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

M. Fatih Tasgetiren
Department of Operations Management and Business Statistics, Sultan Qaboos University, Muscat, Oman
mfatih@squ.edu.om

P. N. Suganthan
School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore 639798
epnsugan@ntu.edu.sg

Quan-Ke Pan
College of Computer Science, Liaocheng University, Liaocheng, Shandong Province, 252059, P. R. China
qkpan@lcu.edu.cn

ABSTRACT

Dividing the set of nodes into clusters in the well-known traveling salesman problem results in the generalized traveling salesman problem which seeking a tour with minimum cost passing through only a single node from each cluster. In this paper, a discrete particle swarm optimization is presented to solve the problem on a set of benchmark instances. The discrete particle swarm optimization algorithm exploits the basic features of its continuous counterpart. It is also hybridized with a local search, variable neighborhood descend algorithm, to further improve the solution quality. In addition, some speed-up methods for greedy node insertions are presented. The discrete particle swarm optimization algorithm is tested on a set of benchmark instances with symmetric distances up to 442 nodes from the literature. Computational results show that the discrete particle optimization algorithm is very promising to solve the generalized traveling salesman problem.

Categories and Subject Descriptors

I.2.8 [**Computing Methodology**]: Problem Solving, Control Methods, and Search – *heuristic methods*

General Terms

Algorithms

Keywords

Generalized traveling salesman problem, discrete particle swarm optimization problem, iterated greedy algorithm, variable neighborhood descend algorithm.

1. INTRODUCTION

A variant of a well-known traveling salesman problem where a tour does not necessarily visit all nodes is so called the generalized traveling salesman problem (GTSP). More specifically, the set of N nodes is divided into m sets or clusters such that $N = \{N_1,..,N_m\}$ with $N = \{N_1 \cup .. \cup N_m\}$ and $N_j \cap N_k = \phi$ where the objective is to find a minimum tour length containing exactly one node from each cluster N_j .There exist several applications of the GTSP such as postal routing in [1], computer file processing in [2], order picking in warehouses in [3], process planning for rotational parts in [4], and the routing of clients through welfare agencies in [5]. Furthermore, many other combinatorial optimization problems can be reduced to the GTSP problem [6]. The GTSP is NP-hard since it is a special case of the TSP which is partitioned into m clusters with each containing only one node. Regarding the literature for the GTSP, exact algorithms can be found in Laporte et al.[7, 8, 9], Fischetti et al. [10, 11], and others in [12, 13] whereas heuristic approaches are applied in Noon [3], Fischetti et al. [11], Renaud and Boctor [14, 15]. Genetic algorithm applied to the GTSP is the recent random key genetic algorithm (GA) by Snyder and Daskin [16]. The GSTP may deal with both symmetric and asymmetric distances. In this paper, a discrete particle swarm optimization algorithm is presented to solve the GTSP on a standard set of benchmark instances with symmetric distances.

The remaining paper is organized as follows. Section 2 introduces the discrete particle swarm optimization (DPSO) algorithm. Computational results are discussed in Section 3. Finally, Section 4 summarizes the concluding remarks.

2. DPSO 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 population size represented as $X^t = [X_1^t, X_2^t,.., X_{NP}^t]$. Then each particle in the swarm population has the following attributes: A current position represented as $X_i^t = [x_{i1}^t, x_{i2}^t,.., x_{im}^t]$; a current velocity represented as $V_i^t = [v_{i1}^t, v_{i2}^t,.., v_{in}^t]$; a current personal best position represented as $P_i^t = [p_{i1}^t, p_{i2}^t,.., p_{im}^t]$; and a current global best position represented as $G^t = [g_1^t, g_2^t,.., g_m^t]$. Assuming that the function f is to be minimized, the current velocity of the *jth* dimension of the *ith* particle is updated as follows.

$$v_{ij}^t = w^{t-1} v_{ij}^{t-1} + c_1 r_1 \left(p_{ij}^{t-1} - x_{ij}^{t-1}\right) + c_2 r_2 \left(g_j^{t-1} - x_{ij}^{t-1}\right) \qquad (2)$$

GECCO'07, July 7–11, 2007, London, England, United Kingdom.
Copyright 2007 ACM 978-1-59593-697-4/07/0007...$5.00.

where w^t 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 is updated using the previous position and current velocity of the particle as follows:

$$x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t \tag{3}$$

The personal best position of each particle is updated using

$$P_i^t = \begin{cases} P_i^{t-1} & if \quad f\left(X_i^t\right) > f\left(P_i^{t-1}\right) \\ X_i^t & if \quad f\left(X_i^t\right) \leq f\left(P_i^{t-1}\right) \end{cases} \tag{4}$$

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

$$G^t = \begin{cases} \arg\min\limits_{P_i^t} f\left(P_i^t\right) & if \quad \min f\left(P_i^t\right) \leq f\left(G^{t-1}\right) \\ G^{t-1} & else \end{cases} \tag{5}$$

Standard PSO equations cannot be used to generate binary/discrete values since positions are real-valued. Pan et al. [17] have presented a DPSO optimization algorithm to tackle the discrete spaces, where particles are updated as follows:

$$X_i^t = c_2 \oplus CR_2\left(c_1 \oplus CR_1\left(w \oplus DC_1\left(X_i^{t-1}\right), P_i^{t-1}\right), G^{t-1}\right) \tag{6}$$

Given that λ_i, and δ_i are two temporary particles, the update equation (6) consists of three operators: The first operator is $\lambda_i^t = w \otimes DC_1\left(X_i^{t-1}\right)$, where DC_1 represents the destruction and construction operator with the 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 destruction and construction operator is applied to generate a perturbed particle by $\lambda_i^t = DC_1\left(X_i^{t-1}\right)$, otherwise current particle is kept as $\lambda_i^t = X_i^{t-1}$. Note that the destruction size and perturbation strength are taken as $ds = 4$ and $ps = 4$, respectively in carrying out the destruction and construction procedure. The second operator is $\delta_i^t = c_1 \oplus CR_1\left(\lambda_i^t, P_i^{t-1}\right)$, where CR_1 represents the crossover operator with the probability of c_1. Note that λ_i^t and P_i^{t-1} will be the first and second parents for the crossover operator, respectively. It results either in $\delta_i^t = CR_1\left(\lambda_i^t, P_i^{t-1}\right)$ or in $\delta_i^t = \lambda_i^t$ depending on the choice of a uniform random number. The third operator is $X_i^t = c_2 \otimes CR_2\left(\delta_i^t, G^t\right)$, where CR_2 represents the crossover operator with the probability of c_2. Note that δ_i^t and G^{t-1} will be the first and second parents for the crossover operator, respectively. It results either in $X_i^t = CR_2\left(\delta_i^t, G^{t-1}\right)$ or in $X_i^t = \delta_i^t$ depending on the choice of a uniform random number. For the DPSO algorithm, the *gbest* (global neighborhood) model of Kennedy et al. [22] was followed. The pseudo code of the DPSO algorithm for the GTSP is given in Figure 1.

```
Procedure DPSO
initialize parameters
initialize particles of population
evaluate particles of population
apply two_opt local search to personal best population
apply VND local search to personal best population
while (not termination) do
    find the personal best
    find the global best
    update particles of population
    evaluate particles of population
    apply two_opt local search to personal best population
    apply VND local search to personal best population
endwhile
return Global best
end
```

Figure 1. DPSO Algorithm for GTSP

2.1 Solution Representation

In order to handle the GTSP properly, we present a unique solution representation where it includes both permutation of clusters (n_j) and tour containing the nodes (π_j) to be visited in m dimensions/clusters. Solution representation along with the distance information is given in Figure 2 where $d_{\pi_j \pi_{j+1}}$ shows the corresponding distance from node π_j to π_{j+1}. A random solution is constructed in a way that first a permutation of clusters is determined randomly. Then since each cluster contains one or more nodes, the tour is established by randomly choosing a single node from each corresponding cluster. For simplicity, we omit the index i of particle X_i from the representation.

	j	1	2	.	m-1	m	1
X	π_j	π_1	π_2	.		π_m	π_1
	n_j	n_1	n_2	.		n_m	n_1
	$d_{\pi_j \pi_{j+1}}$	$d_{\pi_1 \pi_2}$	$d_{\pi_2 \pi_3}$	.	$d_{\pi_{m-1} \pi_m}$	$d_{\pi_m \pi_1}$	

Figure2. Solution Representation.

Then, the fitness function of the particle is the total tour length and given by

$$F(X) = \sum_{j=1}^{m-1} d_{\pi_i \pi_{i+1}} + d_{\pi_m \pi_1} \tag{7}$$

For example, consider a GTSP instance with $N = \{1,..,25\}$ where the clusters are $N_1 = \{1,..,5\}$, $N_2 = \{6,..,10\}$, $N_3 = \{11,..,15\}$, $N_4 = \{16,..,20\}$, and $N_5 = \{21,..,25\}$. Figure 3 illustrates the example solution in detail:

	j	1	2	3	4	5	1
X	π_j	14	5	22	8	16	14
	n_j	3	1	5	2	4	3
	$d_{\pi_j \pi_{j+1}}$	$d_{14,5}$	$d_{5,22}$	$d_{22,8}$	$d_{8,16}$	$d_{16,14}$	

Figure 3. Example Instance.

So, the fitness function of the particle is given by

$$F(X) = d_{14,5} + d_{5,22} + d_{22,8} + d_{8,16} + d_{16,14}$$

2.2 Iterated Greedy Algorithm

Iterated greedy (IG) algorithm has been successfully applied to the Set Covering problem (SCP) in Jacobs and Brusco [18], and

Marchiory and Steenbeek [19], and the permutation flowshop scheduling problem in Ruiz and Stutzle [20]. In the context of the GTSP, the destruction and construction procedure is applied to the particle. d nodes with corresponding clusters are randomly chosen from the solution to be removed and a partial solution $\left(n_{D,j}, \pi_{D,j}\right)$ for $j = 1,..,m-d$ is established. At the same time, the set of d nodes and clusters $\left(n_{R,k}, \pi_{R,k}\right)$ for $k = 1,..,d$ is also established to be reinserted into the partial solution $\left(n_{D,j}, \pi_{D,j}\right)$. The construction phase requires a heuristic procedure to reinsert the set $\left(n_{R,k}, \pi_{R,k}\right)$ onto the partial solution $\left(n_{D,j}, \pi_{D,j}\right)$ in a greedy manner. In other words, the first pair in the set $\left(n_{R,k}, \pi_{R,k}\right)$ is reinserted into all possible $m-d+1$ positions in the partial solution $\left(n_{D,j}, \pi_{D,j}\right)$. Among these $m-d+1$ insertions, the best solution with the minimum partial tour length is chosen as the current partial solution for the next insertion. Then the second pair in the set $\left(n_{R,k}, \pi_{R,k}\right)$ is considered and so on until the set $\left(n_{R,k}, \pi_{R,k}\right)$ is empty.

The destruction and construction procedure for the GTSP is illustrated in the following example. Note that the destruction size is $d = 2$ and the perturbation strength is $p = 1$ in this example. Perturbation strength $p = 1$ indicates replacing only a single node with another one from the same cluster.

CURRENT PARTICLE

j	1	2	3	4	5	1
π_j	14	5	22	8	16	14
n_j	3	1	5	2	4	3

DESTRUCTION PHASE

Step 1.a. Choose $d = 2$ nodes with corresponding clusters, randomly.

j	1	**2**	3	**4**	5	1
π_j	14	**5**	22	**8**	16	14
n_j	3	**1**	5	**2**	4	3

Step 1.b. Establish $\pi_{D,j} = \{14,22,16\}$, $n_{D,j} = \{3,5,4\}$, $\pi_{R,k} = \{5,8\}$ and $n_{R,k} = \{1,2\}$.

j	1	2	3	4	k	1	2
π_D	14	22	16	14	π_R	5	8
n_D	3	5	4	3	n_R	1	2

Step 1.c. Perturb $\pi_{R,k} = \{5,8\}$ to $\pi_{R,k} = \{5,9\}$ by randomly choosing $n_{R,2} = 2$ in the set $n_{R,k} = \{1,2\}$, and randomly replacing $x_{R,2} = 8$ with $x_{R,2} = 9$ from the cluster N_2.

j	1	2	3	4	k	1	2
$\pi_{D,j}$	14	22	16	14	$x_{R,k}$	5	**9**
$n_{D,j}$	3	5	4	3	$n_{R,k}$	1	2

CONSTRUCTION PHASE

Step2.a. After the best insertion of the pair $\left(\pi_{R,1}, n_{R,1}\right) = (5,1)$.

j	1	2	3	4	5	k	1
$\pi_{D,j}$	14	22	**5**	16	14	x_R	9
$n_{D,j}$	3	5	**1**	4	3	n_R	2

Step2.b. After the best insertion of the pair $\left(\pi_{R,2}, n_{R,2}\right) = (9,2)$.

j	1	2	3	4	5	1
x_j	14	**9**	22	**5**	16	14
n_j	3	**2**	5	**1**	4	3

2.3 Insertion Methods

In order to accelerate the search process during both the mutation phase of the DPSO algorithm and the VND local search, we present the following speed-up methods based on the insertion of the pair $\left(n_{R,k}, \pi_{R,k}\right)$ into $m-d+1$ possible slots of a partial solution $\left(n_{D,j}, \pi_{D,j}\right)$. Note that insertion of the node $\pi_{R,k}$ into $m-1$ possible slots is given in Snyder and Daskin [16], i.e., basically an insertion of the node $\pi_{R,k}$ in between an edge $\left(\pi_{D,u}, \pi_{D,v}\right)$ in a partial solution. However, it avoids the insertion of the node $\pi_{R,k}$ on the first and last slots of any given tour. Supposing that the node $\pi_{R,k}$ will be inserted on a tour of a particle with $m = 4$ nodes, we illustrate these three possible insertions with the examples below:

CURRENT PARTICLE

j	1	2	3	4	1	k	1
$\pi_{D,j}$	14	5	22	16	14	$x_{R,k}$	8
$n_{D,j}$	3	1	5	4	3	$n_{R,k}$	2
$d_{\pi_{D,j}\pi_{D,j+1}}$	$d_{14,5}$	$d_{5,22}$	$d_{22,16}$	$d_{16,14}$			

A. Insertion of the node $\pi_{R,k}$ in the first slot in a particle.

a. $\mathrm{Re}\,move = d_{\pi_{D,m}\pi_{D,1}}$ $Add = d_{\pi_{R,k}\pi_{D,1}} + d_{\pi_{D,m}\pi_{R,k}}$

b. $F(X) = F(X_D) + Add - \mathrm{Re}\,move$

Example A. Insertion of the node $\pi_{R,k} = 8$ in the first slot

j	1	2	3	4	5	1
π_j	8	14	5	22	16	8
n_j	2	3	1	5	4	2
$d_{\pi_j\pi_{j+1}}$	$d_{8,14}$	$d_{14,5}$	$d_{5,22}$	$d_{22,16}$	$d_{16,8}$	

$\mathrm{Re}\,move = d_{\pi_{D,m}\pi_{D,1}} = d_{\pi_{D,4}\pi_{D,1}} = d_{16,4}$

$Add = d_{\pi_{R,k}\pi_{D,1}} + d_{\pi_{D,m}\pi_{R,k}} = d_{8,14} + d_{16,8}$

$F(X) = F(X_D) + Add - \mathrm{Re}\,move$

$F(X) = d_{14,5} + d_{5,22} + d_{22,16} + d_{16,14} + d_{8,14} + d_{16,8} - d_{16,14}$

$F(X) = d_{14,5} + d_{5,22} + d_{22,16} + d_{8,14} + d_{16,8}$

B. Insertion of the node $\pi_{R,k}$ in the last slot in a particle

a. $\mathrm{Re}\,move = d_{\pi_{D,m}\pi_{D,1}}$ $Add = d_{\pi_{D,m}\pi_{R,k}} + d_{\pi_{R,k}\pi_{D,1}}$

b. $F(X) = F(X_D) + Add - \mathrm{Re}\,move$

Example B. Insertion of the node $\pi_{R,k} = 8$ in the last slot

j	1	2	3	4	5	1
x_j	14	5	22	16	8	14
n_j	3	1	5	4	2	3
$d_{\pi_j\pi_{j+1}}$	$d_{14,5}$	$d_{5,22}$	$d_{22,16}$	$d_{16,8}$	$d_{8,14}$	

$$\mathrm{Re}\,move = d_{\pi_{D,m}\pi_{D,1}} = d_{\pi_{D,4}\pi_{D,1}} = d_{16,14}$$

$$Add = d_{\pi_{D,m}\pi_{R,k}} + d_{\pi_{R,k}\pi_{D,1}} = d_{16,8} + d_{8,14}$$

$$F(X) = F(X_D) + Add - \mathrm{Re}\,move$$

$$F(X) = d_{14,5} + d_{5,22} + d_{22,16} + d_{16,14} + d_{16,8} + d_{8,14} - d_{16,14}$$

$$F(X) = d_{14,5} + d_{5,22} + d_{22,16} + d_{16,8} + d_{8,14}$$

Note that the insertion of the node $\pi_{R,k}$ into the first and last slot of a tour is equivalent to each other even though the tours are different.

C. Insertion of the node $\pi_{R,k}$ in between the edge $(\pi_{D,u}, \pi_{D,v})$, See Snyder and Daskin [16].

a. $\mathrm{Re}\,move = d_{\pi_{D,u}\pi_{D,v}}$ $Add = d_{\pi_{D,u}\pi_{R,k}} + d_{\pi_{R,k}\pi_{D,v}}$

b. $F(X) = F(X_D) + Add - \mathrm{Re}\,move$

Example C. Insertion of the node $\pi_{R,k} = 8$ in between $(\pi_{D,u}, \pi_{D,v}) = (14,5)$.

j	1	2	3	4	5	1
π_j	**14**	8	**5**	22	16	14
n_j	**3**	2	**1**	5	4	3
$d_{\pi_j\pi_{j+1}}$	$d_{14,8}$	$d_{8,5}$	$d_{5,22}$	$d_{22,16}$	$d_{16,14}$	

$$\mathrm{Re}\,move = d_{\pi_{D,u}\pi_{D,v}} = d_{14,5}$$

$$Add = d_{\pi_{D,u}\pi_{R,k}} + d_{\pi_{R,k}\pi_{D,v}} = d_{14,8} + d_{8,5}$$

$$F(X) = F(X_D) + Add - \mathrm{Re}\,move$$

$$F(X) = d_{14,5} + d_{5,22} + d_{22,16} + d_{16,14} + d_{14,8} + d_{8,5} - d_{14,5}$$

$$F(X) = d_{5,22} + d_{22,16} + d_{16,14} + d_{14,8} + d_{8,5}$$

2.4 VND Local Search

VND is a recent meta-heuristic proposed by Mladenovic & Hansen [21] systematically exploiting the idea of neighborhood change, both in descent to local minima and in escape from the valleys containing them. We apply the VND local search to the personal best P_i^t population at each generation t. For the GTSP, the following neighborhood structures were considered: $\Psi_1 = DC_2(P_i^t)$, $\Psi_2 = DC_3(P_i^t)$. The neighborhood $\psi_1 = DC_2(P_i^t)$ is basically concerned with removing a single node and cluster from the particle P_i^t, replacing that particular node with another node randomly chosen from the same cluster, and finally inserting the randomly chosen node into $m - d + 1$ slots of the particle P_i^t's tour with its corresponding cluster. It implies that $d = 1$ and $p = 1$. On the other hand, the neighborhood $\Psi_2 = DC_3(P_i^t)$ is basically related to first destructing the particle P_i^t with the size of $d = 2$, perturbing those nodes with the size of $p = 2$, and finally inserting the $\pi_{R,k}$ into $\pi_{D,j}$ in a greedy manner with the cheapest insertion method. It implies that two nodes are randomly selected and both of them perturbed with some other nodes from the same cluster.

The implementation sequence of the VND neighborhood structures is chosen as $\Psi_1 + \Psi_2$. The size of the VND local search was the number of cluster for each problem instance. The pseudo code of the VND local search is given in Figure 4.

```
Procedure VND
s0:=Pi
Choose ψh, h=1,..,hmax
While (Not Termination) Do
    h:=1
    While (h<hmax) Do
        s1:= ψh(s0)
        If f(s1)<f(s0) then
            s0:=s1
            h:=1
        Else
            h:=h+1
    Endwhile
Enwhile
    Update Pi and G
    Return Pi and G
End
```

Figure 4. VND Local Search

2.5 PTL Crossover

Two-cut PTL crossover operator presented in [17] is used to update the particles of the DPSO algorithm. Two-cut PTL crossover operator is able to produce a pair of distinct offspring even from two identical parents. An illustration of two-cut PTL crossover operator is shown in Figure 5.

	j	1	2	3	4	5
P1	π_j	24	3	**19**	**8**	14
	n_j	5	1	**4**	**2**	3
P2	π_j	24	3	**19**	**8**	14
	n_j	5	1	**4**	**2**	3
O1	π_j	**19**	**8**	24	3	14
	n_j	**4**	**2**	5	1	3
O2	π_j	24	3	14	**19**	**8**
	n_j	5	1	3	**4**	**2**

Figure 5. Two-Cut PTL Crossover Operator.

3. COMPUTATIONAL RESULTS

Fischetti et al. [11] developed a branch-and-cut algorithm to solve the symmetric GTSP. The benchmark set is derived by applying a partitioning method to 46 standard TSP instances from the TSPLIB library [23]. The benchmark set with optimal objective function values for each of the problems is obtained through a personal communication with Dr. Lawrence V. Snyder. We apply our DPSO algorithm to the same benchmark set except for the 10 problems to make a fair comparison with all the best performing algorithms in the literature. The benchmark set we tested contains between 51 and 442 nodes with Euclidean distances. The DPSO algorithm was coded in Borland C and run on an Intel Centrino Duo 1.83 GHz Laptop with 512MB memory.

Regarding the parameters of the DPSO, they were determined experimentally through inexpensive runs of a few instances collected from the benchmark set. Population size is taken as 100 and the maximum number of generation is set carefully to 100 generations to have very fair comparisons with the current literature. Destruction and construction probability, w, crossover probabilities, CR_1 and CR_2 are all taken as 1.0. Five runs (R=5) were carried out for each problem instance to report the statistics based on the percentage relative deviations as follows:

$$\Delta_{avg} = \sum_{i=1}^{R} \left(\frac{(F_i - OPT) \times 100}{OPT} \right) / R \qquad (8)$$

where F_i, OPT, and R were the fitness function value generated by the DPSO algorithm in each run, the optimal objective function value, and the number of runs, respectively. For convenience, $\Delta_{\min}$ and $\Delta_{\max}$ denote the minimum, and maximum percentage relative deviations from the optimal values, respectively. For the computational effort consideration, $t_{\min}$, $t_{\max}$ and t_{avg} denote the minimum, maximum, average CPU time in seconds to reach the best solution found so far during the run, i.e., the point of time that the global best solution does not change after that time.

The computational results for the benchmark set are given in Table 2 and 3. We first compare the performance of the DPSO algorithm to a very recent random key genetic algorithm developed by Snyder and Daskin [16]. From Table 2, the GA found optimal solutions in at least *one* of the five runs for *30* out of 36 problems tested whereas the DPSO algorithm was able to find optimal solutions in at least *one* of the five runs for *35* out of 36 problems tested. It is important to note that those five optimal solutions belong to the larger instances ranging from 299 to 442 nodes. The overall performance of hit ratio for the DPSO algorithm was 4.50 whereas it is 4.03 for the GA. It can be interpreted that the DPSO algorithm was able to find the 90 percent of the optimal solutions whereas the GA was only able to find 81 percent of the optimal solutions. In addition, it is worthwhile observing that the DPSO algorithm was able to solve the most difficult problems to optimality except for the instance 89PCB442. Again, from Table 2, the DSPO algorithm was superior to the GA algorithms in terms of percent deviations from the optimal solutions. All three statistics were lower than those generated by the GA. Especially in terms of worst case analysis, the DPSO algorithm generated solutions no worst than 2.05% above optimal .

The CPU time requirements are difficult to compare, however, we carefully set the maximum number of generation to 100 so that a fair comparison should be made. Snyder and Daskin [16] used a machine with PIV 3.2 GHz processor and 1.0 GB RAM memory. We feel that we employed a machine, which is faster than the one in Snyder and Daskin [16]. In fact, the mean CPU requirement of the DPSO algorithm was 2.62 seconds on overall average whereas GA needed 1.72 seconds. It is also important to note that the GA was not able to improve the results after 10 generations as reported in Snyder & Daskin [16] whereas the DPSO algorithm was able to improve the results in even further generations leading to the necessity of some more CPU times during its search process without getting trapped at the local minima. This is to say that especially, DPSO generated so much better results than the RKGA that its relatively higher CPU times can be tolerated.

We compare the DPSO algorithm to several other algorithms (four heuristics, one exact algorithms and one meta-heuristic) on the same TSPLIB problems. The first one is the GI^3 heuristic presented by Renaud & Boctor [14]; the second is the NN heuristic which is developed by Noon [3]; the heuristics called “FST-Lagr” and “FST-Root” are the Lagrangian procedure and the root procedure, as well as the branch and cut procedure (B&C) described in Fishetti *et al*. [11]. Note that B&C is an exact algorithm and provided the optimal solutions in Fishetti *et al.* [11]. Note that we do not report the CPU times of other heuristics except for the GA since the CPU time of the GA was comparable to them as indicated and analyzed in Snyder & Daskin [16].

Table 3 gives the comparison of the DPSO algorithm with the best performing algorithms in the literature. In Snyder & Daskin [16], the first trial of the five runs is taken for comparison purposes. However, we feel that taking the average values would be much more suitable since the GA and DPSO are stochastic algorithms for which the average performance is meaningful when compared to deterministic algorithms making a single run. For this reason, we report the average percentage relative deviations for the GA and DPSO algorithms for comparisons to the best performing algorithms in the literature.

In order to statistically test the performance of the DPSO algorithms with the best performing algorithms in the literature, a series of the paired *t*-test at the 95% significance level was carried out based on the results in Table 2.. In the paired t-test, $\mu_D = \mu_1 - \mu_2$ denotes the true average difference between the percentage relative deviations generated by two different algorithms, the null hypothesis is given by $H_0 : \mu_D = \mu_1 - \mu_2 = 0$ saying that there is no difference between the average percentage relative deviations generated by two algorithms compared. On the other hand, the alternative hypothesis is given by $H_1 : \mu_D = \mu_1 - \mu_2 \neq 0$ saying that there is a difference between the average percentage relative deviations generated by two algorithms compared. As a reminder, the null hypothesis is rejected if *p* values are less than 0.05. The paired t-test results are given in Table 3.

Table 3 indicates the poor performance of the GI^3, NN and FST-Lagr algorithms compared since the null hypothesis was rejected on the behalf of the GA and DPSO algorithms. It means that the differences were meaningful at the significance level of 0.95. When comparing the GA and DPSO with the FST-Root algorithm, the null hypothesis was failed to reject indicating that the differences were not meaningful and those three algorithms were equivalent. However, the null hypothesis was rejected on the behalf of the DPSO algorithm compared to GA indicating the differences were meaningful.

4. CONCLUSIONS

A DPSO is presented to solve the GTSP on a set of benchmark instances ranging from 51 to 442 nodes. The statistical analysis showed that the DPSO algorithm is one of the best performing algorithms together with the GA and FST-Root algorithms for the GTSP. Hence, the DPSO is promising in applying it to the other combinatorial optimization algorithms. The authors have already developed a discrete differential evolution (DDE) algorithm for the GTSP too. Detailed results of both DPSO and DDE algorithms will be presented in the literature in the near future.

Table 3. Paired t-test at Significance Level of 0.95

H_0	H_1	*t*	*p*	H_0
DPSO=GA	DPSO#GA	-2.11	0.042	R
GA=GI3	GA#GI3	-3.35	0.002	R
GA=NN	GA#NN	-3.53	0.001	R
GA=FST-Lagr	GA#FST-Lagr	-2.58	0.014	R
GA=FST-Root	GA#FST-Root	0.05	0.963	FR
DPSO=GI3	DPSO#GI3	-3.35	0.002	R
DPSO=NN	DPSO#NN	-3.69	0.001	R
DPSO=FST-Lagr	DPSO#FST-Lagr	-2.58	0.014	R
DPSO=FST-Root	DPSO#Root	-0.89	0.379	FR

R/FR=Reject/Fail to Reject

5. ACKNOWLEDGMENTS

We would like to thank to Dr. Lawrence Snyder for providing the benchmark suite. Dr P. N. Suganthan acknowledges the financial support offered by the A*Star (Agency for Science, Technology and Research) under the grant # 052 101 0020. Dr. Tasgetiren is grateful to Dr. Thomas Stutzle for his generosity in providing his IG code. Even though we developed our own code in C, it was substantially helpful in grasping the components of the IG algorithm in a great detail. We also appreciate his endless support and invaluable suggestions whenever needed.

6. REFERENCES

[1] 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.

[2] 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.

[3] C.E. Noon, The generalized traveling salesman problem, PhD thesis, University of Michigan, 1988.

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

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

[6] G. Laporte, A. Asef-Vaziri, C. Sriskandarajah, Some applications of the generalized travelling salesman problem, Journal of the Operational Research Society 47 (12) (1996) 1461–1467.

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

[8] 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.

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] Pan Q-K, Tasgetiren M. F, Liang Y-C, A Discrete Particle Swarm Optimization Algorithm for the No-Wait Flowshop Scheduling Problem with Makespan and Total Flowtime Criteria, Accepted to Bio-inspired metaheuristics for combinatorial optimization problems, Special issue of *Computers & Operations Research*, 2005.

[18] Jacobs, L.W., Brusco, M.J., 1995. A local search heuristic for large set-covering problems. Naval Research Logistics Quarterly 42 (7), 1129–1140.

[19] Marchiori, E., Steenbeek, A., 2000. An evolutionary algorithm for large set covering problems with applications to airline crew scheduling. In: Cagnoni, S. et al. (Eds.), Real-World Applications of Evolutionary Computing, EvoWorkshops 2000, Lecture Notes in Computer Science, vol. 1803. Springer-Verlag, Berlin, pp. 367–381.

[20] 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.

[21] Mladenovic N, Hansen P, Variable neighborhood search, Computers and Operations Research 24 (1997) 1097-1100.

[22] Kennedy J, Eberhart RC, Shi Y. Swarm intelligence, San Mateo, Morgan Kaufmann, CA, USA, 2001.

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

Table 2. Comparison of Results for GA and DPSO Algorithms

Problem	OPT	Random Key GA							DPSO with VND Local Search						
		n_{opt}	Δ_{avg}	$\Delta_{\min}$	$\Delta_{\max}$	t_{avg}	$t_{\min}$	$t_{\max}$	n_{opt}	Δ_{avg}	$\Delta_{\min}$	$\Delta_{\max}$	t_{avg}	$t_{\min}$	$t_{\max}$
11EIL51	174	5	0.00	0.00	0.00	0.20	0.10	0.30	5	0.00	0.00	0.00	0.03	0.01	0.05
14ST70	316	5	0.00	0.00	0.00	0.20	0.20	0.30	5	0.00	0.00	0.00	0.03	0.02	0.05
16EIL76	209	5	0.00	0.00	0.00	0.20	0.20	0.20	5	0.00	0.00	0.00	0.03	0.02	0.05
16PR76	64925	5	0.00	0.00	0.00	0.20	0.20	0.30	5	0.00	0.00	0.00	0.05	0.03	0.06
20KROA100	9711	5	0.00	0.00	0.00	0.40	0.30	0.50	5	0.00	0.00	0.00	0.09	0.05	0.11
20KROB100	10328	5	0.00	0.00	0.00	0.40	0.20	0.50	5	0.00	0.00	0.00	0.10	0.09	0.11
20KROC100	9554	5	0.00	0.00	0.00	0.30	0.20	0.40	5	0.00	0.00	0.00	0.12	0.09	0.14
20KROD100	9450	5	0.00	0.00	0.00	0.40	0.20	0.80	5	0.00	0.00	0.00	0.09	0.05	0.12
20KROE100	9523	5	0.00	0.00	0.00	0.60	0.30	0.80	5	0.00	0.00	0.00	0.12	0.09	0.16
20RAT99	497	5	0.00	0.00	0.00	0.50	0.30	0.70	5	0.00	0.00	0.00	0.08	0.06	0.11
20RD100	3650	5	0.00	0.00	0.00	0.50	0.30	1.00	5	0.00	0.00	0.00	0.11	0.05	0.17
21EIL101	249	5	0.00	0.00	0.00	0.40	0.20	0.50	5	0.00	0.00	0.00	0.08	0.06	0.12
21LIN105	8213	5	0.00	0.00	0.00	0.50	0.30	0.70	5	0.00	0.00	0.00	0.08	0.05	0.12
22PR107	27898	5	0.00	0.00	0.00	0.40	0.30	0.50	5	0.00	0.00	0.00	0.12	0.06	0.17
25PR124	36605	5	0.00	0.00	0.00	0.80	0.60	1.5	5	0.00	0.00	0.00	0.17	0.14	0.22
26BIER127	72418	5	0.00	0.00	0.00	0.40	0.40	0.50	5	0.00	0.00	0.00	0.20	0.11	0.28
28PR136	42570	5	0.00	0.00	0.00	0.50	0.30	0.70	5	0.00	0.00	0.00	0.26	0.19	0.33
29PR144	45886	5	0.00	0.00	0.00	1.00	0.30	2.10	5	0.00	0.00	0.00	0.29	0.19	0.41
30KROA150	11018	5	0.00	0.00	0.00	0.70	0.30	1.30	5	0.00	0.00	0.00	0.37	0.22	0.45
30KROB150	12196	5	0.00	0.00	0.00	0.90	0.30	1.20	5	0.00	0.00	0.00	0.35	0.26	0.52
31PR152	51576	5	0.00	0.00	0.00	1.20	0.90	1.50	5	0.00	0.00	0.00	0.71	0.42	0.98
32U159	22664	5	0.00	0.00	0.00	0.80	0.40	1.30	5	0.00	0.00	0.00	0.42	0.34	0.55
39RAT195	854	5	0.00	0.00	0.00	1.00	0.70	1.40	5	0.00	0.00	0.00	2.21	0.62	4.51
40D198	10557	5	0.00	0.00	0.00	1.60	1.10	2.70	5	0.00	0.00	0.00	1.22	0.62	1.98
40KROA200	13406	5	0.00	0.00	0.00	1.80	1.10	2.70	5	0.00	0.00	0.00	0.79	0.64	0.95
40KROB200	13111	4	0.00	0.00	0.02	1.90	1.40	2.90	**5**	0.00	0.00	0.00	2.70	0.95	5.77
45TS225	68340	**4**	0.02	0.00	0.09	2.10	1.40	2.60	3	0.04	0.00	0.09	1.42	0.70	2.88
46PR226	64007	5	0.00	0.00	0.00	1.50	0.80	2.40	5	0.00	0.00	0.00	0.46	0.45	0.47
53GIL262	1013	0	0.75	0.10	1.18	1.90	0.70	3.10	**3**	0.32	0.00	0.89	4.51	1.34	7.25
53PR264	29549	5	0.00	0.00	0.00	2.10	1.30	3.50	5	0.00	0.00	0.00	1.10	0.76	1.30
60PR299	22615	0	0.11	0.02	0.27	3.20	1.60	6.10	**3**	0.03	0.00	0.09	3.08	1.84	4.20
64LIN318	20765	2	0.62	0.00	1.26	3.50	2.40	4.90	**3**	0.46	0.00	1.38	8.49	2.98	13.30
80RD400	6361	0	1.19	0.86	1.37	5.90	3.50	8.90	**1**	0.91	0.00	1.97	13.55	7.80	21.05
84FL417	9651	0	0.05	0.03	0.07	5.30	2.40	8.60	**5**	0.00	0.00	0.00	6.74	5.03	9.44
88PR439	60099	0	0.27	0.00	0.65	9.50	5.30	12.90	**4**	0.00	0.00	0.01	20.87	13.22	30.69
89PCB442	21657	0	1.70	1.31	2.19	9.00	4.50	14.50	0	0.86	0.07	2.05	23.14	13.81	28.72
Mean		***4.03***	***0.13***	***0.06***	***0.20***	**1.72**	**0.97**	**2.63**	**4.50**	**0.07**	**0.00**	**0.18**	***2.62***	***1.48***	***3.83***

Table 3. Comparison of Results for Best Performing Algorithms

	GA		DPSO		GI3		NN		FST-Lagr		FST-Root		B&C
Problem	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ	t	Δ	t	Δ	t	Δ	t	t
11EIL51	0.00	0.20	0.00	0.03	0.00	0.30	0.00	0.40	0.00	0.40	0.00	2.90	2.90
14ST70	0.00	0.20	0.00	0.03	0.00	1.70	0.00	0.80	0.00	1.20	0.00	7.30	7.30
16EIL76	0.00	0.20	0.00	0.03	0.00	2.20	0.00	1.10	0.00	1.40	0.00	9.40	9.40
16PR76	0.00	0.20	0.00	0.05	0.00	2.50	0.00	1.90	0.00	0.60	0.00	12.9	12.90
20KROA100	0.00	0.40	0.00	0.09	0.00	6.80	0.00	3.80	0.00	2.40	0.00	18.30	18.40
20KROB100	0.00	0.40	0.00	0.10	0.00	6.4	0.00	2.40	0.00	3.10	0.00	22.10	22.20
20KROC100	0.00	0.30	0.00	0.12	0.00	6.50	0.00	6.30	0.00	2.20	0.00	14.30	14.40
20KROD100	0.00	0.40	0.00	0.09	0.00	8.60	0.00	5.60	0.00	2.50	0.00	14.20	14.30
20KROE100	0.00	0.60	0.00	0.12	0.00	6.70	0.00	2.80	0.00	0.90	0.00	12.90	13.00
20RAT99	0.00	0.50	0.00	0.08	0.00	5.00	0.00	7.30	0.00	3.10	0.00	51.4	51.5
20RD100	0.00	0.50	0.00	0.11	0.08	7.30	0.08	8.30	0.08	2.60	0.00	16.5	16.6
21EIL101	0.00	0.40	0.00	0.08	0.40	5.20	0.40	3.00	0.00	1.70	0.00	25.50	25.60
21LIN105	0.00	0.50	0.00	0.08	0.00	14.40	0.00	3.70	0.00	2.00	0.00	16.20	16.40
22PR107	0.00	0.40	0.00	0.12	0.00	8.70	0.00	5.20	0.00	2.10	0.00	7.30	7.40
25PR124	0.00	0.80	0.00	0.17	0.43	12.20	0.00	12.00	0.00	3.70	0.00	25.70	25.90
26BIER127	0.00	0.40	0.00	0.20	5.55	36.10	9.68	7.80	0.00	11.20	0.00	23.30	23.60
28PR136	0.00	0.50	0.00	0.26	1.28	12.5	5.54	9.60	0.82	7.20	0.00	42.80	43.00
29PR144	0.00	1.00	0.00	0.29	0.00	16.30	0.00	11.8	0.00	2.30	0.00	8.00	8.20
30KROA150	0.00	0.70	0.00	0.37	0.00	17.80	0.00	22.90	0.00	7.60	0.00	100.00	100.30
30KROB150	0.00	0.90	0.00	0.35	0.00	14.20	0.00	20.10	0.00	9.90	0.00	60.30	60.60
31PR152	0.00	1.20	0.00	0.71	0.47	17.60	1.80	10.30	0.00	9.60	0.00	51.40	94.80
32U159	0.00	0.80	0.00	0.42	2.60	18.50	2.79	26.50	0.00	10.90	0.00	139.60	146.40
39RAT195	0.00	1.00	0.00	2.21	0.00	37.2	1.29	86.00	1.87	8.20	0.00	245.50	245.90
40D198	0.00	1.60	0.00	1.22	0.60	60.40	0.60	118.80	0.48	12.00	0.00	762.50	763.10
40KROA200	0.00	1.80	0.00	0.79	0.00	29.70	5.25	53.00	0.00	15.30	0.00	183.30	187.40
40KROB200	0.00	1.90	0.00	2.70	0.00	35.80	0.00	135.20	0.05	19.10	0.00	268.00	268.50
45TS225	0.02	2.10	0.04	1.42	0.61	89.00	0.00	117.80	0.09	19.40	0.09	1298.40	37875.90
46PR226	0.00	1.50	0.00	0.46	0.00	25.50	2.17	67.60	0.00	14.60	0.00	106.20	106.90
53GIL262	0.75	1.90	0.32	4.51	5.03	115.40	1.88	122.7	3.75	15.80	0.89	1443.50	6624.10
53PR264	0.00	2.10	0.00	1.10	0.36	64.40	5.73	147.20	0.33	24.30	0.00	336.00	337.00
60PR299	0.11	3.20	0.03	3.08	2.23	90.30	2.01	281.80	0.00	33.20	0.00	811.40	812.80
64LIN318	0.62	3.50	0.46	8.49	4.59	206.80	4.92	317.00	0.36	52.50	0.36	847.80	1671.90
80RD400	1.19	5.90	0.91	13.55	1.23	403.50	3.98	1137.10	3.16	59.80	2.97	5031.50	7021.40
84FL417	0.05	5.30	0.00	6.74	0.48	427.10	1.07	1341.00	0.13	77.20	0.00	16714.40	16719.40
88PR439	0.27	9.50	0.00	20.87	3.52	611.00	4.02	1238.90	1.42	146.6	0.00	5418.90	5422.80
89PCB442	1.70	9.00	0.86	23.14	5.91	567.70	0.22	838.40	4.22	78.80	0.29	5353.90	58770.50
Mean	0.13	1.72	**0.07**	***2.62***	0.98	83.09	1.48	171.56	0.47	18.48	0.13	1097.32	3821.19