 1 into **two** possible positions of the current partial tour $x^D = \{3,4,2\}$ as follows: $\{3,1,4,2,3\}$ and $\{3,4,1,2,3\}$ because there are only two edges in x^D. Assume that the best objective function value is with the partial tour $\{3,4,1,2,3\}$. So the current partial tour will be $x^D = \{3,4,1,2\}$.
4. Next, insert node 5 into **three** possible positions of the current partial tour $x^D = \{3,4,1,2\}$ as follows: $\{3,5,4,1,2\}$, $\{3,4,5,1,2,3\}$ and $\{3,4,1,5,2,3\}$ because there are only three edges in x^D. Assume that the best objective function value is with the final tour $\{3,5,4,1,2\}$. So the final complete tour will be $\{3,5,4,1,2,3\}$

As seen in the examples above, the **NEH** heuristic considers $(n+1)$ insertions at each step whereas the Rosenkrantz *et al.* [30] makes $(n-1)$ insertions in order to find a complete tour.

2.4 Insertion methods

The following insertion methods are proposed by the authors in [19]. These greedy speed-up methods are based on the insertion of the pair (n_k^R, π_k^R) into $m-d+1$ possible positions of a partial or destructed solution x^d. Note that as an example only a single pair is considered to be removed from the current solution, perturbed with another node from the same cluster and reinserted into the partial solution. For this reason, the destruction size and

the perturbation strength are equal to one (i.e., $\rho = d = k = 1$). As a matter of fact, the insertion of node π_k^R into $m - d - 1$ possible positions is actually proposed by Rosenkrantz *et al.* [30] for the **TSP**. Snyder & Daskin [16] have adopted it for the **GTSP**. It is based on the removal and the insertion of node π_k^R in an edge $\left(\pi_u^D, \pi_v^D\right)$ of a partial tour. However, it avoids the insertion of node π_k^R on the first and the last position of any given partial tour. We illustrate these possible three insertions using the partial solution x^D of the instance **11EIL51** having eleven clusters and nodes. Suppose that the pair $(5,51)$ is removed from the solution in Table 1; perturbed with node 27 from the same cluster N_5. So the current partial solution after removal and the pair to be reinserted are given in Tables 3A and 3B, respectively.

	j	1	2	3	4	5	6	7	8	9	10	1
x^D	n_j^D	10	7	2	6	11	4	9	1	8	3	10
	π_j^D	1	22	20	50	10	33	44	41	25	24	1
	$d_{\pi_j^D \pi_{j+1}^D}$	$d_{1,22}$	$d_{22,20}$	$d_{20,50}$	$d_{50,10}$	$d_{10,33}$	$d_{33,44}$	$d_{44,41}$	$d_{41,25}$	$d_{25,24}$	$d_{24,1}$	
$F\left(x^D\right)$	173	7	15	21	17	12	17	20	21	14	29	

Table 3A. Current Partial Solution

	k	1
x^R	n_k^R	5
	π_k^R	27

Table 3B. Partial Solution to Be Inserted

A. Insertion of pair $\left(n_k^R, \pi_k^R\right)$ *in the first position of the partial solution*

a. $\text{Remove} = d_{\pi_m^D \pi_1^D}$

b. $\text{Add} = d_{\pi_k^R \pi_1^D} + d_{\pi_m^D \pi_k^R}$

c. $F(x) = F\left(x^D\right) + \text{Add} - \text{Remove}$

Example A:

$\text{Remove} = d_{\pi_m^D \pi_1^D}$

$\text{Remove} = d_{\pi_{10}^D \pi_1^D}$

$\text{Remove} = d_{24,1}$

$\text{Add} = d_{\pi_k^R \pi_1^D} + d_{\pi_m^D \pi_k^R}$

$\text{Add} = d_{\pi_1^R \pi_1^D} + d_{\pi_{10}^D \pi_1^R}$

$\text{Add} = d_{27,1} + d_{24,27}$

$F(x) = F\left(x^D\right) + \text{Add} - \text{Remove}$

$$F(x) = d_{1,22} + d_{22,20} + d_{20,50} + d_{50,10} + d_{10,33} + d_{33,44} + d_{44,41} + d_{41,25} + d_{25,24} + d_{24,1} + d_{27,1} + d_{24,27} - d_{24,1}$$

$$F(x) = d_{1,22} + d_{22,20} + d_{20,50} + d_{50,10} + d_{10,33} + d_{33,44} + d_{44,41} + d_{41,25} + d_{25,24} + d_{27,1} + d_{24,27}$$

	j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*1*
x	n_j	**5**	10	7	2	6	11	4	9	1	8	3	5
	π_j	**27**	1	22	20	50	10	33	44	41	25	24	27
	$d_{\pi_j\pi_{j+1}}$	$d_{27,1}$	$d_{1,22}$	$d_{22,20}$	$d_{20,50}$	$d_{50,10}$	$d_{10,33}$	$d_{33,44}$	$d_{44,41}$	$d_{41,25}$	$d_{25,24}$	$d_{24,27}$	
$F(x)$	174	8	7	15	21	17	12	17	20	21	14	22	

Table 3C. Insertion of pair $\left(n_k^R, \pi_k^R\right) = (5,27)$ into the first position of partial solution

B. Insertion of pair $\left(n_k^R, \pi_k^R\right)$ *in the last position of partial solution*

a. $\text{Remove} = d_{\pi_m^D \pi_1^D}$

b. $\text{Add} = d_{\pi_m^D \pi_k^R} + d_{\pi_k^R \pi_1^D}$

c. $F(x) = F\left(x^D\right) + \text{Add} - \text{Remove}$

Example B:

$\text{Remove} = d_{\pi_m^D \pi_1^D}$

$\text{Remove} = d_{\pi_{10}^D \pi_1^D}$

$\text{Remove} = d_{24,1}$

$\text{Add} = d_{\pi_m^D \pi_k^R} + d_{\pi_k^R \pi_1^D}$

$\text{Add} = d_{\pi_{10}^D \pi_1^R} + d_{\pi_1^R \pi_1^D}$

$\text{Add} = d_{24,27} + d_{27,1}$

$F(x) = F\left(x^D\right) + \text{Add} - \text{Remove}$

$$F(x) = d_{1,22} + d_{22,20} + d_{20,50} + d_{50,10} + d_{10,33} + d_{33,44} + d_{44,41} + d_{41,25} + d_{25,24} + d_{24,1} + d_{24,27} + d_{27,1} - d_{24,1}$$

$$F(x) = d_{1,22} + d_{22,20} + d_{20,50} + d_{50,10} + d_{10,33} + d_{33,44} + d_{44,41} + d_{41,25} + d_{25,24} + d_{24,27} + d_{27,1}$$

	j	1	2	3	4	5	6	7	8	9	10	11	1
x	n_j	10	7	2	6	11	4	9	1	8	3	**5**	10
	π_j	1	22	20	50	10	33	44	41	25	24	**27**	1
	$d_{\pi_j\pi_{j+1}}$	$d_{1,22}$	$d_{22,20}$	$d_{20,50}$	$d_{50,10}$	$d_{10,33}$	$d_{33,44}$	$d_{44,41}$	$d_{41,25}$	$d_{25,24}$	$d_{24,27}$	$d_{27,1}$	
$F(x)$	174	7	15	21	17	12	17	20	21	14	22	8	

Table 3D. Insertion of the pair $\left(n_k^R, \pi_k^R\right) = (5,27)$ into the last position of partial solution

Note that even though both tours generated in the examples ***A*** and ***B*** are different, the insertion of pair $\left(n_k^R, \pi_k^R\right) = (5,27)$ into the first and last positions of the partial solution x^D is equivalent to each other in terms of distance information that they have. In addition, note that both solutions are optimal.

C. Insertion of pair $\left(n_k^R, \pi_k^R\right)$ *between the edge* $\left(n_u^D, \pi_u^D\right)$ *and* $\left(n_v^D, \pi_v^D\right)$

a. $\text{Remove} = d_{\pi_u^D \pi_v^D}$

b. $\text{Add} = d_{\pi_u^D \pi_k^R} + d_{\pi_k^R \pi_v^D}$

c. $F(x) = F\left(x^D\right) + \text{Add} - \text{Remove}$

Example C:

$u = 6$

$v = 7$

$\text{Remove} = d_{\pi_u^D \pi_v^D}$

$\text{Remove} = d_{\pi_6^D \pi_7^D}$

$\text{Remove} = d_{33,44}$

$\text{Add} = d_{\pi_u^D \pi_k^R} + d_{\pi_k^R \pi_v^D}$

$\text{Add} = d_{\pi_6^D \pi_1^R} + d_{\pi_1^R \pi_7^D}$

$\text{Add} = d_{33,27} + d_{27,44}$

$F(x) = F\left(x^D\right) + \text{Add} - \text{Remove}$

$F(x) = d_{1,22} + d_{22,20} + d_{20,50} + d_{50,10} + d_{10,33} + d_{33,44} + d_{44,41} + d_{41,25} + d_{25,24} + d_{24,1} + d_{33,27} + d_{27,44} - d_{33,44}$

$F(x) = d_{1,22} + d_{22,20} + d_{20,50} + d_{50,10} + d_{10,33} + d_{44,41} + d_{41,25} + d_{25,24} + d_{24,1} + d_{33,27} + d_{27,44}$

	j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*1*
x	n_j	10	7	2	6	11	4	5	9	1	8	3	10
	π_j	1	22	20	50	10	33	27	44	41	25	24	1
	$d_{\pi_j \pi_{j+1}}$	$d_{1,22}$	$d_{22,20}$	$d_{20,50}$	$d_{50,10}$	$d_{10,33}$	$d_{33,27}$	$d_{27,44}$	$d_{44,41}$	$d_{41,25}$	$d_{25,24}$	$d_{24,1}$	
$F(x)$	223	7	15	21	17	12	41	33	20	21	14	22	

Table 3E. Insertion of the pair $\left(n_k^R, \pi_k^R\right) = (5,27)$ between pairs $\left(n_u^D, \pi_u^D\right) = (4,33)$ and $\left(n_v^D, \pi_v^D\right) = (9,44)$.

It is important to note that above insertion methods, especially insertion to the first and the last nodes, make the **NEH** heuristic applicable in the destruction and construction procedure to establish a final complete solution. For this reason, the insertion methods given above are neccessary for an **IG** algorithm to solve the **GTSP**.

2.5 Hybridization with local search

The hybridization of **DPSO** algorithm with local search heuristics is trivial. It can be achieved through the improvement of each solution generated in the construction phase by some local search methods. As improvement heuristics, a simple local search (**LS**) method

and the 2-opt heuristic [31] were separately applied to the reconstructed solution. Note that the 2-opt heuristic is employed with the first improvement strategy in this study. Regarding the **LS** heuristic, we choose a simple one that is again based on the **DC** procedure. In other words, the destruction and construction procedures with the destruction size and the perturbation strength equal to one (i.e., $\rho = d = 1$) are used in the **LS** procedure whereas the **LS** size is fixed at $w = ncluster \times 5$ in order to intensify the search on the local minima. We will denote the hybrid **DPSO** algorithm with both local search improvement heuristics as **mDPSO** from now on. The pseudo code of the **LS** procedure is given in Fig. 2 whereas the proposed **mDPSO** algorithm is given in Fig. 3.

```
procedure LS_GTSP(x)
        h := 1
        while (h ≤ w) do
                x* := DC(x)                          % d=1 and p=1
                if (f(x*) ≤ f(x)) then
                        x := x*
                        h := 1
                else
                        h := h + 1
                endif
        endwhile
        return x
endprocedure
```

Fig. 2. Local Search Employed

```
procedure DPSO_GTSP
Set c1, w, NP, tmax
tA := GetTickCount/1000
Π = (x1^0, x2^0, .., xNP^0)                  %NEH_RANDOM population individuals and evaluate
f(xi^0), i:=1,2,..,NP                        %Evaluate population
pi^0 = xi^0, i:=1,2,..,NP                    %Initialize bestsofar population
xi^0 = two_opt(xi^0), i=1,2,..,NP            %Apply two-opt
xi^0 = LS(xi^0), i=1,2,..,NP                 %Apply LS local search
g^0 = arg min{f(xi^0)}, i=1,2,..,NP          %Find gbest solution
xB := g^0                                    %Set bestsofar
k := 1
tB := GetTickCount/1000
```

while $\left((t_B - t_A) < t_{\max}\right)$ *do*

$a_i^k = w \oplus \underset{i=1,2,..,NP}{DC_{d\rho}}\left(x_i^{k-1}\right)$ %Temporary population individual by destruction and construction

$x_B = \arg\min_{i:=1,2,..,NP}\left\{f(x_B), f\left(a_i^k\right)\right\}$ %Update bestsofar

$x_i^k = c_1 \oplus \underset{i:=1,2,..,NP}{CR}\left(a_i^k, p^{k-1}, g^{k-1}\right)$ %Update population individual by Eq. [6]

$\underset{i:=1,2,..,NP}{f\left(x_i^k\right)}$ %Evaluate population

$p_i^k = \arg\min_{i:=1,2,..,NP}\left\{f\left(x_i^k\right), f\left(p_i^{k-1}\right)\right\}$ %Update personal best

$x_i^k = \underset{i=1,2,..,NP}{two\_opt\left(x_i^k\right)}$ %Apply two-opt

$x_i^k = \underset{i=1,2,..,NP}{LS\left(x_i^k\right)}$ %Apply LS local search

$g_i^k = \arg\min_{i:=1,2,..,NP}\left\{f\left(p_i^k\right), f\left(g_i^{k-1}\right)\right\}$ %Update global best solution

$x_B = \arg\min\left\{f(x_B), f\left(g^k\right)\right\}$ %Update bestsofar

$k := k+1$

endwhile

return x_B

endprocedure

Fig. 3. **DPSO** Algorithm with Improvement Heuristics.

2.6 Crossover operator

In this paper, the traditional two-cut crossover operator is used in the **mDPSO** algorithm. The two-cut crossover operator is is illustrated in Table 4.

P_1	j	*1*	*2*	*3*	*4*	***5***	***6***	***7***	*8*	*9*	*10*	*11*	*1*
	n_j	10	5	7	2	**6**	**4**	**11**	9	8	1	3	10
	π_j	1	51	22	20	**50**	**33**	**10**	44	25	41	24	1
P_2	n_j	10	***6***	7	***11***	5	1	2	9	8	***4***	3	10
	π_j	1	***50***	22	***10***	27	41	20	44	25	***33***	24	1
O_1	j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*1*
	n_j	10	7	5	1	**6**	**4**	**11**	2	9	8	3	10
	π_j	1	22	27	41	**50**	**33**	**10**	20	44	25	24	1

Table 4. Two-Cut Crossover Operator.

2.7 Insert mutation operator

The insert mutation operator is basically related to first determining a cluster randomly, then removing the corresponding node from the tour of the individual, and replacing that

particular node with another node from the same cluster randomly. As shown in Table 5, the cluster $n_2 = 5$ is randomly chosen and its corresponding node $\pi_2 = 51$ is replaced by the node $\pi_2 = 27$ from the same cluster $n_2 = 5$ using the GTSP instance of 11EIL51.

	j	1	**2**	3	4	5	6	7	8	9	10	11	1
x	n_j	10	**5**	7	2	6	4	11	9	8	1	3	10
	π_j	1	**51**	22	20	50	33	10	44	25	41	24	1
x	j	1	**2**	3	4	5	6	7	8	9	10	11	1
	n_j	10	**5**	7	2	6	4	11	9	8	1	3	10
	π_j	1	**27**	22	20	50	33	10	44	25	41	24	1

Table 5. Insert Mutation Operator

3. Computational results

We consider **RKGA** and **mrOXGA** for comparison in this paper since they produced some of the best heuristic results for the **GTSP**. The first benchmark set contains between 51 (11) and 442 (89) nodes (clusters) and the optimal objective function value for each of the problems is available. The second benchmark set contains between 493 (99) and 1084 (217) nodes. Since optimal solutions are not available for larger instances, we compare our results to Silberholz & Golden [20]. The **DPSO** algorithm was coded in Visual C++ and run on an Intel P IV 3.20GHz with 512MB memory. The population size was fixed at 30. The initial population is constructed randomly and then the **NEH** heuristic was applied to each random solution. Destruction size and perturbation strength were taken as 5 and 3, respectively. The traditional two-cut crossover is employed where the search direction and mutation probabilities are taken as $c_1 = 0.5$ and $w = 0.9$, respectively. The **DPSO** algorithm was terminated when the best so far solution was not improved after 50 consecutive generations. Five runs were carried out for each problem instance to report the statistics based on the relative percent deviations (Δ) from optimal solutions. For the computational effort consideration, t_{avg} denotes average **CPU** time in seconds to reach the best solution found so far during the run, i.e., the point of time that the best so far solution does not improve thereafter. n_{opt} stands for the number of optimal solutions found by each algorithm whereas f_{avg} represents the average objective function values out of five runs.

We compare the **mDPSO** algorithm to two genetic algorithms, namely, **RKGA** by Snyder & Daskin [16] and **mrOXGA** by Silberholz & Golden [20] where **RKGA** is re-implemented under the same machine environment. Table 6 summarizes the solution quality in terms of relative percent deviations from the optimal values and **CPU** time requirements for all three algorithms. Note that our machine has a similar speed as Silberholz & Golden [20]. A two-sided paired *t*-test which compares the results on Table 6 with a null hypothesis that the algorithms were identical generated *p*-values of 0.167 and 0.009 for **mDPSO** vs. **mrOXGA** and **mDPSO** vs. **RKGA**, suggesting near-identical results between **mDPSO** and **mrOXGA**. On the other hand, the paired *t*-test confirms that the differences between **mDPSO** and **RKGA** were significant on the behalf of **mDPSO** subject to the fact that **RKGA** was computationally less expensive than both **mDPSO** and **mrOXGA** when solely the optimal instances are considered.

	mDPSO			mrOXGA		RKGA	
Instance	n_{opt}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}
11EIL51	5	0.00	0.10	0.00	0.26	0.00	0.08
14ST70	5	0.00	0.12	0.00	0.35	0.00	0.07
16EIL76	5	0.00	0.13	0.00	0.37	0.00	0.11
16PR76	5	0.00	0.17	0.00	0.45	0.00	0.16
20KROA100	5	0.00	0.24	0.00	0.63	0.00	0.25
20KROB100	5	0.00	0.23	0.00	0.60	0.00	0.22
20KROC100	5	0.00	0.23	0.00	0.62	0.00	0.23
20KROD100	5	0.00	0.24	0.00	0.67	0.00	0.43
20KROE100	5	0.00	0.23	0.00	0.58	0.00	0.15
20RAT99	5	0.00	0.21	0.00	0.50	0.00	0.24
20RD100	5	0.00	0.23	0.00	0.51	0.00	0.29
21EIL101	5	0.00	0.19	0.00	0.48	0.00	0.18
21LIN105	5	0.00	0.25	0.00	0.60	0.00	0.33
22PR107	5	0.00	0.23	0.00	0.53	0.00	0.20
25PR124	5	0.00	0.41	0.00	0.68	0.00	0.26
26BIER127	5	0.00	0.44	0.00	0.78	0.00	0.28
28PR136	5	0.00	0.52	0.00	0.79	0.16	0.36
29PR144	5	0.00	0.46	0.00	1.00	0.00	0.44
30KROA150	5	0.00	0.47	0.00	0.98	0.00	0.32
30KROB150	5	0.00	0.60	0.00	0.98	0.00	0.71
31PR152	5	0.00	1.38	0.00	0.97	0.00	0.38
32U159	5	0.00	0.64	0.00	0.98	0.00	0.55
39RAT195	5	0.00	0.99	0.00	1.37	0.00	1.33
40D198	5	0.00	1.77	0.00	1.63	0.07	1.47
40KROA200	5	0.00	1.11	0.00	1.66	0.00	0.95
40KROB200	5	0.00	2.44	0.05	1.63	0.01	1.29
45TS225	2	0.05	1.75	0.14	1.71	0.28	1.09
46PR226	5	0.00	0.74	0.00	1.54	0.00	1.09
53GIL262	5	0.00	4.76	0.45	3.64	0.55	3.05
53PR264	5	0.00	1.11	0.00	2.36	0.09	2.72
60PR299	1	0.07	5.66	0.05	4.59	0.16	4.08
64LIN318	5	0.00	5.72	0.00	8.08	0.54	5.39
80RD400	4	0.02	13.66	0.58	14.58	0.72	10.27
84FL417	4	0.00	13.06	0.04	8.15	0.06	6.18
88PR439	3	0.00	16.15	0.00	19.06	0.83	15.09
89PCB442	3	0.15	28.59	0.01	23.43	1.23	11.74
Avg	4.64	0.01	2.92	0.04	2.99	0.13	2.00
Machine	P IV 3.20 GHz			P IV 3.00 GHz			

Table 6. Comparison for Optimal Instances

	mDPSO		mrOXGA		RKGA	
Instance	f_{avg}	t_{avg}	f_{avg}	t_{avg}	f_{avg}	t_{avg}
11EIL51	**174.0**	100.0	**174.0**	259.2	**174.0**	78.2
14ST70	**316.0**	120.0	**316.0**	353.0	**316.0**	65.6
16EIL76	**209.0**	130.0	**209.0**	369.0	**209.0**	106.4
16PR76	**64925.0**	170.0	**64925.0**	447.0	**64925.0**	156.2
20KROA100	**9711.0**	240.0	**9711.0**	628.2	**9711.0**	249.8
20KROB100	**10328.0**	230.0	**10328.0**	603.2	**10328.0**	215.6
20KROC100	**9554.0**	230.0	**9554.0**	621.8	**9554.0**	225.0
20KROD100	**9450.0**	240.0	**9450.0**	668.8	**9450.0**	434.4
20KROE100	**9523.0**	230.0	**9523.0**	575.0	**9523.0**	147.0
20RAT99	**497.0**	210.0	**497.0**	500.0	**497.0**	243.8
20RD100	**3650.0**	230.0	**3650.0**	506.2	**3650.0**	290.8
21EIL101	**249.0**	190.0	**249.0**	478.2	**249.0**	184.6
21LIN105	**8213.0**	250.0	**8213.0**	603.2	**8213.0**	334.4
22PR107	**27898.0**	230.0	27898.6	534.4	27898.6	I97.0
25PR124	**36605.0**	410.0	**36605.0**	678.0	**36605.0**	259.0
26BIER127	**72418.0**	440.0	**72418.0**	784.4	**72418.0**	275.2
28PR136	**42570.0**	520.0	**42570.0**	793.8	42639.8	362.8
29PR144	**45886.0**	460.0	**45886.0**	1003.2	45887.4	437.6
30KROA150	**11018.0**	470.0	**11018.0**	981.2	**11018.0**	319.0
30KROB150	**12196.0**	600.0	**12196.0**	978.4	**12196.0**	712.4
31PR152	**51576.0**	1380.0	**51576.0**	965.4	**51576.0**	381.2
32U159	**22664.0**	640.0	**22664.0**	984.4	**22664.0**	553.2
39RAT195	**854.0**	990.0	**854.0**	1374.8	**854.0**	1325.0
40D198	**10557.0**	1770.0	**10557.0**	1628.2	10564.0	1468.6
40KROA200	**13406.0**	1110.0	**13406.0**	1659.4	**13406.0**	950.2
40KROB200	**13111.0**	2440.0	13117.6	1631.4	13112.2	1294.2
45TS225	**68376.0**	1750.0	68435.2	1706.2	68530.8	1087.4
46PR226	**64007.0**	740.0	**64007.0**	1540.6	**64007.0**	1094.0
53GIL262	**1013.0**	4760.0	1017.6	3637.4	1018.6	3046.8
53PR264	**29549.0**	1110.0	**29549.0**	2359.4	29574.8	2718.6
60PR299	22631.0	5660.0	**22627.0**	4593.8	22650.2	4084.4
64LIN318	**20765.0**	5720.0	**20765.0**	8084.4	20877.8	5387.6
80RD400	6362.4	13660.0	6397.8	14578.2	6407.0	10265.6
84FL417	**9651.2**	13060.0	9654.6	8152.8	9657.0	6175.2
88PR439	60099.4	16150.0	**60099.0**	19059.6	60595.4	15087.6
89PCB442	21690.0	28590.0	**21658.2**	23434.4	21923.0	11743.8
99D493	20118.6	23193.8	**20117.2**	35718.8	20260.4	14887.8
115RAT575	2419.8	33521.6	**2414.8**	48481.0	2442.4	46834.4
131P654	**27432.4**	39847.0	27508.2	32672.0	27448.4	46996.8
132D657	22714.6	64956.2	**22599.0**	132243.6	22857.6	58449.8
145U724	17422.8	141587.8	**17370.6**	161815.2	17806.2	59625.2
157RAT783	**3297.2**	114315.8	3300.2	152147.0	3341.0	89362.4
201PR1002	115759.4	231546.6	**114582.2**	464356.4	117421.2	332406.2
212U1060	**107316.4**	341759.6	108390.4	594637.4	110158.0	216999.8
217VM1084	**131716.8**	310097.4	131884.6	562040.6	133743.4	390115.6
Overal Avg	27553.3	31245.7	27554.3	50930.4	27741.3	30169.1

Table 7. Comparision to Silberholz & Golden [20]

Silberholz & Golden [20] provided larger problem instances ranging from 493 (99) to 1084 (217) nodes (clusters) where no optimal solutions are available. However, they provided the results of **mrOXGA** and **RKGA**. We compare the **mDPSO** results to those presented in Silberholz & Golden [20]. As seen in Table 7, **mDPSO** generated consistently better results than both **RKGA** and **mrOXGA** in terms of solution quality even if the larger instances are considered. In particular, 4 out 9 larger instances are further improved by **mDPSO**. The paired *t*-test on the objective function values on Table 7 confirms that the differences between **mDPSO** and **RKGA** were significant since *p*-value was 0.030 (null hypothesis is rejected) whereas **mDPSO** was equivalent to **mrOXGA** since *p*-value was 0.979. In terms of **CPU** times, the paired *t*-test on the **CPU** times confirms that the differences between **mDPSO** and **mrOXGA** were significant since the *p*-values was 0.040 whereas it was failed to reject the null hypothesis of being equal difference between **mDPSO** and **RKGA** since the *p*-value was 0.700. The paired *t*-test indicates that **mDPSO** was able to generate lower objective function values with less **CPU** times than **mrOXGA**. On the other hand, **mDPSO** yielded much better objective function values with identical **CPU** times than **RKGA**. Finally, the detailed statistics accumulated for the **mDPSO** algorithm during the runs are given in Table 8. Briefly, the statistics about the objective function values, **CPU** times, number of generations, average number of 2-opts, and average number of **DC**, respectively.

4. Conclusions

The **mDPSO** algorithm proposed employs the destruction and construction procedure of the iterated greedy algorithm (**IG**) in its mutation phase. Its performance is enhanced by employing a population initialization scheme based on an **NEH** constructive heuristic for which some speed-up methods previously developed by authors are used for greedy node insertions. Furthermore, the **mDPSO** algorithm is hybridized with local search heuristics to achieve further improvements in the solution quality. To evaluate its performance, the **mDPSO** algorithm is tested on a set of benchmark instances with symmetric Euclidean distances ranging from 51 (11) to 1084 (217) nodes (clusters) from the literature. Furthermore, the **mDPSO** algorithm was able to find optimal solutions for a large percentage of problem instances from a set of test problems in the literature. It was also able to further improve 4 out of 9 larger instances from the literature. Both solution quality and computation times are competitive to or even better than the best performing algorithms from the literature.

5. Acknowledgment

The first author dedicates this paper to Dr. Alice E. Smith from Industrial and Systems Engineering Department at Auburn University. He is grateful to Dr. Thomas Stützle from IRIDIA, University of Brussels, for his generosity in providing his IG code. We are also greatiful to Dr. Gregory Gutin and Daniel Karapetyan from University of London for preparing the larger GTSP instances. In addition, P. N. Suganthan acknowledges the financial support offered by the A*Star (Agency for Science, Technology and Research) under the grant # 052 101 0020.

Instance	f_{avg}	$f_{\min}$	$f_{\max}$	t_{avg}	$t_{\min}$	$t_{\max}$	g_{avg}	$g_{\min}$	$g_{\max}$	$2opt$	DC
11EIL51	174.0	174.0	174.0	0.1	0.1	0.2	1.0	1.0	1.0	2.0	7346.6
14ST70	316.0	316.0	316.0	0.1	0.1	0.1	1.0	1.0	1.0	2.0	10387.8
16EIL76	209.0	209.0	209.0	0.1	0.1	0.1	1.0	1.0	1.0	2.0	11026.4
16PR76	64925.0	64925.0	64925.0	0.2	0.2	0.2	1.0	1.0	1.0	2.0	14108.2
20KROA100	9711.0	9711.0	9711.0	0.2	0.2	0.3	1.0	1.0	1.0	2.0	19958.6
20KROB100	10328.0	10328.0	10328.0	0.2	0.2	0.3	1.0	1.0	1.0	2.0	18637.0
20KROC100	9554.0	9554.0	9554.0	0.2	0.2	0.3	1.0	1.0	1.0	2.0	18370.0
20KROD100	9450.0	9450.0	9450.0	0.2	0.2	0.3	1.0	1.0	1.0	2.0	19146.4
20KROE100	9523.0	9523.0	9523.0	0.2	0.2	0.3	1.0	1.0	1.0	2.0	19235.8
20RAT99	497.0	497.0	497.0	0.2	0.2	0.2	1.0	1.0	1.0	2.0	17025.2
20RD100	3650.0	3650.0	3650.0	0.2	0.2	0.2	1.0	1.0	1.0	2.0	18345.6
21EIL101	249.0	249.0	249.0	0.2	0.2	0.2	1.0	1.0	1.0	2.0	15256.0
21LIN105	8213.0	8213.0	8213.0	0.3	0.2	0.3	1.0	1.0	1.0	2.0	20275.6
22PR107	27898.0	27898.0	27898.0	0.2	0.2	0.2	1.0	1.0	1.0	2.0	17978.0
25PR124	36605.0	36605.0	36605.0	0.4	0.3	0.7	1.8	1.0	4.0	2.8	31702.0
26BIER127	72418.0	72418.0	72418.0	0.4	0.3	0.6	1.8	1.0	3.0	2.8	34417.4
28PR136	42570.0	42570.0	42570.0	0.5	0.4	0.8	2.0	1.0	4.0	3.0	39157.2
29PR144	45886.0	45886.0	45886.0	0.5	0.4	0.7	1.4	1.0	3.0	2.4	34640.6
30KROA150	11018.0	11018.0	11018.0	0.5	0.4	0.6	1.2	1.0	2.0	2.2	35139.2
30KROB150	12196.0	12196.0	12196.0	0.6	0.4	1.3	2.2	1.0	7.0	3.2	44800.0
31PR152	51576.0	51576.0	51576.0	1.4	0.5	2.3	6.6	1.0	13.0	7.6	102702.0
32U159	22664.0	22664.0	22664.0	0.6	0.5	1.0	2.2	1.0	5.0	3.2	47115.2
39RAT195	854.0	854.0	854.0	1.0	0.6	1.2	2.6	1.0	4.0	3.6	68885.4
40D198	10557.0	10557.0	10557.0	1.8	0.7	2.5	5.8	1.0	10.0	6.8	123194.6
40KROA200	13406.0	13406.0	13406.0	1.1	0.7	1.3	2.8	1.0	4.0	3.8	76493.0
40KROB200	13111.0	13111.0	13111.0	2.4	1.2	4.1	9.6	3.0	16.0	10.6	169724.4
45TS225	68376.0	68340.0	68400.0	1.7	0.7	3.3	6.2	1.0	16.0	37.2	418896.2
46PR226	64007.0	64007.0	64007.0	0.7	0.7	0.8	1.0	1.0	1.0	2.0	48324.4
53GIL262	1013.0	1013.0	1013.0	4.8	2.0	9.1	16.2	4.0	37.0	17.2	300605.2
53PR264	29549.0	29549.0	29549.0	1.1	1.0	1.4	1.2	1.0	2.0	2.2	68722.2
60PR299	22631.0	22615.0	22635.0	5.7	4.0	7.9	13.8	8.0	29.0	54.8	860095.2
64LIN318	20765.0	20765.0	20765.0	5.7	3.2	9.7	12.4	5.0	30.0	13.4	334602.4
80RD400	6362.4	6361.0	6368.0	13.7	6.7	17.3	18.6	8.0	30.0	29.6	911334.6
84FL417	9651.2	9651.0	9652.0	13.1	11.0	15.7	32.6	24.0	44.0	43.6	829024.2
88PR439	60099.4	60099.0	60100.0	16.2	8.2	24.8	28.4	9.0	48.0	49.4	1173370.8
89PCB442	21690.0	21657.0	21802.0	28.6	8.1	59.6	57.2	10.0	125.0	78.2	1813548.8
99D493	20118.6	20045.0	20271.0	23.2	9.7	39.6	30.4	7.0	67.0	81.4	2240001.4
115RAT575	2419.8	2388.0	2449.0	33.5	20.5	43.4	32.0	18.0	50.0	83.0	2681845.4
131P654	27432.4	27432.0	27433.0	39.8	11.8	54.7	58.0	12.0	83.0	109.0	2740248.6
132D657	22714.6	22543.0	22906.0	65.0	38.1	85.1	61.2	22.0	91.0	112.2	3891504.4
145U724	17422.8	17257.0	17569.0	141.6	64.8	209.1	100.2	38.0	171.0	151.2	6502515.2
157RAT783	3297.2	3283.0	3324.0	114.3	80.2	157.3	70.2	47.0	99.0	121.2	5182433.0
201PR1002	115759.4	114731.0	116644.0	231.5	131.5	325.1	70.2	40.0	125.0	121.2	7972666.2
212U1060	107316.4	106659.0	107937.0	341.8	169.7	514.4	125.4	65.0	208.0	176.4	10209723.6
217VM1084	131716.8	131165.0	132394.0	310.1	133.9	389.8	113.6	36.0	156.0	164.6	9468416.8
Avg	27553.3	27491.5	27617.2	31.2	15.9	44.2	20.1	8.5	33.4	34.0	1304065.5

Table 8. Experimental Data Collected for **mDPSO**

6. References

G. Laporte, A. Asef-Vaziri, C. Sriskandarajah, Some applications of the generalized travelling salesman problem, Journal of the Operational Research Society 47 (12) (1996) 461–1467.

A. Henry-Labordere, The record balancing problem—A dynamic programming solution of a generalized travelling salesman problem, Revue Francaise D Informatique DeRecherche Operationnelle 3 (NB2) (1969) 43–49.

C.E. Noon, The generalized traveling salesman problem, Ph.D. thesis, University of Michigan, 1988.

D. Ben-Arieh, G. Gutin, M. Penn, A. Yeo, A. Zverovitch, Process planning for rotational parts using the generalized traveling salesman problem, International Journal of Production Research 41 (11) (2003) 2581–2596.

J.P. Saskena, Mathematical model of scheduling clients through welfare agencies, Journal of the Canadian Operational Research Society 8 (1970) 185–200.

S.S. Srivastava, S. Kumar, R.C. Garg, P. Sen, Generalized traveling salesman problem through n sets of nodes, Journal of the Canadian Operational Research Society 7 (1970) 97–101.

G. Laporte, H. Mercure, Y. Nobert, Finding the shortest Hamiltonian circuit through n clusters: A Lagrangian approach, Congressus Numerantium 48 (1985) 277–290.

G. Laporte, H. Mercure, Y. Nobert, Generalized travelling salesman problem through n-sets of nodes—The asymmetrical case, Discrete Applied Mathematics 18 (2) (1987) 185–197.

G. Laporte, Y. Nobert, Generalized traveling salesman problem through n-sets of nodes—An integer programming approach, INFOR 21 (1) (1983) 61–75.

M. Fischetti, J.J. Salazar-Gonzalez, P. Toth, The symmetrical generalized traveling salesman polytope, Networks 26(2) (1995) 113–123.

M. Fischetti, J.J. Salazar-Gonza´lez, P. Toth, A branch-and-cut algorithm for the symmetric generalized travelling salesman problem, Operations Research 45 (3) (1997) 378–394.

A.G. Chentsov, L.N. Korotayeva, The dynamic programming method in the generalized traveling salesman problem, Mathematical and Computer Modelling 25 (1) (1997) 93–105.

C.E. Noon, J.C. Bean, A Lagrangian based approach for the asymmetric generalized traveling salesman problem, Operations Research 39 (4) (1991) 623–632.

J. Renaud, F.F. Boctor, An efficient composite heuristic for the symmetric generalized traveling salesman problem, European Journal of Operational Research 108 (3) (1998) 571–584.

J. Renaud, F.F. Boctor, G. Laporte, A fast composite heuristic for the symmetric traveling salesman problem, INFORMS Journal on Computing 4 (1996) 134–143.

L.V. Snyder and M.S. Daskin, A random-key genetic algorithm for the generalized traveling salesman problem, European Journal of Operational research 174 (2006) 38-53.

M.F. Tasgetiren, P.N. Suganthan, Q.-K. Pan, A discrete particle swarm optimization algorithm for the generalized traveling salesman problem, In the Proceedings of the 9th annual conference on genetic and evolutionary computation (GECCO2007), 2007, London, UK, pp.158-167.

M.F. Tasgetiren, P.N. Suganthan, Q.-K. Pan, Y.-C. Liang, A genetic algorithm for the generalized traveling salesman problem, In the Proceeding of the World Congress on Evolutionary Computation (CEC2007), 2007, Singapore, p:2382-2389.

M.F. Tasgetiren, P.N. Suganthan, Q.-K. Pan, Y.-C. Liang, A hybrid iterated greedy algorithm for the generalized traveling salesman problem, 2006, Under second revision by European Journal of Operational Research.

J. Silberholz, B. Golden, The generalized traveling salesman problem: A new genetic algorithm approach, In: Edward K. B. et al. (Eds.), Extending the horizons: Advances in Computing, Optimization and Decision Technologies. Vol. 37, Springer-Verlag, pp. 165-181.

R.C. Eberhart, J. Kennedy A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995. p. 39-43.

J. Kennedy, R. C. Eberhart, and Y. Shi, Swarm Intelligence. San Mateo, Morgan Kaufmann, CA, USA, 2001.

M. Clerc Particle Swarm Optimization, ISTE Ltd., France, 2006.

Q.-K. Pan, M.F. Tasgetiren, Y.-C. Liang, 2006a, Minimizing total earliness and tardiness penalties with a common due date on a single machine using a discrete particle swarm optimization algorithm. In: Proceedings of Ant Colony Optimization and Swarm Intelligence (ANTS2006), LNCS 4150, Springer-Verlag, pp. 460-467.

Q.-K. Pan, M.F. Tasgetiren, Y.-C. Liang, 2006b, A discrete particle swarm optimization algorithm for the permutation flowshop sequencing problem with makespan criterion. In: Proceedings of the 26th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI-2006), Cambridge, UK, pp. 19-31.

Q.-K. Pan, M.F. Tasgetiren, Y.-C. Liang, 2007a, A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem with makespan and total flowtime criteria. Computers and Operations Research 35(9) (2008) 2807-2839.

G. Reinelt, TSPLIB—A traveling salesman problem library, ORSA Journal on Computing 4 (1996) 134–143.

M. Nawaz, E.E. Enscore Jr., I.A. Ham., Heuristic algorithm for the m-machine, n-job flow shop sequencing problem. OMEGA; 11(1) (1983) 91-95.

R. Ruiz and T. Stutzle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 174 (2006)38-53.

D. Rosenkrantz, R. Stearns, P. Lewis, Approximate algorithms for the traveling salesman problem. Proceedings of the 15th Annual Symposium of Switching and Automata Theory 1974 .33-42.

S. Lin, B.W. Kernighan, An effective heuristic algorithm for the traveling salesman problem Operations Research, 21 (1973) 498-516.

Traveling Salesman Problem

Edited by Federico Greco

ISBN 978-953-7619-10-7
Hard cover, 202 pages
Publisher InTech
Published online 01, September, 2008
Published in print edition September, 2008

The idea behind TSP was conceived by Austrian mathematician Karl Menger in mid 1930s who invited the research community to consider a problem from the everyday life from a mathematical point of view. A traveling salesman has to visit exactly once each one of a list of m cities and then return to the home city. He knows the cost of traveling from any city i to any other city j. Thus, which is the tour of least possible cost the salesman can take? In this book the problem of finding algorithmic technique leading to good/optimal solutions for TSP (or for some other strictly related problems) is considered. TSP is a very attractive problem for the research community because it arises as a natural subproblem in many applications concerning the every day life. Indeed, each application, in which an optimal ordering of a number of items has to be chosen in a way that the total cost of a solution is determined by adding up the costs arising from two successively items, can be modelled as a TSP instance. Thus, studying TSP can never be considered as an abstract research with no real importance.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Mehmet Fatih Tasgetiren, Yun-Chia Liang, Quan-Ke Pan and P. N. Suganthan (2008). A Modified Discrete Particle Swarm Optimization Algorithm for the Generalized Traveling Salesman Problem, Traveling Salesman Problem, Federico Greco (Ed.), ISBN: 978-953-7619-10-7, InTech, Available from: http://www.intechopen.com/books/traveling_salesman_problem/a_modified_discrete_particle_swarm_optimization_algorithm_for_the_generalized_traveling_salesman_pro

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821