5
Smallest Position Value Approach

Fatih Tasgetiren[1], Angela Chen[2], Gunes Gencyilmaz[3], and Said Gattoufi[4]

[1] Department of Operations Management and Business Statistics, Sultan Qaboos University, Muscat, Sultanate of Oman
mfatih@squ.edu.om
[2] Department of Finance, Nanya Institute of Technology, Taiwan 320, R.O.C
achen@nanya.edu.tw
[3] Department of Management, Istanbul Kultur University, Istanbul, Turkey
g.gencyilmaz@iku.edu.tr
[4] Department of Operations Management and Business Statistics, Sultan Qaboos University, Muscat, Sultanate of Oman
gattoufi@squ.edu.om

Abstract. In a traveling salesman problem, if the set of nodes is divided into clusters for a single node from each cluster to be visited, then the problem is known as the generalized traveling salesman problem (GTSP). Such problem aims to find a tour with minimum cost passing through only a single node from each cluster. In attempt to show how a continuous optimization algorithm can be used to solve a discrete/combinatorial optimization problem, this chapter presents a standard continuous differential evolution algorithm along with a smallest position value (SPV) rule and a unique solution representation to solve the GTSP. The performance of the differential evolution algorithm is tested on a set of benchmark instances with symmetric distances ranging from 51 (11) to 442 (89) nodes (clusters) from the literature. Computational results are presented and compared to a random key genetic algorithm (RKGA) from the literature.

5.1 Introduction

The generalized traveling salesman problem (GTSP), one of several variations of the traveling salesman problem (TSP), has been originated from diverse real life or potential applications. The TSP finds a routing of a salesman who starts from an origin (i.e. a home location), visits a prescribed set of cities, and returns to the origin in such a way that the total distance is minimum and each city is travelled once. On the other hand, in the GTSP, a salesman when making a tour does not necessarily visit all nodes. But similar to the TSP, the salesman will try to find a minimum-cost tour and travel each city exactly once. Since the TSP in its generality represents a typical NP-Hard combinatorial optimization problem, the GTSP is also NP-hard. While many other combinatorial optimization problems can be reduced to the GTSP problem [11], applications of the GTSP spans over several areas of knowledge including computer science, engineering, electronics, mathematics, and operations research, etc. For example, publications can be found in postal routing [11], computer file processing [9], order picking in warehouses [17], process planning for rotational parts [3], and the routing of clients through welfare agencies [24].

Let us first define the GTSP, a complete graph $G = (V,E)$ is a weighted undirected whose edges are associated with non-negative costs. We denote the cost of an edge $e = (i,j)$ by c_{ij}. Then, the set of V nodes is divided into m sets or clusters such that $V = \{V_1,..,V_m\}$ with $V = \{V_1 \cup .. \cup V_m\}$ and $V_j \cap V_k = \phi$. The problem involves two related decisions- choosing a node from the subset and finding a minimum cost tour in the subgraph of G. In other words, the objective is to find a minimum tour length containing exactly single node from each cluster V_j.

The GTSP was first addressed in [9, 24, 28]. Applications of various exact algorithms can be found in Laporte et al. [12, 13], Laporte & Nobert [11], Fischetti et al. [7, 8], and others in [4, 18]. Laporte & Nobert [11], developed an exact algorithm for GTSP by formulating an integer programming and finding the shortest Hamiltonian cycle through some clusters of nodes. Noon and Bean [18], presented a Lagrangean relaxation algorithm. Fischetti et al. [8] dealt with the asymmetric version of the problem and developed a branch and cut algorithm to solve this problem. While exact algorithms are very important, they are unreliable with respect to their running time which can easily reach many hours or even days, depending on the problem sizes. Meanwhile several other researchers use transformations from GTSP to TSP since a large variety of exact and heuristic algorithms have been applied for the TSP [3],. Lien et. al. [15] first introduced transformation of a GTSP into a TSP, where the number of nodes of the transformed TSP was very large. Then Dimitrijevic and Saric [6] proposed another transformation to decrease the size of the corresponding TSP. However, many such transformations depend on whether or not the problem is symmetric; moreover, while the known transformations usually allow to produce optimal GTSP tours from the obtained optimal TSP tours, such transformations do not preserve suboptimal solutions. In addition, such conversions of near-optimal TSP tours may result in infeasible GTSP solutions.

Because of the multitude of inputs and the time needed to produce best results, the GTSP problems are harder and harder to solve. That is why, in such cases, applications of several worthy heuristic approaches to the GTSP are considered. The most used construction heuristic is the nearest-neighbor heuristic which, in its adaptation form, was presented in Noon [17]. Similar adaptations of the farthest-insertion, nearest-insertion, and cheapest-insertion heuristics are proposed in Fischetti et al. [8]. In addition, Renaud & Boctor [20] developed one of the most sophisticated heuristics, called GI^3 (Generalized Initilialization, Insertion, and Improvement), which is a gen-eralization of the I^3 heuristic in Renaud et al. [21]. GI^3 contains three phases: in the Initialization phase, the node close to the other clusters is chosen from each cluster and greedily built into a tour that passes through some, but not necessarily all, of the chosen nodes. Next in the Insertion phase, nodes from unvisited clusters are inserted between two consecutive clusters on the tour in the cheapest possible manner, allowing the visited node to change for the adjacent clusters; after each insertion, the heuristic performs a modification of the 3-opt improvement method. In the Improvement phase, modifications of 2-opt and 3-opt are used to improve the tour. Here the modifications, called G2-opt, G3-opt, and G-opt, allow the visited nodes from each cluster to change as the tour is being re-ordered by the 2-opt or 3-opt procedures.

Application of evolutionary algorithms specifically to the GTSP have been few in the literature until Snyder & Daskin [26] who proposed a random key genetic algorithm

(RKGA) to solve this problem. In their RKGA, a random key representation is used and solutions generated by the RKGA are improved by using two local search heuristics namely, 2-opt and ”swap”. In the search process, their ”swap” procedure is considered as a speed-up method which basically removes a node j from a tour and inserts all possible nodes ks from the corresponding cluster in between an edge (u,v) in a tour (i.e., between the node u and the node v). Such insertion is based on a modified nearest-neighbor criterion. These two local search heuristics have been separately embedded 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 (they use $p = 0.05$ in their implementation). These two improvement procedures in Snyder & Daskin [26] are implemented as follows:

1. If the new solution is worse than the pre-improvement cost of this individual, the *level-I improvement* is considered. That is, one 2-opt exchange and one ”swap” procedure (assuming a profitable one can be found) are performed and the resulting individual are stored.
2. Otherwise, the *level-II improvement* is considered. So the 2-opts are executed until no profitable 2-opts can be found, then the ”swap” procedures are carried out until no profitable swaps can be found. The procedure is repeated until no further 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 to the previous solutions than the others. Both *level-I* and *level-II* improvements consider a ”first-improvement” strategy, which means implementing the first improvement of a move, rather than the best improvement of such move.

Thereafter, Tasgetiren et al. [30, 31, 32] presented a discrete particle swarm optimization (**DPSO**) algorithm, a genetic algorithm (**GA**) and a hybrid iterated greedy (**HIG**) algorithm, respectively.They hybridized the above methods with a local search, called variable neighborhood descend algorithm, to further improve the solution quality; at the same time, they applied some speed-up methods for greedy node insertions. Silberholz & Golden proposed another **GA** in [25], which is denoted as **mrOXGA**.

Section 2 introduces a brief summary of discrete differential evolution algorithm. Section 3 provides the details of solution representation. Insertion methods are summarized in Section 4. Section 5 gives the details of the local search improvement heuristics. The computational results on benchmark instances are discussed in Section 6. Finally, Section 7 summarizes the concluding remarks.

5.2 Differential Evolution Algorithm

Differential evolution (DE) is a latest evolutionary optimization methods proposed by Storn & Price [27]. Like other evolutionary-type algorithms, DE is a population-based and stochastic global optimizer. The DE algorithm starts with establishing the initial population. Each individual has an **m**-dimensional vector with parameter values determined randomly and uniformly between predefined search ranges. In a DE algorithm, candidate solutions are represented by chromosomes based on floating-point numbers. In the mutation process of a DE algorithm, the weighted difference between two randomly selected population members is added to a third member to generate a mutated solution. Then, a crossover operator follows to combine the mutated solution with the target solution so as to generate a trial solution. Thereafter, a selection operator is applied to compare the fitness function value of both competing solutions, namely, target and trial solutions to determine who can survive for the next generation. Since DE was first introduced to solve the Chebychev polynomial fitting problem by Storn & Price [25], [27], it has been successfully applied in a variety of applications that can be found in Corne et. al [5], Lampinen [10], Babu & Onwubolu [1]; and Price et al. [19].

Currently, there are several variants of DE algorithms. We follow the *DE/rand/1/bin* scheme of Storn & Price [27] with the inclusion of SPV rule in the algorithm. Pseudocode of the DE algorithm is given in Fig 5.1.

```
Initialize parameters
Initialize the target population individuals
Find the tour of the target population individuals
Evaluate the target population individuals
Apply local search to the target population individuals (Optional)
Do{
        Obtain the mutant population individuals
        Obtain the trial population individuals
        Find the tour of trial population individuals
        Evaluate the trial population individuals
        Do selection between the target and trial population individuals
        Apply local searchto the target population individuals (Optional)
}While (Not Termination)
```

Fig. 5.1. DE Algorithm with Local Search

The basic elements of DE algorithm are summarized as follows:

Target individual: X_i^t denotes the i^{th} individual in the population at generation t and is defined as $X_i^t = \left[x_{i1}^t, x_{i2}^t, ..., x_{in}^t\right]$, where x_{ij}^t is the parameter value of the i^{th} individual with respect to the j^{th} dimension ($j = 1, 2, ..., n$).

Mutant individual: V_i^t denotes the i^{th} individual in the population at generation t and is defined as $V_i^t = \left[v_{i1}^t, v_{i2}^t, ..., v_{in}^t\right]$, where v_{ij}^t is the parameter value of the i^{th} individual with respect to the j^{th} dimension ($j = 1, 2, ..., n$).

Trial individual: U_i^t denotes the i^{th} individual in the population at generation t and is defined as $U_i^t = \left[u_{i1}^t, u_{i2}^t, ..., u_{in}^t\right]$, where u_{ij}^t is the parameter value of the i^{th} individual with respect to the j^{th} dimension ($j = 1, 2, ..., n$).

Target population: X^t is the set of NP individuals in the target population at generation t, i.e., $X^t = [X_1^t, X_2^t, ..., X_{NP}^t]$.

Mutant population: V^t is the set of NP individuals in the mutant population at generation t, i.e., $V^t = [V_1^t, V_2^t, ..., V_{NP}^t]$.

Trial population: U^t is the set of NP individuals in the trial population at generation t, i.e., $U^t = [U_1^t, U_2^t, ..., U_{NP}^t]$.

Tour: a newly introduced variable π_i^t, denoted a tour of the GTSP solution implied by the individual X_i^t, is represented as $\pi_i^t = \left[\pi_{i1}^t, \pi_{i2}^t, ..., \pi_{in}^t\right]$, where π_{ij}^t is the assignment of node j of the individual i in the tour at generation t.

Mutant constant: $F \in (0,2)$ is a real number constant which affects the differential variation between two individuals.

Crossover constant: $CR \in (0,1)$ is a real number constant which affects the diversity of population for the next generation.

Fitness function: In a minimization problem, the objective function is $f_i\left(\pi_i^t \leftarrow X_i^t\right)$, where π_i^t denotes the corresponding tour of individual X_i^t.

5.2.1 Solution Representation

In this section, we present a solution representation which enables **DEs** to solve the **GTSP**. Bean [2] suggested an encoding for the **GA** to solve the **GTSP**, where each set V_j has a gene consisting of an integer part between $\left[1, \left|V_j\right|\right]$ and a fractional part between $[0,1]$. The integer part indicates which node from the cluster is included in the tour, and the nodes are sorted by their fractional part to indicate the order. Similarly, a continuous **DE** can be used to solve the **GTSP**. First, we say each parameter value represents a cluster for the **GTSP** and is restricted to each cluster size of the GTSP instances.

From the following example, consider a GTSP instance with $V = \{1,..,20\}$ and $V_1 = \{1,..,5\}$, $V_2 = \{6,,..,10\}$, $V_3 = \{11,..,15\}$ and $V_4 = \{16,..,20\}$. The parameter values (x_j) can be positive or negative, e.g. for dimension j equal to 4.23 and for dimension j 2 is -3.07, etc. This feature indicates the difference between the random key encoding and the one in this chapter. Table 5.1 shows the solution representation of the DE for the GTSP. Then the integer parts of these parameter values (x_j) are respectively decoded as node 4 (the fourth node in V_1), node 8 (the third node in V_2), node 11 (the first node in V_3), and node 18 (the third node in V_1).

Table 5.1. Solution Representation

j	1	2	3	4
x_j	4.23	-3.07	1.80	3.76
v_j	4	8	11	18
s_j	0.23	-0.07	0.80	0.76
π_j	8	4	18	11
$F(\pi)$	$d_{8,4}$	$d_{4,18}$	$d_{18,11}$	$d_{11,8}$

Since the values of parameter x_j can be positive or negative, to determine which node (v_j) should be taken, the absolute value of the parameter x_j needs to be considered. Then the random key values (s_j) are determined by simply subtracting the integer part of the parameter x_j from its current value considering the negative signs, i.e., $s_j = x_j - \mathrm{int}(x_j)$. So for dimension 1, its parameter x_1 is equal to 4.23 and the random key value s_1 is 0.23 ($S_1 = 4.23 - 4$). Finally, with respect to the random key values (s_j), the smallest position value (SPV) rule of Tasgetiren et. al. [29] is applied to the random key vector to determine the tour π. As illustrated in Table 5.1, the objective function value implied by a solution x with m nodes is the total tour length, which is given by

$$F(\pi) = \sum_{j=1}^{m-1} d_{\pi_j \pi_{j+1}} + d_{\pi_m \pi_1} \tag{5.1}$$

However, with this proposed representation scheme, a problem may rise such that when the DE update equations are applied, any parameter value might be outside of the initial search range, which is restricted to the size of each cluster. Let $x_{\min}[j]$ and $x_{\max}[j]$ represent the minimum and maximum value of each parameter value for dimension j. Then they stand for the minimum and maximum cluster sizes of each dimension j. Regarding the initial population, each parameter value for the set V_j is drawn uniformly from $[-V_j + 1, V_j + 1]$. Obviously, $x_{\max}[j]$ is restricted to $[V_j + 1]$, whereas $x_{\min}[j]$ is restricted to $-x_{\max}[j]$. During the reproduction of the DE, when any parameter value is outside of the cluster size, it is randomly re-assigned to the corresponding cluster size again.

5.2.2 An Example Instance of the GTSP

In this section, we summarize the solution representation by using a GTSP instance of 11EIL51 from TSPLIB Library [23] with $V = \{1,..,51\}$, where the clusters are $V_1 = \{19,40,41\}$, $V_2 = \{3,20,35,36\}$, $V_3 = \{24,43\}$, $V_4 = \{33,39\}$, $V_5 = \{11,12,27,32,46,47,51\}$, $V_6 = \{2,16,21,29,34,50\}$, $V_7 = \{8,22,26,28,31\}$, $V_8 = \{13,14,18,25\}$,

Table 5.2. Clusters for the Instance 11EIL51

Cluster	Node						
V_1	19	40	41				
V_2	3	20	35	36			
V_3	24	43					
V_4	33	39					
V_5	11	12	27	32	46	47	51
V_6	2	16	21	29	34	50	
V_7	8	22	26	28	31		
V_8	13	14	18	25			
V_9	4	15	17	37	42	44	45
V_{10}	1	6	7	23	48		
V_{11}	5	9	10	30	38	49	

Table 5.3. Clusters for the Instance 11EIL51

Cluster	Node										
j	1	2	3	4	5	6	7	8	9	10	11
x_j	3.45	-2.66	1.86	1.11	-3.99	-6.24	-2.81	4.52	6.23	-1.89	3.02
v_j	41	20	24	33	27	50	22	25	44	1	10
s_j	0.45	-0.66	0.86	0.11	-0.99	-0.24	-0.81	0.52	0.23	-0.89	0.02
π_j	27	1	22	20	50	10	33	44	41	25	24
$d_{\pi_j\pi_{j+1}}$	$d_{27,1}$	$d_{1,22}$	$d_{22,20}$	$d_{20,50}$	$d_{50,10}$	$d_{10,33}$	$d_{33,44}$	$d_{44,41}$	$d_{41,25}$	$d_{25,24}$	$d_{24,27}$
$F(\pi)$	8	7	15	21	17	12	17	20	21	14	22

$V_9 = \{4,15,17,37,42,44,45\}$, $V_{10} = \{1,6,7,23,48\}$, and $V_{11} = \{5,9,10,30,38,49\}$. To make clearer, we show the 11EIL51 instance in Table 5.2 below:

In order to establish the GTSP solution, each parameter value for the dimension j is restricted to each cluster size such that $-4 < x_1 < 4$, $-5 < x_2 < 5$, $-3 < x_3 < 3$, $-3 < x_4 < 3$, $-8 < x_5 < 8$, $-7 < x_6 < 7$, $-6 < x_7 < 6$, $-5 < x_8 < 5$, $-8 < x_9 < 8$, $-6 < x_{10} < 6$ and $-7 < x_{11} < 7$. This provides the feasibility of the GTSP solution generated by the DE algorithm. Suppose that a DE solution is obtained by the traditional update equations and the parameter values $x_j's$ of the individual are given as in Table 5.3.

Similar to what we have explained via Table 5.1 example, the integer parts of the individual parameter values (x_j) are respectively decoded as node 41 (the third node in V_1), node 20 (the second node in V_2), node 24 (the first node in V_3), node 33 (the first node in V_4), node 27 (the third node in V_5), node 50 (the sixth node in V_6), node 22 (the second node in V_7), node 25 (the fourth node in V_8), node 44 (the sixth node in V_9), node 1 (the first node in V_{10}) and node 10 (the third node in V_{11}). Unlike the case in the RKGA, where the random key is defined as another vector, the fractional part of the individual parameter values (x_j) can be directly obtained as a random key to obtain the tour. As shown in Table 5.3, while applying the SPV rule to the random key vector (s_j), the tour (π_j) can be obtained very easily. As well, the objective function value of the individual X is given by

$$F(\pi) = \sum_{j=1}^{10} d_{\pi_j\pi_{j+1}} + d_{\pi_{10}\pi_1} = d_{27,1} + d_{1,22} + d_{22,20} + d_{20,50} + d_{50,10} + d_{10,33} + d_{33,44} + d_{44,41} + d_{41,25} + d_{25,24} + d_{24,27}$$

$$F(\pi) = \sum_{j=1}^{10} d_{\pi_j\pi_{j+1}} + d_{\pi_{11}\pi_1} = 8+7+15+21+17+12+17+20+21+14+22 = 174$$

5.2.3 Complete Computational Procedure of DE

The complete computational procedure of the DE algorithm for the GTSP problem can be summarized as follows:

- **Step 1: Initialization**
  - Set $t = 0$, $NP = 100$
  - Generate NP individuals randomly as in Table 5.1, $\{X_i^0, i = 1,2,...,NP\}$ where $X_i^0 = [x_{i1}^0, x_{i2}^0, ..., x_{in}^0]$.
  - Apply the SPV rule to find the tour $\pi_i^0 = [\pi_{i1}^0, \pi_{i2}^0, ..., \pi_{in}^0]$ of individual X_i^0 for $i = 1,2,...,NP$.
  - Evaluate each individual i in the population using the objective function f_i^0 $(\pi_i^o \leftarrow X_i^0)$ for $i = 1,2,...,NP$.
- **Step 2: Update generation counter**
  - $t = t + 1$
- **Step 3: Generate mutant population**
  - For each target individual, X_i^t, $i = 1,2,...,NP$, at generation t, a mutant individual, $V_i^t = [v_{i1}^t, v_{i2}^t, ..., v_{in}^t]$, is determined such that:

$$V_i^t = X_{a_i}^{t-1} + F\left(X_{b_i}^{t-1} - X_{c_i}^{t-1}\right) \tag{5.2}$$

  where a_i, b_i and c_i are three randomly chosen individuals from the population such that $(a_i \neq b_i \neq c_i)$.
- **Step 4: Generate trial population**
  - Following the mutation phase, the crossover (re-combination) operator is applied to obtain the trial population. For each mutant individual, $V_i^t = [v_{i1}^t, v_{i2}^t, ..., v_{in}^t]$, an integer random number between 1 and n, i.e., $D_i \in (1,2,...,n)$, is chosen, and a trial individual, $U_i^t = [u_{i1}^t, u_{i2}^t, ..., u_{in}^t]$ is generated such that:

$$u_{ij}^t = \begin{cases} v_{ij}^t,\ if\ r_{ij}^t \leq CR\ or\ j = D_i \\ x_{ij}^{t-1},\quad Otherwise \end{cases} \tag{5.3}$$

  where the index D refers to a randomly chosen dimension $(j = 1,2,...,n)$, which is used to ensure that at least one parameter of each trial individual U_i^t differs from its counterpart in the previous generation U_i^{t-1}, CR is a user-defined crossover constant in the range (0, 1), and r_{ij}^t is a uniform random number between 0 and 1. In other words, the trial individual is made up with some parameters of mutant individual, or at least one of the parameters randomly selected, and some other parameters of target individual.
- **Step 5: Find tour**
  - Apply the SPV rule to find the tour $\pi_i^t = [\pi_{i1}^t, \pi_{i2}^t, ..., \pi_{in}^t]$ for $i = 1,2,...,NP$.
- **Step 6: Evaluate trial population**
  - Evaluate the trial population using the objective function $f_i^t\ (\pi_i^t \leftarrow U_i^t)$ for $i = 1,2,...,NP$.
- **Step 7: Selection**
  - To decide whether or not the trial individual U_i^t should be a member of the target population for the next generation, it is compared to its counterpart target individual X_i^{t-1} at the previous generation. The selection is based on the survival of fitness among the trial population and target population such that:

$$X_i^t = \begin{cases} U_i^t, & if\ f(\pi_i^t \leftarrow U_i^t) \leq f\left(\pi_i^{t-1} \leftarrow X_i^{t-1}\right) \\ X_i^{t-1}, & otherwise \end{cases} \tag{5.4}$$

- **Step 8: Stopping criterion**
  - If the number of generations exceeds the maximum number of generations, or some other termination criterion, then stop; otherwise go to step 2.

5.3 Insertion Methods

In this section of the chapter, the insertion methods denoted as **LocalSearchSD()** are modified from the literature and facilitate the use of the local search. Insertion methods are based on the insertion of node π_k^R into $m+1$ possible positions of a partial or destructed tour π^D with m nodes and an objective function value of $F\left(\pi^D\right)$. Note that as an example, only a single node is considered to be removed from the current solution to establish π_k^R with a single node and re-inserted into the partial solution. Such insertion of node π_k^R into $m-1$ possible positions is actually proposed by Rosenkrantz et al. [22] for the TSP. Snyder & Daskin [26] 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. Suppose that node π_k^R=27 will be inserted in a partial tour in Table 5.4.

Table 5.4. Partial Solution to Be Inserted for the Instance 11EIL51

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

A Insertion of node π_k^R in the first position of the partial tour π^D

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

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

b $F(\pi) = F\left(\pi^D\right) + Add - Remove$, where $F(\pi)$ and $F\left(\pi^D\right)$ are fitness function values of the tour after insertion and the partial tour, respectively.

Example A:

$Remove = d_{\pi_m^D \pi_1^D}$
$Remove = d_{\pi_{10}^D \pi_1^D}$
$Remove = d_{24,1}$
$Add = d_{\pi_k^R \pi_1^D} + d_{\pi_m^D \pi_k^R}$
$Add = d_{\pi_1^R \pi_1^D} + d_{\pi_{10}^D \pi_1^R}$
$Add = d_{27,1} + d_{24,27}$
$F(\pi) = F\left(\pi^D\right) + Add - Remove$

Table 5.5. Insertion of node π_k^R=27 into the first position of partial solution for Case A

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

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

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

B Insertion of node π_k^R in the first position of the partial tour π^D

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

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

b $F(\pi) = F\left(\pi^D\right) + Add - Remove$, where $F(\pi)$ and $F\left(\pi^D\right)$ are fitness function values of the tour after insertion and the partial tour, respectively.

Example B:

$Remove = d_{\pi_m^D \pi_1^D}$
$Remove = d_{\pi_{10}^D \pi_1^D}$
$Remove = d_{24,1}$
$Add = d_{\pi_m^D \pi_k^R} + d_{\pi_k^R \pi_1^D}$
$Add = d_{\pi_{10}^D \pi_1^R} + d_{\pi_1^R \pi_1^D}$
$Add = d_{24,27} + d_{27,1}$
$F(\pi) = F\left(\pi^D\right) + Add - Remove$

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

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

C Insertion of node π_k^R between an edge $\left(\pi_u^D, \pi_v^D\right)$

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

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

b $F(\pi) = F\left(\pi^D\right) + Add - Remove$, where $F(\pi)$ and $F\left(\pi^D\right)$ are fitness function values of the tour after insertion and the partial tour, respectively.

Table 5.6. Insertion of node π_k^R=27 into the last position of partial solution for Case B

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

Example C:

$u = 6$
$v = 7$
$Remove = d_{\pi_u^D \pi_v^D}$
$Remove = d_{\pi_6^D \pi_7^D}$
$Remove = d_{33,44}$
$Add = d_{\pi_u^D \pi_k^R} + d_{\pi_k^R \pi_v^D}$
$Add = d_{\pi_6^D \pi_1^R} + d_{\pi_1^R \pi_7^D}$
$Add = d_{33,27} + d_{27,44}$
$F(\pi) = F\left(\pi^D\right) + Add - Remove$

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

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

Table 5.7. Insertion of node π_k^R between an edge $\left(\pi_u^D, \pi_v^D\right)$ for Case C

j	1	2	3	4	5	6	7	8	9	10	11
π_j^D	1	22	20	50	10	33	27	44	41	25	24
$d_{\pi_j \pi_{j+1}}$	$d_{1,22}$	$d_{22,50}$	$d_{20,50}$	$d_{50,10}$	$d_{10,33}$	$d_{33,27}$	$d_{27,44}$	$d_{44,41}$	$d_{41,25}$	$d_{25,24}$	$d_{24,1}$
230	7	15	21	17	12	41	33	20	21	14	29

Note that **Case B** can actually be managed by **Case C**, since the tour is cyclic. Note again that the above insertion approach is somewhat different than the one in Snyder & Daskin [26], where the cost of an insertion of node π_k^R in an edge $\left(\pi_u^D, \pi_v^D\right)$.

5.3.1 Hybridization with Local Search

The hybridization of DE algorithm with local search heuristics is achieved by performing a local search phase on every trial individual generated. The SWAP procedure [26], denoted as **LocalSearchSD** in this chapter, and the **2-opt** heuristic [21] were

```
procedure  $LS(\pi)$
    $h := 1$
    while  $(h \leq m)$ do
        $\pi^* := LocalSearchSD(\pi)$
        if  $(f(\pi^*) \leq f(\pi))$  then
            $\pi := \pi^*$
            $h := 1$
        else
            $h := h + 1$
        else
    endwhile
    return  $\pi$
end procedure
```

Fig. 5.2. The Local Search Scheme

```
Procedure  $SWAP()$
remove  $j$  from  $T$
for each  $k \in V_j$
    $c^k \leftarrow \min\{d_{uk} + d_{kv} - d_{uv} / (u,v)$ is an edge in $T\}$
    $k^* \leftarrow \arg\min_{k \in V_j} \{c_k\}$
insert  $k^*$  into T between (u, v)
```

Fig. 5.3. The SWAP Procedure

separately applied to each trial individual. The **2-opt** heuristic finds two edges of a tour that can be removed and two edges that can be inserted in order to generate a new tour with a lower cost. More specifically, in the **2-opt** heuristic, the neighborhood of a tour is obtained as the set of all tours that can be replaced by changing two nonadjacent edges in that tour. Note that the **2-opt** heuristic is employed with the first improvement strategy in this study. The pseudo code of the local search (**LS**) procedures is given in Fig 5.2.

As to the **LocalSearchSD** procedure, it is based on the **SWAP** procedure and is basically concerned with removing a node from a cluster and inserting a different node from that cluster into the tour. The insertion is conducted using a modified nearest-neighbour criterion, so that the new node may be inserted on the tour in a spot different. Each node in the cluster is inserted into all possible spots in the current solution and the best insertion is replaced with the current solution. The **SWAP** procedure of Snyder & Daskin [26] is outlined in Fig 5.3, whereas the proposed DE algorithm is given in Fig 5.4. Note that in **SWAP** procedure, the followings are given such that tour T; set V_j; node $j \in V_j$, $j \in T$; distances d_{uv} between each $u, v \in V$.

5.4 Computational Results

Fischetti et al. [8] developed a branch-and-cut algorithm to solve the symmetric GTSP. The benchmark set is derived by applying a partitioning method to standard TSP

```
$Procedure\ DE\_GTSP$
$Set\ CR, F, NP, TerCriterion$
$X = (x_1^0, x_2^0, .., x_{NP}^0)$
$\underset{i:=1,2,..,NP}{f(\pi_i^0 \leftarrow x_i^0)}$
$\pi_i^0 \leftarrow x_i^0 = \underset{i=1,2,..,NP}{2\_opt(\pi_i^0 \leftarrow x_i^0)}$
$\pi_i^0 \leftarrow x_i^0 = \underset{i=1,2,..,NP}{LS(\pi_i^0 \leftarrow x_i^0)}$
$\pi_g^0 \leftarrow x_i^0 = \arg\min \underset{i=1,2,..,NP}{\{f(\pi_i^0 \leftarrow x_i^0)\}}$
$\pi_B := \pi_g^0 \leftarrow x_i^0$
$k := 1$
$while\ (Not\ TerCriterion)\ do$
    $\underset{i:=1,2,..,NP, j=1,2,..,m}{v_{ij}^k := x_{ia}^k + F(x_{ib}^k + x_{ic}^k)}$
    $u_{ij}^k = \underset{i=1,2,..,NP, j=1,2,..,m}{\begin{cases} v_{ij}^k\ if\ r_{ij}^k < CR\ or\ j = D_j \\ x_{ij}^{k-1}\ otherwise \end{cases}}$
    $\underset{i:=1,2,..,NP}{f(\pi_i^k \leftarrow u_i^k)}$
    $\pi_i^k \leftarrow u_i^k = \underset{i=1,2,..,NP}{2\_opt(\pi_i^k \leftarrow u_i^k)}$
    $\pi_i^k \leftarrow u_i^k = \underset{i=1,2,..,NP}{LS(\pi_i^k \leftarrow u_i^k)}$
    $x_i^k = \underset{i=1,2,..,NP}{\begin{cases} u_i^k\ if\ f(\pi_i^k \leftarrow u_i^k) < f(\pi_i^{k-1} \leftarrow x_i^{k-1}) \\ x_i^{k-1}\ otherwise \end{cases}}$
    $\pi_g^k \leftarrow x_g^k = \arg\min \underset{i:=1,2,..,NP}{\{f(\pi_i^k \leftarrow x_i^k), f(\pi_g^{k-1} \leftarrow x_g^{k-1})\}}$
    $\pi_B = \arg\min \{f(\pi_B), f(\pi_g^k \leftarrow x_g^k)\}$
    $k := k + 1$
$endwhile$
$return\ \pi_B$
```

Fig. 5.4. The DE Algorithm with Local Search Heuristics

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. The benchmark set contains between 51 (11) and 442 (89) nodes (clusters) with Euclidean distances and the optimal objective function value for each of the problems is available. The DE algorithm was coded in Visual C++ and run on an Intel Centrino Duo 1.83 GHz Laptop with 512MB memory.

We consider the RKGA by Snyder & Daskin [26] for comparison in this paper due to the similarity in solution representation. The population size is taken as 100. Cross-over and mutation probability are taken as 0.9 and 0.2, respectively. To be consistent with Snyder & Daskin [26], the algorithm is terminated when 100 generations have been carried out or when 10 consecutive generations have failed to improve the best-known

Table 5.8. Computational Results of DE and RKGA Implementations

	DE		RKGA	
	F_{avg}	Δ_{avg}	F_{avg}	Δ_{avg}
11EIL51	219.4	26.1	227.4	30.7
14ST70	473.8	49.9	450.8	42.7
16EIL76	358.8	71.7	352	68.4
16PR76	93586.2	44.1	85385.2	31.5
20KROA100	20663	112.8	20191	107.9
20KROB100	20764.2	101	18537.4	79.5
20KROC100	20597.2	115.6	17871.6	87.1
20KROD100	19730.2	108.8	18477	95.5
20KROE100	20409.2	114.3	19787.6	107.8
20RAT99	1049	111.1	1090	119.3
20RD100	7349.2	101.3	7353.4	101.5
21EIL101	530.8	113.2	526.4	111.4
21LIN105	16170.2	96.9	14559.4	77.3
22PR107	64129.8	129.9	57724.6	106.9
25PR124	91609.4	150.3	82713	126
26BIER127	146725.2	102.6	154703.2	113.6
28PR136	115003.4	170.2	112674.6	164.7
29PR144	112725.6	145.7	94969.2	107
30KROA150	34961.8	217.3	31199.2	183.2
30KROB150	35184.8	188.5	34685.2	184.4
31PR152	140603.6	172.6	118813.4	130.4
32U159	61456.6	171.2	59099.2	160.8
39RAT195	3332	290.2	2844.2	233
40D198	30688.6	190.7	26453	150.6
40KROA200	49109.6	266.3	46866.4	249.6
40KROB200	48553.2	270.3	47303.2	260.8
45TS225	237888.4	248.1	229495.2	235.8
46PR226	259453.2	305.4	263699	312
53GIL262	4497	343.9	4233.6	314.8
53PR264	165646.6	460.6	145789.4	393.4
60PR299	116716.2	416.1	110977.8	390.2
64LIN318	98943.8	376.5	94469.2	352.1
80RD400	37058.6	482.6	34502.2	436.1
84FL417	68102	605.6	65025.6	573.5
88PR439	365437.8	508.1	364282.4	504.5
89PCB442	132388	511.3	131711.8	498

solution. Five runs were carried out for each problem instance to report the statistics based on the relative percent deviations (Δ) from optimal solutions as follows:

$$\Delta_{avg} = \sum_{i=1}^{R} \left(\frac{(H_i - OPT) \times 100}{OPT} \right) / R \tag{5.5}$$

Table 5.9. Comparison for Optimal Instances of DE and RKGA Implementations

	DE		RKGA	
Instance	F_{avg}	Δ_{avg}	F_{avg}	Δ_{avg}
11EIL51	0	0.08	0	0.2
14ST70	0	0.1	0	0.2
16EIL76	0	0.12	0	0.2
16PR76	0	0.14	0	0.4
20KROA100	0	0.21	0	0.4
20KROB100	0	0.22	0	0.3
20KROC100	0	0.2	0	0.4
20KROD100	0	0.21	0	0.6
20KROE100	0	0.2	0	0.5
20RAT99	0	0.2	0	0.5
20RD100	0	0.2	0	0.4
21EIL101	0	0.19	0	0.5
21LIN105	0	0.21	0	0.4
22PR107	0	0.23	0	0.8
25PR124	0	0.28	0	0.4
26BIER127	0	0.33	0	0.5
28PR136	0	1.27	0	1
29PR144	0	0.37	0	0.7
30KROA150	0	0.48	0	0.9
30KROB150	0	0.46	0	1.2
31PR152	0.01	1.49	0	0.8
32U159	0	0.55	0	1
39RAT195	0.07	4.6	0	1.6
40D198	0.04	3.54	0	1.8
40KROA200	0	1.81	0	1.9
40KROB200	0.04	2.03	0	2.1
45TS225	0.25	2.98	0.02	1.5
46PR226	0	0.76	0	1.9
53GIL262	1.24	5.65	0.75	2.1
53PR264	0.01	4.38	0	3.2
60PR299	0.71	10.4	0.11	3.5
64LIN318	0.77	8.89	0.62	5.9
80RD400	1.64	18.89	1.19	5.3
84FL417	0.09	25.26	0.05	9.5
88PR439	1.13	22.94	0.27	9
89PCB442	1.78	12.12	1.7	1.72
Avg	0.22	3.67	0.13	0.2

where H_i, OPT and R are the objective function values generated by the DE in each run, the optimal objective function value, and the number of runs, respectively. 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. F_{avg} represents the average objective function values out of five runs.

Table 5.8 shows the computational results of implementing DE without the local search methods and those adopted from Snyder & Daskin [26]. As seen in Table 5.8, the DE results are very competitive to the RKGA of Snyder & Daskin [26], even though a two-sided paired t-test favors the RKGA. However, our objective is just to show how a continuous optimization algorithm can be used for solving a combinatorial optimization problem. We would like to point out that with some better parameter tuning, the DE results could be further improved. In addition, our observation reveals the fact that the performance of the DE algorithm is tremendously affected by the mutation equation [14]. After applying the mutation operator, most dimension values fall outside of search limits (cluster sizes). To force them to be in the search range, they are randomly re-initialized between the search bounds in order to keep the DE algorithm search for nodes from clusters predefined. However, the random re-initialization causes the DE algorithm to conduct a random search, which ruins its learning ability. Based on our observation, using some different levels of crossover and mutation probabilities as well as other mutation operators did not have so much positive effect in the solution quality.

In spite of all the disadvantages above, the inclusion of local search improvement heuristics in Snyder & Daskin [26] has led the DE algorithm to be somehow competitive to the RKGA. The computational results with the local search heuristics are presented in Table 5.9.

As seen in Table 5.9, the DE algorithm with the local search improvement heuristics was able to generate competitive results to the RKGA of Snyder & Daskin [26]. However, as seen in both Table 5.8 and 5.9, the success was mainly due to the use of the local search improvement heuristics. A two-sided paired t-test on the relative percent deviations in Table 5.9 confirms that both DE and RKGA were statistically equivalent, since the p-value was 0.014. However, DE was computationally more expensive than RKGA.

5.5 Conclusions

A continuous DE algorithm is presented to solve the GTSP on a set of benchmark instances ranging from 51 (11) to 442 (89) nodes (clusters). The main contribution of this chapter is due to use of a continuous DE algorithm to solve a combinatorial optimization problem. For this reason, a unique solution representation is presented and the SPV rule is used to determine the tour. The pure DE algorithm without local search heuristics is competitive to RKGA. However, inclusion of the local search heuristics led the DE algorithm to be very competitive to the RKGA of Snyder & Daskin [26].

As we mentioned before, with some better parameter tuning, the DE results could have been further improved. However, our observation reveals the fact that the performance of the DE algorithm is tremendously affected by the mutation equation [14]. After applying the mutation operator, most parameter values fall outside of search limits (cluster sizes). To force them to be in the search range, they are randomly re-initialized between the search bounds in order to keep the DE algorithm search for nodes from clusters predefined. However, the random re-initialization causes the DE algorithm to conduct a random search, which ruins its learning ability. Based on our observation, using some different levels of crossover and mutation probabilities as well as other mutation operators did not have so much positive impact on the solution quality. In spite of all the disadvantages above, this work clearly shows the applicability of a continuous algorithm to solve a combinatorial optimization problem. .

For the future work, the current DE algorithm can be extended to solve some other combinatorial/discrete optimization problems based on clusters such as resource constrained project scheduling (mode selection), generalized assignment problem (agent selection), and so on. It will be also interesting to use the same representation for the particle swarm optimization and harmony search algorithms to solve the GTSP.

References

1. Babu, B., Onwubolu, G.: New Optimization Techniques in Engineering. Springer, Germany (2004)
2. Bean, J.: Genetic algorithms and random keys for sequencing and optimization. ORSA, Journal on Computing 6, 154–160 (1994)
3. Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A., Zverovitch, A.: Process planning for rotational parts using the generalized traveling salesman problem. Int. J. Prod. Res. 41(11), 2581–2596 (2003)
4. Chentsov, A., Korotayeva, L.: The dynamic programming method in the generalized traveling salesman problem. Math. Comput. Model. 25(1), 93–105 (1997)
5. Corne, D., Dorigo, M., Glover, F.: Differential Evolution, Part Two. In: New Ideas in Optimization, pp. 77–158. McGraw-Hill, New York (1999)
6. Dimitrijevic, V., Saric, Z.: Efficient Transformation of the Generalized Traveling Salesman Problem into the Traveling Salesman Problem on Digraphs. Information Science 102, 65–110 (1997)
7. Fischetti, M., Salazar-Gonzalez, J., Toth, P.: The symmetrical generalized traveling salesman polytope. Networks 26(2), 113–123 (1995)
8. Fischetti, M., Salazar-Gonzalez, J., Toth, P.: A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Oper. Res. 45(3), 378–394 (1997)
9. Henry-Labordere, A.: The record balancing problem–A dynamic programming solution of a generalized traveling salesman problem. Revue Francaise D Informatique DeRecherche Operationnelle 3(NB2), 43–49 (1969)
10. Lampinen, J.: A Bibliography of Differential Evolution Algorithm, Technical Report, Lappeenranta University of Technology, Department of Information Technology. Laboratory of Information Processing (2000)
11. Laporte, G., Nobert, Y.: Generalized traveling salesman problem through n-sets of nodes - An integer programming approach. INFOR 21(1), 61–75 (1983)
12. Laporte, G., Mercure, H., Nobert, Y.: Finding the shortest Hamiltonian circuit through n clusters: A Lagrangian approach. Congressus Numerantium 48, 277–290 (1985)
13. Laporte, G., Mercure, H., Nobert, Y.: Generalized traveling salesman problem through n - sets of nodes - The asymmetrical case. Discrete Appl. Math. 18(2), 185–197 (1987)
14. Laporte, G., Asef-Vaziri, A., Sriskandarajah, C.: Some applications of the generalized traveling salesman problem. J. Oper. Res. Soc. 47(12), 461–1467 (1996)
15. Lien, Y., Ma, E., Wah, B.: Transformation of the Generalized Traveling Salesman Problem into the Standard Traveling Salesman Problem. Information Science 64, 177–189 (1993)
16. Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. Oper. Res. 21, 498–516 (1973)

17. Noon, C.: The generalized traveling salesman problem, Ph.D. thesis. University of Michigan (1988)
18. Noon, C., Bean, J.: A Lagrangian based approach for the asymmetric generalized traveling salesman problem. Oper. Res. 39(4), 623–632 (1991)
19. Price, K., Storn, R., Lapinen, J.: Differential Evolution - A Practical Approach to Global Optimization. Springer, Heidelberg (2006)
20. Renaud, J., Boctor, F.: An efficient composite heuristic for the symmetric generalized traveling salesman problem. Eur. J. OPer. Res. 108(3), 571–584 (1998)
21. Renaud, J., Boctor, F., Laporte, G.: A fast composite heuristic for the symmetric traveling salesman problem. INFORMS Journal on Computing 4, 134–143 (1996)
22. Rosenkrantz, D., Stearns, R., Lewis, P.: Approximate algorithms for the traveling salesman problem. In: Proceedings of the 15th annual symposium of switching and automata theory, pp. 33–42 (1974)
23. Reinelt, G.: TSPLIB. A travelling salesman problem library. ORSA Journal on Computing 4, 134–143 (1996)
24. Saskena, J.: Mathematical model of scheduling clients through welfare agencies. Journal of the Canadian Operational Research Society 8, 185–200 (1970)
25. Silberholz, J., Golden, B.: 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, pp. 165–181. Springer, Heidelberg (1997)
26. Snyder, L., Daskin, M.: A random-key genetic algorithm for the generalized traveling salesman problem. Eur. J. Oper. Res. 174, 38–53 (2006)
27. Storn, R., Price, K.: Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. ICSI, Technical Report TR-95-012 (1995)
28. Srivastava, S., Kumar, S., Garg, R., Sen, R.: Generalized traveling salesman problem through n sets of nodes. Journal of the Canadian Operational Research Society 7, 97–101 (1970)
29. Tasgetiren, M., Sevkli, M., Liang, Y.-C., Gencyilmaz, G.: Particle Swarm Optimization Algorithm for the Single Machine Total Weighted Tardiness Problem, In: The Proceeding of the World Congress on Evolutionary Computation, CEC 2004, pp. 1412–1419 (2004)
30. Tasgetiren, M., Suganthan, P., Pan, Q.-K.: 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 (GECCO 2007), London UK, pp. 158–167 (2007)
31. Tasgetiren, M., Suganthan, P., Pan, Q.-K., Liang, Y.-C.: A genetic algorithm for the generalized traveling salesman problem. In: The Proceeding of the World Congress on Evolutionary Computation (CEC 2007), Singapore, pp. 2382–2389 (2007)
32. Tasgetiren, M., Pan, Q.-K., Suganthan, P., Chen, A.: A hybrid iterated greedy algorithm for the generalized traveling salesman problem. Computers and Industrial Engineering (submitted, 2008)