6

Discrete/Binary Approach

Fatih Tasgetiren[1], Yun-Chia Liang[2], Quan-Ke Pan[3], and Ponnuthurai Suganthan[4]

[1] Department of Operations Management and Business Statistics, Sultan Qaboos University, Muscat, Sultanate of Oman
mfatih@squ.edu.om
[2] Department of Industrial Engineering and Management, Yuan Ze University, Taiwan
ycliang@saturn.yzu.edu.tw
[3] College of Computer Science, Liaocheng University, Liaocheng, P.R. China
qkpan@lctu.edu.cn
[4] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798
epnsugan@ntu.edu.sg

Abstract. In a traveling salesman problem, if the set of nodes is divided into clusters so that a single node from each cluster can be visited, then the problem is known as the generalized traveling salesman problem where the objective is to find a tour with minimum cost passing through only a single node from each cluster. In this chapter, a discrete differential evolution algorithm is presented to solve the problem on a set of benchmark instances. The discrete differential evolution algorithm is hybridized with local search improvement heuristics to further improve the solution quality. Some speed-up methods presented by the authors previously are employed to accelerate the greedy node insertion into a tour. The performance of the hybrid discrete differential evolution algorithm is tested on a set of benchmark instances with symmetric distances ranging from 51 (11) to 1084 (217) nodes (clusters) from the literature. Computational results show its highly competitive performance in comparison to the best performing algorithms from the literature.

6.1 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

G.C. Onwubolu and D. Davendra (Eds.): Differential Evolution, SCI 175, pp. 139–162.
springerlink.com

warehouses [18], process planning for rotational parts [3], and the routing of clients through welfare agencies [28].

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 d_{ij}. Then, 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 $n_j \cap n_k = \phi$. 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 n_j.

The GTSP was first addressed in [9, 28, 32]. 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, 19]. 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 [19], 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 [18]. Similar adaptations of the farthest-insertion, nearest-insertion, and cheapest-insertion heuristics are proposed in Fischetti et al. [8]. In addition, Renaud & Boctor [24] 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. [25]. 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 [30] 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 [30] 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. [34, 35, 36] presented a discrete particle swarm optimization (**DPSO**) algorithm, a genetic algorithm (**GA**) and a hybrid iterated greedy (**HIG**) algorithm, respectively, whereas Silberholz & Golden proposed another **GA** in [29], which is denoted as **mrOXGA**.

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

6.2 Discrete Differential Evolution Algorithm

Differential evolution (DE) is a latest evolutionary optimization methods proposed by Storn & Price [31]. 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 [31], it has been successfully applied in a variety of applications that can be found in Corne et. al [5], Lampinen [19], Babu & Onwubolu [1]; and Price et al. [22].

Currently, there are several variants of DE algorithms. We follow the *DE/rand/1/bin* scheme of Storn & Price [31] with the inclusion of SPV rule in the algorithm. Pseudocode of the DE algorithm is given in Fig 6.1.

```
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 search to the target population individuals (Optional)
}While (Not Termination)
```

Fig. 6.1. DE Algorithm with Local Search

The basic elements of DE algorithm are summarized as follows:

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

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

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

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

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

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

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 given by $f_i\left(X_i^k\right)$, for the individual X_i^k.

Traditional DEs explained above are designed for continuous optimization problems where chromosomes are floating-point numbers. To cope with discrete spaces, a simple and novel discrete DE (DDE) algorithm is presented in [36, 20], where solutions are based on discrete/binary values. In the DDE algorithm, each target individual belonging to the *NP* number of individuals is represented by a solution as $X_i^k = \left[x_{i1}^k, x_{i2}^k, ..., x_{im}^k\right]$, consisting of discrete values of a permutation of clusters as well as a tour of nodes visited, at the generation *k*. The mutant individual is obtained by perturbing the generation best solution in the target population. So the differential variation is achieved in the form of perturbations of the best solution from the generation best solution in the target population. Perturbations are stochastically managed such that each individual in the mutant population is expected to be distinctive. To obtain the mutant individual, the following equation can be used:

$$V_i^k = \begin{cases} DC_d\left(X_g^{k-1}\right) & if\ r < P_m \\ insert\left(X_g^{k-1}\right) & otherwise \end{cases} \tag{6.1}$$

Where X_g^{k-1} is the best solution in the target population at the previous generation; P_m is the perturbation probability; DC_d is the destruction and construction procedure with the destruction size of d as a perturbation operator; and insert is a simple random insertion move from a given node to another node in the same cluster. A uniform random number r is generated between [0, 1]. If r is less than then the DC_d operator is applied to generate the mutant individual $V_i^k = DC_d\left(X_g^{k-1}\right)$; otherwise, the best solution from the previous generation is perturbed with a random insertion move resulting in the mutant individual $V_i^k = insert\left(X_g^{k-1}\right)$. Equation 6.1 will be denoted as $V_i^k := P_m \underset{i:=1,2,..,NP}{\oplus} DC_d\left(X_g^{k-1}\right)$ to ease the understanding of pseudocodes. Following the perturbation phase, the trial individual is obtained such that:

$$U_i^k = \begin{cases} CR\left(V_i^k, X_i^{k-1}\right) & if\ r < P_c \\ V_i^k & otherwise \end{cases} \tag{6.2}$$

where CR is the crossover operator; and P_c is the crossover probability. In other words, if a uniform random number r is less than the crossover probability P_c, then the crossover operator is applied to generate the trial individual $U_i^k = CR\left(V_i^k, X_i^{k-1}\right)$. Otherwise the

trial individual is chosen as $U_i^k = V_i^k$. By doing so, the trial individual is made up either from the outcome of perturbation operator or from the crossover operator. Equation 6.2 will be denoted as $U_i^k := P_c \underset{i:=1,2,..,NP}{\oplus} CR\left(V_i^k, X_i^{k-1}\right)$.

Finally, the selection operator is carried out based on the survival of the fitness among the trial and target individuals such that:

$$X_i^k = \begin{cases} U_i^k \ if \ f\left(\pi_i^k \leftarrow U_i^k\right) < f\left(\pi_i^{k-1} \leftarrow X_i^{k-1}\right) \\ X_i^{k-1} \quad otherwise \end{cases} \tag{6.3}$$

Equation 6.3 will be denoted as $X_i^k = \arg\min \left\{ f\left(\pi_i^k \leftarrow \underset{i:=1,2,..,NP}{U_i^k}\right), f\left(\pi_i^{k-1} \leftarrow X_i^{k-1}\right)\right\}$.

6.2.1 Solution Representation

We employ a path representation for the GTSP in this chapter. In the path representation, each consecutive node is listed in order. A disadvantage of this representation is due to the fact that there is no guarantee that a randomly selected solution will be a valid GTSP tour because there is no guarantee that each cluster is represented exactly once in the path without some repair procedures. To handle the GTSP, we include both cluster and tour information in the solution representation. The solution representation is illustrated in Table 6.1 where $d_{\pi_j \pi_{j+1}}$ shows the distance from node π_j to node π_{j+1}. Population individuals can be constructed in such a way that first a permutation of clusters is determined randomly, and then since each cluster contains one or more nodes, a tour is established by randomly choosing a single node from each corresponding cluster. For example, n_j stands for the cluster in the j^{th} dimension, whereas π_j represents the node to be visited from the cluster n_j.

Table 6.1. Solution Representation

	j	1	...	$m-1$	m
	n_j	n_1	...	n_{m-1}	n_m
	π_j	π_1	...	π_{m-1}	π_m
X	$d_{\pi_j \pi_{j+1}}$	$d_{\pi_1 \pi_2}$	...	$d_{\pi_{m-1} \pi_m}$	$d_{\pi_m \pi_1}$
	$\sum_{j=1}^{m} d_{\pi_j \pi_{j+1}} + d_{\pi_m \pi_1}$	$d_{\pi_1 \pi_2}$	...	$d_{\pi_{m-1} \pi_m}$	$d_{\pi_m \pi_1}$

As illustrated in Table 6.1, the objective function value implied by a solution X with m nodes is the total tour length, which is given by

$$F(\pi) = \sum_{j=1}^{m-1} d_{\pi_j \pi_{j+1}} + d_{\pi_m \pi_1} \tag{6.4}$$

Table 6.2. Solution Representation

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

Now, 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 $n_5 = \{21,..,25\}$. Table 6.2 illustrates the example solution in detail.

So, the fitness function of the individual is given by $F(\pi) = d_{14,5} + d_{5,22} + d_{22,8} + d_{8,16} + d_{16,14}$.

6.2.2 Complete Computational Procedure of DDE

The complete computational procedure of the DDE 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 6.2, $\left\{X_i^0, i = 1,2,...,NP\right\}$ where $X_i^0 = \left[x_{i1}^0, x_{i2}^0, ..., x_{im}^0\right]$.
  - Evaluate each individual i in the population using the objective function $f_i^0\left(\pi_i^o \leftarrow X_i^0\right)$ for $i = 1,2,...,NP$.
- **Step 2: Update generation counter**
  - $k = k + 1$
- **Step 3: Generate mutant population**
  - For each target individual, X_i^k, $i = 1,2,...,NP$, at generation k, a mutant individual, $V_i^k = \left[v_{i1}^k, v_{i2}^k, ..., v_{im}^k\right]$, is determined such that:

$$V_i^k = X_{a_i}^{k-1} + F\left(X_{b_i}^{k-1} - X_{c_i}^{k-1}\right) \tag{6.5}$$

    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, $V_i^k = \left[v_{i1}^k, v_{i2}^k, ..., v_{im}^k\right]$, an integer random number between 1 and n, i.e., $D_i \in (1,2,...,m)$, is chosen, and a trial individual, $U_i^k = \left[u_{i1}^k, u_{i2}^t, ..., u_{im}^k\right]$ is generated such that:

$$u_{ij}^k = \begin{cases} v_{ij}^k, \; if \; r_{ij}^k \leq CR \; or \; j = D_i \\ x_{ij}^{k-1}, \quad Otherwise \end{cases} \tag{6.6}$$

    where the index D refers to a randomly chosen dimension ($j = 1,2,...,m$), which is used to ensure that at least one parameter of each trial individual U_i^k differs

from its counterpart in the previous generation U_i^{k-1}, *CR* is a user-defined crossover constant in the range (0, 1), and r_{ij}^k 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: Evaluate trial population**
  - Evaluate the trial population using the objective function $f_i^k\left(\pi_i^k \leftarrow U_i^k\right)$ for $i = 1,2,...,NP$.
- **Step 6: Selection**
  - To decide whether or not the trial individual U_i^k should be a member of the target population for the next generation, it is compared to its counterpart target individual X_i^{k-1} at the previous generation. The selection is based on the survival of fitness among the trial population and target population such that:

$$X_i^k = \begin{cases} U_i^k, & if\ f\left(\pi_i^k \leftarrow U_i^k\right) \leq f\left(\pi_i^{k-1} \leftarrow X_i^{k-1}\right) \\ X_i^{t-1}, & otherwise \end{cases} \tag{6.7}$$

- **Step 7: 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

6.2.3 NEH Heuristic

Due to the availability of the insertion methods from the TSP literature, which are modified in this chapter, it is possible to apply the NEH heuristic of Nawaz et al. [17] 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 π.
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 a Hamiltonian cycle, partial tours will be evaluated with the first node being the last node as well. As an example, partial tours, (π_1, π_2, π_1) and (π_2, π_1, π_2) are evaluated first.
3. Repeat the following steps until all nodes are inserted. In the k^{th} 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 of these k tentative partial tours the one 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 $\pi = \{3,1,4,2,5\}$. Following example illustrates the implementation of the NEH heuristic for the GTSP:

1. Current solution is $\pi = \{3,1,4,2,5\}$
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:
   a) 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 $\{3,4,1\}$.
   b) 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 $\{3,2,4,1\}$.
   c) 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 $\pi = \{3,2,4,5,1\}$.

6.2.4 Insertion Methods

In this section of the chapter, the insertion methods are modified from the literature and facilitate the use of the local search. It is important to note that for simplicity, we do not include the cluster information in the following examples. However, whenever an insertion move is carried out, the corresponding cluster is also inserted in the solution. 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. [26] for the TSP. Snyder & Daskin [30] 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=8 will be inserted in a partial tour in Table 6.3.

Table 6.3. Current solution

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

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.

Table 6.4. Insertion of node π_k^R=8 in the first slot

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

$Remove = d_{\pi_4\pi_1} = d_{16,14}$
$Add = d_{\pi_u\pi_k} + d_{\pi_k\pi_v} = d_{14,8} + d_{8,5}$
$F(\pi) = F\left(\pi^D\right) + Add - Remove$
$F(\pi) = d_{14,5} + d_{5,22} + d_{22,16} + d_{16,14} + d_{8,14} + d_{16,8} - d_{16,14}$
$F(\pi) = d_{14,5} + d_{5,22} + d_{22,16} + d_{8,14} + d_{16,8}$

B Insertion of node,pair π_k^R in the last 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.

Table 6.5. Insertion of node π_k^R=8 in the last slot

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

$Remove = d_{\pi_4\pi_1} = d_{16,14}$
$Add = d_{\pi_4\pi_k} + d_{\pi_k\pi_1} = d_{16,8} + d_{8,14}$
$F(\pi) = F\left(\pi^D\right) + Add - Remove$
$F(\pi) = d_{14,5} + d_{5,22} + d_{22,16} + d_{16,14} + d_{16,8} + d_{8,14} - d_{16,14}$
$F(\pi) = d_{14,5} + d_{5,22} + d_{22,16} + d_{16,8} + d_{8,14}$

C Insertion of node π_k^R between the 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 complete and the partial solutions respectively.

$Remove = d_{\pi_u\pi_v} = d_{14,5}$
$Add = d_{\pi_u\pi_k} + d_{\pi_k\pi_v} = d_{14,8} + d_{8,5}$
$F(\pi) = F\left(\pi^D\right) + Add - Remove$

Table 6.6. Insertion of node π_k^R=8 in between the edge $\left(\pi_u^D, \pi_v^D\right)$

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

$F(\pi) = d_{14,5} + d_{5,22} + d_{22,16} + d_{14,8} + d_{8,5} - d_{33,44} + d_{44,41} + d_{41,25} + d_{25,24} + d_{14,5}$

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

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 [30], where the cost of an insertion of node π_k^R in an edge $\left(\pi_u^D, \pi_v^D\right)$ is evaluated by $C = d_{\pi_u^D \pi_k^R} + d_{\pi_k^R \pi_v^D} - d_{\pi_u^D \pi_v^D}$. Instead, we directly calculate the fitness function value of the complete tour after using the insertion methods above, i.e., well suited for the NEH insertion heuristic..

6.2.5 Destruction and Construction Procedure

We employ the destruction and construction (DC) procedure of the iterated greedy (IG) algorithm [27] in the DDE 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.

The DC procedure for the GTSP is illustrated through Table 6.7 and Table 6.12 using the GTSP instance in Table 6.2. Note that the destruction size is $d = 2$ and the perturbation strength is $p = 1$ in this example. Perturbation strength $p = 1$ indicates replacing (mutating) only a single node among two nodes with another one from the same cluster.

Table 6.7. Current Solution

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

Table 6.8. Destruction Phase

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

Table 6.9. Destruction Phase

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

Table 6.10. Destruction Phase-Mutation

j	1	2	3	j	1	2
n_j^D	3	5	4	n_j^R	1	**2**
π_j^D	14	22	16	π_j^R	5	**9**

Table 6.11. Construction Phase

j	1	2	3	4	j	1
n_j^D	3	5	1	4	n_j^R	2
π_j^D	14	22	5	16	π_j^R	9

Table 6.12. Final Solution

j	1	2	3	4	5
n_j^D	3	**2**	5	**1**	4
π_j^D	14	**9**	22	**5**	16

Table 6.13. Two-Cut PTL Crossover Operator

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

Step 1.a. Choose $d = 2$ nodes with corresponding clusters, randomly.
Step 1.b. Establish $\pi^D = \{14, 22, 16\}$, $n^D = \{3, 5, 4\}$, $\pi^R = \{5, 8\}$ and $n^R = \{1, 2\}$.
Step 1.c. Perturb $\pi^R = \{5, 8\}$ to $\pi^R = \{5, 9\}$ by randomly choosing $n_2^R = 2$ in the set $n^R = \{1, 2\}$, and randomly replacing $n_2^R = 8$ with $n_2^R = 9$ from the same cluster n_2.
Step2.a. After the best insertion of node $\pi_1^R = 5$ and the cluster $\pi_1^R = 1$.
Step2.b. After the best insertion of node $n_2^R = 9$ and the cluster $\pi_1^R = 2$.
$F(\pi) = d_{5,22} + d_{22,16} + d_{16,14} + d_{14,8} + d_{8,5}$

6.2.6 PTL Crossover Operator

Two-cut PTL crossover operator developed by Pan et al. [21] is used in the DDE algorithm. The two-cut PTL crossover operator is able to produce a pair of distinct offspring even from two identical parents. An illustration of the two-cut PTL cross-over operator is shown in Table 6.13.

In the PTL crossover, a block of nodes and clusters from the first parent is determined by two cut points randomly. This block is either moved to the right or left corner of the offspring. Then the offspring is filled out with the remaining nodes and corresponding clusters from the second parent. This procedure will always produce two distinctive offspring even from the same two parents as shown in Table 6.13. In this chapter, one of these two unique offspring is chosen randomly with an equal probability of 0.5.

6.2.7 Insert Mutation Operator

Insert mutation operator is a modified insert mutation considering the clusters in the solution representation. It is also used in the perturbation of the solution in the destruction and construction procedure. It is basically related to removing a node from a tour of an individual, and replacing that particular node with another one from the same cluster. It is illustrated in Table 6.14.

As shown in Table 6.14, the cluster $n_2 = 5$ is randomly selected and its corresponding node $\pi_2 = 23$ is replaced by node $\pi_2 = 22$ from the same cluster $n_2 = 5$.

Table 6.14. Insert Mutation

j		1	**2**	3	4	5
X_i	n_j	3	**5**	2	1	4
	π_j	12	**23**	8	4	19
X_i	n_j	3	**5**	2	1	4
	π_j	12	**22**	8	4	19

6.2.8 DDE Update Operations

To figure out how the individuals are updated in the DDE algorithm, an example using the GTSP instance in Table 6.2 is also illustrated through Table 6.15 and Table 6.18. Assume that the mutation and crossover probabilities are 1.0, the two-cut PTL crossover

Table 6.15. An Example of Individual Update

	j	1	2	3	4	5
X_i	n_j	3	5	2	1	4
	π_j	12	23	8	4	19
G_i	n_j	3	1	5	2	4
	π_j	15	4	24	7	17

Table 6.16. Insert Mutation

	j	1	2	**3**	4	5
X_i	n_j	3	1	**5**	2	4
	π_j	15	4	**24**	7	17
G_i	n_j	3	1	**5**	2	4
	π_j	15	4	**25**	7	17

Table 6.17. Two-Cut PTL Crossover

	j	1	2	**3**	**4**	5
X_i	n_j	3	5	**2**	**1**	4
	π_j	12	23	**8**	**4**	19
G_i	n_j	3	*1*	5	*2*	4
	π_j	15	*4*	25	*7*	17
U_i	n_j	**2**	**1**	3	5	4
	π_j	**8**	**4**	15	25	17

Table 6.18. Selection For Next Generation

	j	1	2	3	4	5
X_i	n_j	3	5	2	1	4
	π_j	12	23	8	4	19
U_i	n_j	2	1	3	5	4
	π_j	8	4	15	25	17
Assume that $f(U_i) \leq f(X_i)$ $X_i = U_i$						
U_i	n_j	2	1	3	5	4
	π_j	8	4	15	25	17

and insert mutation operators are employed. Given the individual and the global best (best so far solution for DDE) solution, the global best solution is first mutated by using equation 6.1. For example, in Table 6.15, the dimensions u=3 is chosen randomly with its corresponding cluster and node. Node $\pi_u = \pi_3 = 25$ is replaced by $\pi_u = \pi_3 = 24$ from the same cluster $n_u = n_3 = 5$, thus resulting in the mutant individual V_i. Then the mutant individual V_i is recombined with its corresponding individual X_i in the target population to generate the trial individual U_i by using equation 6.2. Finally, the target individual X_i is compared to the trial individual U_i to determine which one would survive for the next generation based on the survival of the fittest by using equation 6.3.

6.3 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 [30], denoted as **LocalSearchSD** in this chapter, and the **2-opt** heuristic [16] were 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 6.2.

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 [30] is outlined in Fig 6.3, whereas the proposed DDE algorithm is given in Fig 6.4. 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 $u, v \in V$.

```
procedure  LS (π ← X)
    h := 1
    while  (h ≤ m) do
        π* := LocalSearchSD(π ← X)
        if  (f(π* ← X) ≤ f(π ← X))  then
            π ← X := π* ← X
            h := 1
        else
            h := h + 1
        else
    endwhile
    return  π ← X
end procedure
```

Fig. 6.2. The Local Search Scheme

```
Procedure  SWAP()
remove  j  from  T
for each k ∈ V_j
    c^k ← min{ d_uk + d_kv − d_uv / (u,v) is an edge in T }
    k* ← arg min_{k∈V_j} {c_k}
insert  k*  into T between (u, v)
```

Fig. 6.3. The SWAP Procedure

6.4 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 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 DDE algorithm was coded in Visual C++ and run on an Intel Centrino Duo 1.83 GHz Laptop with 512MB memory. The population size was fixed at 100. The initial population is constructed randomly and then the NEH insertion heuristic was applied to each random solution. The destruction size and perturbation strength were taken as 5 and 3, respectively. The crossover and mutation probability were taken as 0.9 and 0.2, respectively. PTL [33] crossover operator is used in the DDE algorithm. The DDE 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 as follows

$$\Delta_{avg} = \sum_{i=1}^{R} \left(\frac{(H_i - OPT) \times 100}{OPT} \right) / R \tag{6.8}$$

$$
\begin{aligned}
&\textit{Procedure DE\_GTSP}\\
&\textit{Set } CR, F, NP, TerCriterion\\
&X = \left(x_1^0, x_2^0, .., x_{NP}^0\right)\\
&f\left(\pi_i^0 \leftarrow x_i^0\right)\\
&\quad i{:=}1,2,..,NP\\
&\pi_i^0 \leftarrow x_i^0 = 2\_opt\left(\pi_i^0 \leftarrow x_i^0\right)\\
&\qquad\qquad i{=}1,2,..,NP\\
&\pi_i^0 \leftarrow x_i^0 = LS\left(\pi_i^0 \leftarrow x_i^0\right)\\
&\qquad\qquad i{=}1,2,..,NP\\
&\pi_g^0 \leftarrow x_i^0 = \arg\min\left\{f\left(\pi_i^0 \leftarrow x_i^0\right)\right\}\\
&\qquad\qquad\quad i{=}1,2,..,NP\\
&\pi_B := \pi_g^0 \leftarrow x_i^0\\
&k := 1\\
&\textit{while } \left(\textit{Not } TerCriterion\right) \textit{ do}\\
&\quad V_{ij}^k := x_{ia}^k + F\left(X_{ib}^k + X_{ic}^k\right)\\
&\qquad\qquad i{:=}1,2,..,NP, j{=}1,2,..,m\\
&\quad u_{ij}^k = \begin{cases} v_{ij}^k \textit{ if } r_{ij}^k < CR \textit{ or } j = D_j \\ x_{ij}^{k-1} \textit{ otherwise} \end{cases}\\
&\qquad\qquad i{=}1,2,..,NP, j{=}1,2,..,m\\
&\quad f\left(\pi_i^k \leftarrow U_i^k\right)\\
&\qquad i{:=}1,2,..,NP\\
&\quad \pi_i^k \leftarrow U_i^k = 2\_opt\left(\pi_i^k \leftarrow U_i^k\right)\\
&\qquad\qquad i{=}1,2,..,NP\\
&\quad \pi_i^k \leftarrow U_i^k = LS\left(\pi_i^k \leftarrow U_i^k\right)\\
&\qquad\qquad i{=}1,2,..,NP\\
&\quad X_i^k = \begin{cases} u_i^k \textit{ if } f\left(\pi_i^k \leftarrow u_i^k\right) < f\left(\pi_i^{k-1} \leftarrow X_i^{k-1}\right) \\ x_i^{k-1} \textit{ otherwise} \end{cases}\\
&\qquad\qquad\qquad i{=}1,2,..,NP\\
&\quad \pi_g^k \leftarrow X_g^k = \arg\min\left\{f\left(\pi_i^k \leftarrow X_i^k\right), f\left(\pi_g^{k-1} \leftarrow X_g^{k-1}\right)\right\}\\
&\qquad\qquad\qquad i{:=}1,2,..,NP\\
&\quad \pi_B \leftarrow X_B = \arg\min\left\{f\left(\pi_B \leftarrow X_B\right), f\left(\pi_g^k \leftarrow X_g^k\right)\right\}\\
&\quad k := k+1\\
&\textit{endwhile}\\
&\textit{return } \pi_B \leftarrow X_B
\end{aligned}
$$

Fig. 6.4. The DDE Algorithm with Local Search Heuristics

where H_i, OPT and R are the objective function values generated by the DDE 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.

6.4.1 Solution Quality

Table 6.19 gives the computational results for each of the problem instances in detail. As seen in Table 6.19, the DDE algorithm was able to obtain optimal solutions in at least two of the five runs for 35 out of 36 problems tested (97%). For 32 (89%) out of

Table 6.19. Computational Results of DDE algorithm

Instance	OPT	n_{opt}	Δ_{avg}	Δ_{min}	Δ_{max}	I_{avg}	I_{min}	I_{max}	t_{avg}	t_{min}	t_{max}
11EIL51	174	5	0	0	0	1	1	1	0.04	0.02	0.06
14ST70	316	5	0	0	0	1	1	1	0.04	0.03	0.05
16EIL76	209	5	0	0	0	1	1	1	0.05	0.05	0.06
16PR76	64925	5	0	0	0	1	1	1	0.06	0.05	0.06
20KROA100	9711	5	0	0	0	1	1	1	0.09	0.08	0.09
20KROB100	10328	5	0	0	0	1	1	1	0.09	0.08	0.09
20KROC100	9554	5	0	0	0	1	1	1	0.08	0.08	0.09
20KROD100	9450	5	0	0	0	1	1	1	0.08	0.08	0.09
20KROE100	9523	5	0	0	0	1	1	1	0.09	0.08	0.09
20RAT99	497	5	0	0	0	1	1	1	0.08	0.08	0.09
20RD100	3650	5	0	0	0	1	1	1	0.09	0.08	0.09
21EIL101	249	5	0	0	0	1	1	1	0.08	0.08	0.09
21LIN105	8213	5	0	0	0	1	1	1	0.1	0.09	0.11
22PR107	27898	5	0	0	0	1	1	1	0.1	0.09	0.11
25PR124	36605	5	0	0	0	1	1	1	0.13	0.13	0.14
26BIER127	72418	5	0	0	0	1	1	1	0.14	0.13	0.14
28PR136	42570	5	0	0	0	1	1	1	0.18	0.16	0.19
29PR144	45886	5	0	0	0	1	1	1	0.18	0.17	0.2
30KROA150	11018	5	0	0	0	1	1	1	0.2	0.19	0.2
30KROB150	12196	5	0	0	0	1	1	1	0.2	0.19	0.2
31PR152	51576	5	0	0	0	1.2	1	2	0.22	0.19	0.28
32U159	22664	5	0	0	0	1	1	1	0.23	0.22	0.24
39RAT195	854	5	0	0	0	1.4	1	2	0.42	0.36	0.48
40D198	10557	5	0	0	0	1.4	1	2	0.44	0.38	0.52
40KROA200	13406	5	0	0	0	1.2	1	2	0.41	0.38	0.48
40KROB200	13111	5	0	0	0	7	1	22	0.93	0.41	2.03
45TS225	68340	3	0.04	0	0.09	9.8	1	33	1.32	0.47	3.05
46PR226	64007	5	0	0	0	1	1	1	0.42	0.41	0.44
53GIL262	1013	2	0.41	0	0.69	11.4	1	44	2	0.72	5.36
53PR264	29549	5	0	0	0	1.4	1	3	0.79	0.67	1.23
60PR299	22615	2	0.05	0	0.09	11.2	6	19	3.24	2.5	5.36
64LIN318	20765	5	0	0	0	14.2	3	45	4.37	2.13	10.28
80RD400	6361	5	0	0	0	14.8	11	18	8.3	6.86	9.97
84FL417	9651	3	0.01	0	0.02	13.8	8	24	6.86	4.58	10.88
88PR439	60099	5	0	0	0	15.2	8	23	8.54	6.06	11.08
89PCB442	21657	5	0	0	0	19	10	35	11.72	7.86	17.8
Overal Avg		4.72	0.01	0	0.02	4.03	2.11	8.22	1.45	1	2.27

36 problems, the DDE algorithm obtained the optimal solution in *every* trial. The DDE algorithm solved all the problems with a 0.01% deviation on average, 0.00% deviation on minimum and 0.02% deviation on maximum. The overall hit ratio was 4.72, which indicates that the DDE algorithm was able to find the 95% of the optimal solutions on overall average. The worst case performance was never more than 0.02% above optimal

on overall average. In other words, it indicates that the DDE algorithm has a tendency of yielding consistent solutions across a wide variety of problem instances. To highlight its consistency more, the range between the best and worst case was only 0.02% on overall average, thus indicating a very robust algorithm.

6.4.2 Computation Time

Table 6.19 also gives necessary information about CPU time requirement for each of the problem instances. The DDE algorithm is very fast in terms of CPU time requirement due to the mean CPU time of less than 12 seconds for all instances. In addition, its maximum CPU time was no more than 18 seconds for all instances. The DDE algorithm was able to find its best solution in the first 4.03 generations on overall average and spent most of the time waiting for the termination condition. It took 2.11 generations at minimum and only 8.22 generations at maximum on overall average to find its best solution for each problem instance. Since the local search heuristics are applied to each problem instance at each generation, most of the running times have been devoted to the local search improvement heuristics, which indicates the impact of the them on the solution quality. It implies the fact that with some better local search heuristics such as Renaud and Boctor's G2-opt or G3-opt, as well as with some speed-up methods for 2-opt heuristic, its CPU time performance may be further improved.

6.4.3 Comparison to Other Algorithms

We compare the DDE algorithm to two genetic algorithms, namely, RKGA by Snyder & Daskin [30] and mrOXGA by Silberholz & Golden [29], where RKGA is re-implemented under the same machine environment. Table 6.20 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 [29]. A two-sided paired t-test which compares the results on Table 6.20 with a null hypothesis that the algorithms were identical generated p-values of 0.175 and 0.016 for DDE vs. mrOXGA and DDE vs. RKGA,respectively, suggesting near-identical results between DDE and mrOXGA. On the other hand, the paired t-test confirms that the differences between DDE and RKGA were significant on the behalf of DDE subject to the fact that DDE was computationally less expensive than both RKGA and mrOXGA since p-values were 0.001 for DDE vs. mrOXGA and 0.008 for DDE vs. RKGA.

In addition to above, Silberholz & Golden [29] 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 DDE results to those presented in Silberholz & Golden [29]. As seen in Table 6.21, DDE generated consistently better results than both RKGA and mrOXGA in terms of both solution quality and CPU time requirement even if the larger instances are considered. In particular, 8 out 9 larger instances are further improved by the DDE algorithm. The paired t-test on the objective function values on Table 6.21 confirms that the differences between DDE and RKGA were significant since p-value was 0.033 (null hypothesis is rejected), whereas DDE was equivalent to mrOXGA since p-value was 0.237. In

Table 6.20. Comparison for Optimal Instances

	DDE		mrOXGA		RKGA	
Instance	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}	Δ_{avg}	t_{avg}
11EIL51	0	0.04	0	0.26	0	0.08
14ST70	0	0.04	0	0.35	0	0.07
16EIL76	0	0.05	0	0.37	0	0.11
16PR76	0	0.06	0	0.45	0	0.16
20KROA100	0	0.09	0	0.63	0	0.25
20KROB100	0	0.09	0	0.6	0	0.22
20KROC100	0	0.08	0	0.62	0	0.23
20KROD100	0	0.08	0	0.67	0	0.43
20KROE100	0	0.09	0	0.58	0	0.15
20RAT99	0	0.08	0	0.5	0	0.24
20RD100	0	0.09	0	0.51	0	0.29
21EIL101	0	0.08	0	0.48	0	0.18
21LIN105	0	0.1	0	0.6	0	0.33
22PR107	0	0.1	0	0.53	0	0.2
25PR124	0	0.13	0	0.68	0	0.26
26BIER127	0	0.14	0	0.78	0	0.28
28PR136	0	0.18	0	0.79	0.16	0.36
29PR144	0	0.18	0	1	0	0.44
30KROA150	0	0.2	0	0.98	0	0.32
30KROB150	0	0.2	0	0.98	0	0.71
31PR152	0	0.22	0	0.97	0	0.38
32U159	0	0.23	0	0.98	0	0.55
39RAT195	0	0.42	0	1.37	0	1.33
40D198	0	0.44	0	1.63	0.07	1.47
40KROA200	0	0.41	0	1.66	0	0.95
40KROB200	0	0.93	0.05	1.63	0.01	1.29
45TS225	0.04	1.32	0.14	1.71	0.28	1.09
46PR226	0	0.42	0	1.54	0	1.09
53GIL262	0.41	2	0.45	3.64	0.55	3.05
53PR264	0	0.79	0	2.36	0.09	2.72
60PR299	0.05	3.24	0.05	4.59	0.16	4.08
64LIN318	0	4.37	0	8.08	0.54	5.39
80RD400	0	8.3	0.58	14.58	0.72	10.27
84FL417	0.01	6.86	0.04	8.15	0.06	6.18
88PR439	0	8.54	0	19.06	0.83	15.09
89PCB442	0	11.72	0.01	23.43	1.23	11.74
Avg	0.01	1.45	0.04	2.99	0.13	2

Table 6.21. Comparision to Silberholz & Golden-Time is milliseconds

	DE		mrOXGA		RKGA	
Instance	F_{avg}	t_{avg}	F_{avg}	t_{avg}	F_{avg}	t_{avg}
11EIL51	**174**	37.6	**174**	259.2	**174**	78.2
14ST70	**316**	43.8	**316**	353	**316**	65.6
16EIL76	**209**	50	**209**	369	**209**	106.4
16PR76	**64925**	56.4	**64925**	447	**64925**	156.2
20KROA100	**9711**	90.6	**9711**	628.2	**9711**	249.8
20KROB100	**10328**	87.6	**10328**	603.2	**10328**	215.6
20KROC100	**9554**	84.4	**9554**	621.8	**9554**	225
20KROD100	**9450**	81.2	**9450**	668.8	**9450**	434.4
20KROE100	**9523**	87.6	**9523**	575	**9523**	147
20RAT99	**497**	81.2	**497**	500	**497**	243.8
20RD100	**3650**	90.6	**3650**	506.2	**3650**	290.8
21EIL101	**249**	81.2	**249**	478.2	**249**	184.6
21LIN105	**8213**	96.8	**8213**	603.2	**8213**	334.4
22PR107	**27898**	96.8	27898.6	534.4	27898.6	197
25PR124	**36605**	134.2	**36605**	678	**36605**	259
26BIER127	**72418**	137.4	**72418**	784.4	**72418**	275.2
28PR136	**42570**	175	**42570**	793.8	42639.8	362.8
29PR144	**45886**	184.2	**45886**	1003.2	45887.4	437.6
30KROA150	**11018**	200	**11018**	981.2	**11018**	319
30KROB150	**12196**	200	**12196**	978.4	**12196**	712.4
31PR152	**51576**	218.8	**51576**	965.4	**51576**	381.2
32U159	**22664**	228.2	**22664**	984.4	**22664**	553.2
39RAT195	**854**	415.6	**854**	1374.8	**854**	1325
40D198	**10557**	437.6	**10557**	1628.2	10564	1468.6
40KROA200	**13406**	412.4	**13406**	1659.4	**13406**	950.2
40KROB200	**13111**	931.2	13117.6	1631.4	13112.2	1294.2
45TS225	**68364**	1322	68435.2	1706.2	68530.8	1087.4
46PR226	**64007**	421.8	**64007**	1540.6	**64007**	1094
53GIL262	**1017.2**	2000	1017.6	3637.4	1018.6	3046.8
53PR264	**29549**	793.8	**29549**	2359.4	29574.8	2718.6
60PR299	**22627**	3243.6	**22627**	4593.8	22650.2	4084.4
64LIN318	**20765**	4368.8	**20765**	8084.4	20877.8	5387.6
80RD400	**6361**	8303.2	6397.8	14578.2	6407	10265.6
84FL417	**9651.6**	6856.4	9654.6	8152.8	9657	6175.2
88PR439	**60099**	8543.6	**60099**	19059.6	60595.4	15087.6
89PCB442	**21657**	11718.8	21658.2	23434.4	21923	11743.8
99D493	**20059.2**	15574.8	20117.2	35718.8	20260.4	14887.8
115RAT575	**2421**	20240.2	2414.8	48481	2442.4	46834.4
131P654	**27430**	30428.4	27508.2	32672	27448.4	46996.8
132D657	**22544.8**	57900	22599	132243.6	22857.6	58449.8
145U724	**17367.2**	74687.4	17370.6	161815.2	17806.2	59625.2
157RAT783	**3272.2**	77000.2	3300.2	152147	3341	89362.4
201PR1002	114692.8	211025.2	**114582.2**	464356.4	117421.2	332406.2
212U1060	**106460**	247187.4	108390.4	594637.4	110158	216999.8
217VM1084	**131718.2**	292381.6	131884.6	562040.6	133743.4	390115.6
Overal Avg	**27502.7**	**23971.9**	27554.28	50930.41	27741.29	29503.03

terms of CPU times, the paired t-test on the CPU times confirms that the differences between DDE and mrOXGA were significant since the p-values was 0.020, whereas it was failed to reject the null hypothesis of being equal difference between DDE and RKGA due to the p-value of 0.129. Briefly, the paired t-test indicates that DDE was able to generate lower objective function values with less CPU times than mrOXGA. On the other hand, DDE yielded much better objective function values with identical CPU times than RKGA.

6.5 Conclusions

A DDE algorithm is presented to solve the GTSP on a set of benchmark instances ranging from 51 (11) to 1084 (217) nodes (clusters). The contributions of this paper can be summarized as follows. A unique solution representation including both cluster and tour information is presented, which handles the GTSP properly when carrying out the DDE operations. To the best of our knowledge, this is the first reported application of the DDE algorithm applied to the GTSP. The perturbation scheme is presented in the destruction procedure. Furthermore, the DDE algorithm is donated with very effective local search methods, 2-opt and SWAP procedure, in order to further improve the solution quality. Ultimately, the DDE algorithm was able to find optimal solutions for a large percentage of problem instances from a set of test problems from the literature. It was also able to further improve 8 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. In particular, its performance on the larger instances is noteworthy.

Acknowledgement

P. Suganthan acknowledges the financial support offered by the A*Star (Agency for Science, Technology and Research) under the grant # 052 101 0020.

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. Nawaz, M., Enscore, E., Ham, I.: Heuristic algorithm for the m-machine, n-job flow shop sequencing problem. OMEGA 11(1), 91–95 (1983)
18. Noon, C.: The generalized traveling salesman problem, Ph.D. thesis. University of Michigan (1988)
19. Noon, C., Bean, J.: A Lagrangian based approach for the asymmetric generalized traveling salesman problem. Oper. Res. 39(4), 623–632 (1991)
20. Pan, Q.-K., Tasgetiren, M., Liang, Y.-C.: A Discrete differential evolution algorithm for the permutation flowshop scheduling problem. Comput. Ind. Eng. (2008)
21. Pan, Q.-K., Tasgetiren, M., Liang, Y.-C.: A Discrete Particle Swarm Optimization Algorithm for the No-Wait Flowshop Scheduling Problem with Makespan and Total Flowtime Criteria. Comput. Oper. Res. 35, 2807–2839 (2008)
22. Price, K., Storn, R., Lapinen, J.: Differential Evolution - A Practical Approach to Global Optimization. Springer, Heidelberg (2006)
23. Reinelt, G.: TSPLIB. A travelling salesman problem library. ORSA Journal on Computing 4, 134–143 (1996)
24. Renaud, J., Boctor, F.: An efficient composite heuristic for the symmetric generalized traveling salesman problem. Eur. J. Oper. Res. 108(3), 571–584 (1998)
25. Renaud, J., Boctor, F., Laporte, G.: A fast composite heuristic for the symmetric traveling salesman problem. INFORMS Journal on Computing 4, 134–143 (1996)
26. 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)
27. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the sequence flowshop scheduling problem. Eur. J. Oper. Res. 177(3), 2033–2049 (2007)
28. Saskena, J.: Mathematical model of scheduling clients through welfare agencies. Journal of the Canadian Operational Research Society 8, 185–200 (1970)
29. 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)

30. Snyder, L., Daskin, M.: A random-key genetic algorithm for the generalized traveling salesman problem. Eur. J. Oper. Res. 174, 38–53 (2006)
31. 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)
32. 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)
33. 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)
34. Tasgetiren, M., Suganthan, P., Pan, Q.-K.: A discrete particle swarm opti-mization 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)
35. 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)
36. 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)