# Memetic Algorithm with a Large Neighborhood Crossover Operator for the Generalized Traveling Salesman Problem

Boris Bontoux, Christian Artigues, Dominique Feillet

# Memetic Algorithm with a Large Neighborhood Crossover Operator for the Generalized Traveling Salesman Problem

Boris Bontoux [a,*], Christian Artigues [b], Dominique Feillet [a]

[a] *Université d'Avignon et des Pays de Vaucluse*
*Laboratoire d'Informatique d'Avignon*
*F-84911 Avignon Cedex 9, France*

[b] *LAAS CNRS, 7 avenue du Colonel Roche, 31077 Toulouse CEDEX 4*

**Abstract**

The Generalized Traveling Salesman Problem (GTSP) is a generalization of the well-known Traveling Salesman Problem (TSP), in which the set of nodes is divided into mutually exclusive clusters. The objective of the GTSP consists in visiting each cluster exactly once in a tour, while minimizing the sum of the routing costs. This paper addresses the solution of the GTSP using a Memetic Algorithm procedure. The originality of our approach rests on the crossover procedure that uses a large neighborhood search. This algorithm is compared with other algorithms on a set of 41 standard test problems with up to 442 nodes. The obtained results show that our algorithm is efficient in both solution quality and computation time.

*Key words:* Genetic Algorithm, Traveling Salesman Problem, Large Neighborhood Search

## 1 Introduction

In this paper, we propose a solution method for the Generalized Traveling Salesman Problem (GTSP) based on a memetic algorithm (genetic algorithm plus local search, see (11) for further details). The GTSP is a generalization

* Corresponding author.
*Email addresses:* boris.bontoux@univ-avignon.fr (Boris Bontoux), artigues@laas.fr (Christian Artigues), dominique.feillet@univ-avignon.fr (Dominique Feillet).

of the well-known Traveling Salesman Problem (TSP). The main contribution of the paper stands in the crossover operator based on the exploration of a large neighborhood around the father and mother individuals.

The GTSP can be described as follows. Let $G = (V, E)$ be a complete undirected graph, $V = (v_1, \ldots, v_n)$ a set of cities, $W = (W_1, \ldots, W_m)$ a set of clusters, where $0 < m \leq n$. Each city $v_i \in V$ belongs to exactly one cluster (the clusters are mutually disjoint, thus for $i \neq j$, $W_i \cap W_j = \emptyset$ and $W_1 \cup \ldots \cup W_m = V$). Routing costs $c_{ij}$ for $v_i, v_j \in V$ are defined. The objective is to find a tour which visits exactly once each cluster while minimizing the sum of the routing costs. In this work, we only consider symmetric cost matrices ($c_{ij} = c_{ji}$), but the algorithm could easily be generalized to the asymmetric case. In particular, the crossover operator can indifferently be applied on symmetric or asymmetric instances.

The GTSP is NP-hard, since the special case where $m = n$ (a city per cluster) is a TSP.

In Section 2, we review the literature on the GTSP. In this work, we address the resolution of the GTSP with a Genetic Algorithm using a Large Neighborhood Search crossover procedure (see (1) for a recent work on Large Neighborhood Search techniques). This algorithm is presented in Section 3. Section 4 provides a computational evaluation of our algorithm through benchmarks from the GTSPLIB (22).

## 2 State of the art

The GTSP was first introduced by Srivastava *et al.* (28) and Henry-Labordere (12), each one proposing a resolution through dynamic programming. Laporte and Norbert (13) and Laporte *et al.* (14) proposed integer programming approaches to solve exactly the GTSP. More recently, an efficient Branch & Cut solution scheme was proposed by Fischetti *et al.* (9), who provide optimal objective values for instances up to 442 nodes. They also proposed a deterministic partitioning method to obtain GTSP instances from TSP instances.

Many researches have been made to transform the GTSP to the TSP (16; 20; 6; 15; 2). Some of the resulting TSP instances have nearly the same number of nodes as the original GTSP instances. Moreover, some transformations of the GTSP into the TSP ((20)) have an important property : an optimal solution to the related TSP can be converted to an optimal solution to the GTSP. Unfortunately, a feasible non-optimal solution for the TSP may not be feasible for the GTSP. Furthermore, well-known heuristics for the TSP may not perform well for the GTSP.

In (19), Noon proposed several heuristics, including an adaptation of the nearest-neighbor heuristic developed for the resolution of the TSP. Similar adaptations have been implemented by Fischetti *et al.* in (9), such as farthest-insertion, nearest-insertion and cheapest-insertion.

More recently, Renaud and Boctor (24) proposed an heuristic called $GI^3$ (Generalized Initialization, Insertion and Improvement), which is a generalization of the $I^3$ heuristic presented in (23). This heuristic consists of three phases: an initialisation during which a partial tour is constructed, an insertion phase which completes the tour by inserting at the cheapest cost cities from unvisited clusters and an improvement phase based on 2-opt and 3-opt moves between clusters, called here G2-opt and G3-opt. They also presented the ST algorithm (for Shortest Tour). This algorithm determines the cheapest cost sequence of cities visiting the clusters in a chosen order. They showed that this problem can be solved in polynomial time.

Snyder and Daskin (27) proposed a resolution by a genetic algorithm using a random-key encoding, encoding which assures that solutions constructed by crossover or mutations are feasible solutions. The genetic algorithm was coupled with local search improvement, namely a swap procedure and a 2-opt neighborhood search, yielding a memetic algorithm. The computational results show that their algorithm is quite successful, in solution quality and computational time.

Finally, Silberholz and Golden (26) proposed a genetic algorithm with several new features which include isolated initial populations and a new reproduction mechanism, based upon the TSP ordered crossover operator. This new mechanism is called *mrOX*, for modified rotational ordered crossover. Local improvement procedures combined with this mechanism, yielding again a memetic algorithm, permit to obtain very good results on large new instances and outperform the other heuristics solutions in terms of solution quality.

A particle swarm optimization based algorithm was recently presented by Shi *et al.* in (25). An uncertainty searching strategy and technique to delete the crossover of traveling lines were used to accelerate the convergence speed, enhanced by two local search techniques.

Silberholz and Golden (26) proposed the most competitive algorithms published to date.

## 3 The Memetic Algorithm

A genetic algorithm is a search technique widely used to find approximate solutions in many optimisation problems (see (18) for further details). Genetic algorithms are categorized as metaheuristics and are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. Memetic algorithms are genetic algorithms paired with local search techniques (see (11) for further details). In this section, we present a new memetic algorithm. We particularly insist on the crossover operator, which is our main contribution. Especially, to clearly evaluate the impact of this operator, we voluntarily adopt a very standard implementation for the rest of the algorithm.

### *3.1 Individuals*

Each individual (a solution of the problem) is represented by an ordered list of clusters, where the first and last clusters are identical. From this representation, a city tour can be derived, defined as the optimal tour maintaining the visiting order of clusters. The cost of the individual is the cost of this city tour. It is obtained using the Shortest Tour algorithm.

The principle of the Shortest Tour algorithm (see (24) for details) is the following. A succession of clusters defines a sequence of sets of cities, where cities from one cluster can only be attained from cities belonging to a preceding cluster. Representing cities by nodes, we obtain a directed acyclic graph. In this graph, the subset of paths having identical starting and ending nodes exactly corresponds to the set of GTSP solutions respecting the order defined by the cluster sequence. The best solution coincides with the shortest path of this subset. Seeing that calculating the shortest path in an acyclic graph can be done in linear time with a simple recursion, the best city tour can easily be obtained. One just has to successively consider each city of the first (and last) cluster and compute the shortest path constrained to start and end with this city. The optimal city tour is the best of these solutions. This procedure is computationally cheap and, consequently, can be called very often.

### *3.2 Initial Population*

Our initial population contains $N$ individuals. Individuals are constructed through randomly generated clusters lists. The Shortest Tour algorithm is applied to determine the optimal city tour and the individual cost. In order to avoid symmetries, the first (and last) cluster is the same for all individuals;

in order to limit the computation time of the Shortest Tour procedure, the cluster containing the fewest cities is chosen.

### *3.3 Population Renewal*

At each generation, two individuals are randomly chosen through Roulette Wheel Selection and paired for crossover. These two parents breed two children. This operation is repeated $k$ times. The children are then added to the population and only the $N$ best individuals are kept.

### *3.4 Crossover operator*

The crossover operator is very important in a genetic or a memetic algorithm. This operator allows constructing new solutions from existing solutions and plays a great part in diversification. The crossover procedure we propose here is inspired from the *dropstar* procedure used in Bontoux and Feillet (3); shortly, in this paper, the context was the solution of the Traveling Purchaser Problem and *dropstar* was used as a local search operator determining the best subsequence from a sequence of cities. It is noteworthy saying that this operator is also inspired from the algorithm proposed by Prins in (21).

Let $W_{i'_1}, \ldots, W_{i'_m}$ and $v_{i_1}, \ldots, v_{i_m}$ respectively be the clusters tour and the derived cities tour of an individual, called the father. Let $W_{j'_1}, \ldots, W_{j'_m}$ and $v_{j_1}, \ldots, v_{j_m}$ respectively be these tours for another individual, called the mother. A new individual - a child - is built by the following procedure. Note that once a child has been constructed, the roles of the two parents are reversed and a second child is obtained using the same procedure.

Each mother city is inserted one by one in the father cities tour. The insertion cost of city $v_{j_k}$ in position $l$ in the father cities tour is equal to $c_{i_l j_k} + c_{j_k i_{l+1}} - c_{i_l i_{l+1}}$. We consider the insertion of the city $v_{j_k}$ in the position $l$ in the father cities tour only if cities placed before and after do not belong to the cluster of $v_{j_k}$, that means when $v_{j_k} \notin W_{i'_l}$ and $v_{j_k} \notin W_{i'_{l+1}}$. The city $v_{j_k}$ is inserted to the position which minimizes insertion costs. The order in which cities are inserted is determined by the mother cities tour, i.e. the first inserted city is the the first city belonging to the mother cities tour.

Once each mother city is inserted, we obtain a clusters tour in which each cluster appears twice. This sequence is called the master sequence (in (5), a master sequence is defined as a sequence containing at least one optimal sequence). Thus, we have to determine the optimal sub tour in which every cluster is visited exactly once. To this purpose, we adapt the *dropstar* pro-

cedure, which determines a optimal sub sequence for the resolution of the Traveling Purchaser Problem. One can expect the optimal subsequence to be difficult to compute. Actually, computing this subsequence is NP-hard.

This search is computed through a dynamic programming algorithm, applied to the graph obtained from the tour containing each cluster twice.

This graph is built by the following procedure. An edge is created for each city from each cluster, each time a cluster appears in the sequence. Vertices are added from every edge to all edges to the following clusters in the clusters tour (cf. Fig. 1 for an aggregated vision of the graph and Fig. 2 for an extract of the complete graph). The procedure consists then in finding the shortest path in the graph between the first and the last cluster of the clusters tour, with the constraint that all clusters must be visited exactly once and that the solution must be a cycle. The cost of the path is equal to the sum of the routing costs. This crossover permits to obtain the optimal sub sequence in which each cluster appears once from a sequence where each cluster appears twice.

For example, let suppose that $W = \{1, \ldots, 5\}$ is the set of clusters. let

$$W_4\ W_3\ W_1\ W_5\ W_2\ W_4$$
$$W_4\ W_1\ W_2\ W_5\ W_3\ W_4$$

be the clusters tours of the father and the mother. The tour in which each cluster appears twice may be :

$$W_4\ W_1\ W_3\ W_1\ W_5\ W_2\ W_3\ W_2\ W_5\ W_4$$

The dynamic programming algorithm determines an optimal cities sequence, which, once a clusters sequence is derived, defines a new individual.

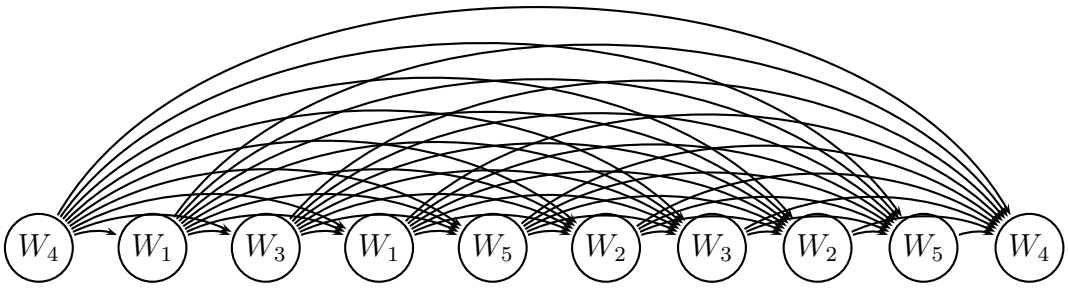


Figure 1. Example of crossover and corresponding graph used by the *dropstar* procedure : aggregated view of clusters

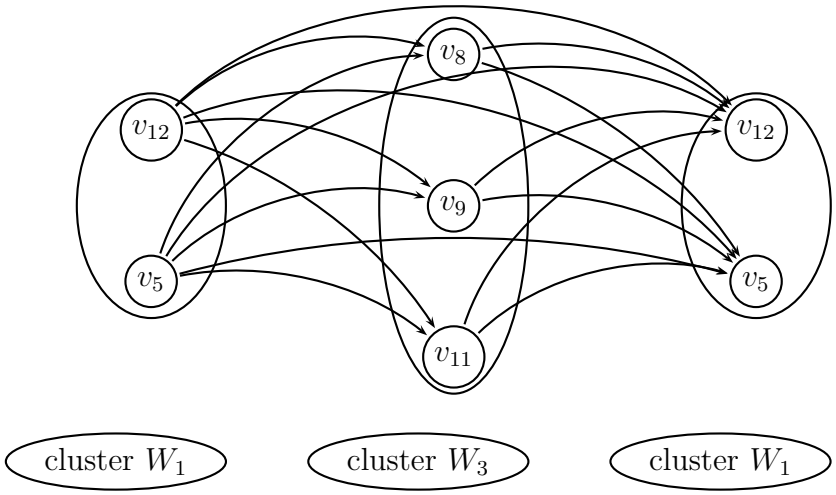


Figure 2. Example of crossover and corresponding graph used by the *dropstar* procedure : precise view

The crossover procedure permits to choose the best individual among a very large set of individuals; it allows an interesting diversification in the memetic algorithm and ensures populations of good quality.

### *3.4.1 Dominance Rules*

The dynamic programming algorithm used to find the shortest path in the graph is inspired from the algorithm developed in Feillet *et al.* (7) for the Elementary Shortest Path Problem with Resource Constraints This algorithm is an extension of the classical Bellman's labelling algorithm. Our algorithm starts from the first cluster and constructs iteratively paths, with the constraint that every cluster has to be visited exactly once. Solutions are built through partial paths extension. The algorithm maintains a set of labels corresponding to partial paths. Extension of a label $l$ consists in creating a label $l'$ by adding to the partial path an outgoing edge to any city belonging to a non visited cluster. In order to limit the number of labels, we propose to apply dominance rules.

In the following, we note $L = (C, v_{i_1}, \ldots, v_{i_k})$ a label corresponding to a partial path, where $C$ represents the routing cost of the partial path and $v_{i_j}$ the city $v_i$ visited in the position $j$. It is easy to know which clusters have been visited by the partial path since each city belongs to a cluster. A label $L^1$ dominates a label $L^2$, which is noted $L^1 < L^2$, when the two partial paths represented by

these labels lead to the same vertex and one can be sure that any extension of $L^1$ is going to be cheaper than the identical extension for $L^2$. In this problem, $L^1$ dominates the label $L^2$ if and only if all the clusters visited by $L^2$ have also been visited by $L^1$ and if the cost of $L^1$ is lower or equal to the cost of $L^2$. In that case, $L^1$ dominates $L^2$ and $L^2$ can be deleted.

### *3.4.2 Lower Bounds*

Each time a label is extended to a city, a lower bound is computed. If this lower bound is greater than the cost of the father (which is an upper bound), the label is deleted; we can note that the father clusters tour belongs to the master sequence sub set, while it is not the case for the mother clusters tour. As a consequence, the cost of the father is an upper bound. Furthermore, if a complete solution with a better cost than the father solution cost is found during the resolution of the crossover, the upper bound is updated.

The lower bound is computed as follows. Since each cluster has to be visited, a visiting cost is added to the actual cost of the label for each cluster that has not been visited yet by the label. The visiting cost for the cluster $j$ is the minimal cost to reach any city belonging to the cluster $j$, starting from any cluster located between the position of the last cluster visited by the label and the position of the cluster $j$ last occurence in the sequence.

The visiting cost is computed once for all, as soon as the master sequence is set. It is computed for each cluster and for each position in the sequence.

For example, let the sequence be :

$$W_4\ W_1\ W_3\ W_1\ W_5\ W_2\ W_3\ W_2\ W_5\ W_4$$

We compute the vising cost at the position 5 for each cluster. We only consider clusters which have at least one occurence located after the position 5. At the position 5 (corresponding to the cluster $W_5$), the visiting cost of the cluster $W_1$ is undefined since the cluster $W_1$ is no longer reachable (all occurences of the cluster $W1$ are located before the position 5). The visiting cost of the cluster $W_2$ is the minimal cost to reach any city belonging from the cluster $W_2$ starting from the cluster $W_5$ or the cluster $W_3$ (the cluster $W1$ is not considered, since its last occurence is located before the position 5). The visiting cost of the cluster $W_3$ is the minimal cost to reach any city belonging from the cluster $W_3$ starting from the cluster $W_5$ or the cluster $W_2$. Finally, the visiting cost of the cluster $W_4$ is the minimal cost to reach any city belonging from the cluster $W_4$ starting from the cluster $W_5$, the cluster $W_2$ or the cluster $W_3$.

These lower bounds permit to limit the number of labels and therefore to speed up the resolution of the crossover procedure.

### *3.4.3 Graph Reduction*

Since every cluster has to be visited, no edge is allowed to skip all occurences of a cluster. Moreover, two occurences of a cluster cannot be connected. Finally, a cluster at position $j$ in the master sequence should be connected only to the first occurence of any other cluster, if both occurences are located after position $j$. Fig.3 presents the graph obtained from the previous example, applying these rules.

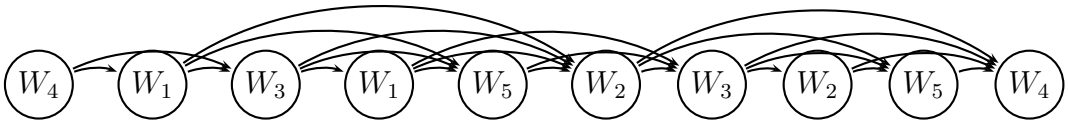


Figure 3. Example of crossover and corresponding reduced graph used by the *drop-star* procedure : aggregated view of clusters

### *3.4.4 No longer reachable clusters*

For each position on the master sequence, a list of no longer reachable clusters can be easily computed (a cluster is no longer reachable if its latest position in the sequence is located before the current position). When a label is extended to a cluster located on the position $l$, a simple procedure controls if every cluster on the list attached to the position $l$ has been visited by the label. If this is not the case, the label will not be able to construct a valid solution, and is therefore deleted.

### *3.4.5 Heuristic crossover variant*

The crossover procedure is a very long procedure. In order to speed up its resolution, we proposed three heuristics crossover variants.

**Limitation of the labels list sizes :** In our dynamic programming algorithm, a list of labels is associated to every city. Despite of the dominance rules, the size of these lists may be significant. The goal is therefore to limit the number of labels in the list. To this purpose, an evaluation is associated to a label. Firstly, once the master sequence is set, a coefficient called $\text{coeff}_{eval}$ is computed. This coefficient is equal to the ratio between the cost of the father solution and the lower bound at the beginning of the master sequence (i.e. when no cluster has been visited yet). The label evaluation is then equal to the cost of the label partial path, added to the sum of the

visiting cost of non visited clusters, multiplied by $\text{coeff}_{eval}$.

$$\text{evaluation} = \text{cost} + \sum_{\text{cluster } i \text{ non visited}} \text{visiting cost}(i) * \text{coeff}_{eval}$$

The formula avoids the preference of labels that have not visited many clusters (i.e. with a very low lower bound). This evaluation permits to establish an ordered list of labels. Once the labels list has reached the limit size, labels with the worst evalution are removed. In our experiments, the limit size of the lists has been set to 100.

**Limitation of the city number :** An estimation of the insertion cost in the master sequence is computed for every city. The estimation for the city $v_{j_k}$ in the position $l$ in the master sequence is computed as follows :

$$\text{estimation} = \sum_{i_a \in (l-2,l-1) \text{ and } i_b \in (l+1,l+2)} c_{i_a j_k} + c_{j_k i_b} - c_{i_a i_b}$$

In order to limit the number of cities in each cluster, the maximum size of the cluster is set to $\lceil \text{size}_i^{\text{coeff}_{city}} \rceil$ where $\text{size}_i$ is the size of the cluster $i$. If the maximum size of the cluster is reached, the cities with the greatest insertion cost are removed. In our experiments, $\text{coeff}_{city}$ has been set to 0.8.

### *3.5 Mutation*

A mutation procedure is applied to improve the population diversity. Each individual has a 5% probability of being selected for mutation. The mutation consists in swapping two randomly choosen clusters and applying the Shortest Tour algorithm to compute the optimal city tour and the new individual cost.

### *3.6 Local search heuristic*

The procedures presented here are applied on the city tour obtained from the crossover as long as some improvements are reached. Each time a better city tour is obtained, the corresponding cluster sequence is extracted and the Shortest Tour algorithm is applied to compute the optimal city tour and the new individual cost.

#### *3.6.1 2-opt*

This procedure is well-known in the context of the TSP (see (17) for more details). The move consists in choosing two arcs in the cities tour, permutating the circulation between the ending vertices of these arcs and reconnecting the

tour. The complexity of the procedure is $O(m^2)$ where $m$ is the number of clusters. The Shortest Tour algorithm is applied once the procedure is over, i.e when no improvrement is reached.

### *3.6.2 3-opt*

Similar to the *2-opt*, the 3-opt (presented also in (17))chooses three arcs in the tour and de-interlace the path between the ending cities of these arcs. The complexity of the procedure is $O(m^3)$. The Shortest Tour algorithm is applied once the procedure is over, i.e when no improvrement is reached.

### *3.6.3 Move*

This procedure consists in removing a cluster, duplicating it $m-1$ times and inserting it between all clusters. On this sequence in which one cluster appears $m-1$ times, a procedure similar to the one used for the crossover is applied. This procedure determines the best position for the cluster, computing the shortest path in a graph with the constraint that each cluster must be visited only once. This procedure is called for every cluster, as long as some improvements are reached. Due to its complexity, the procedure is called when a new best solution to the problem has been reached.

Let

$$W_4\ W_3\ W_1\ W_5\ W_2\ W_4$$

be the clusters tours of an individual. The *move* procedure is applied, inserting the cluster $W_1$. The sequence obtained is the following :

$$W_4\ W_1\ W_3\ W_1\ W_5\ W_1\ W_2\ W_1\ W_4$$

Fig.4 presents the resulting graph on which the *dropstar* procedure is applied.

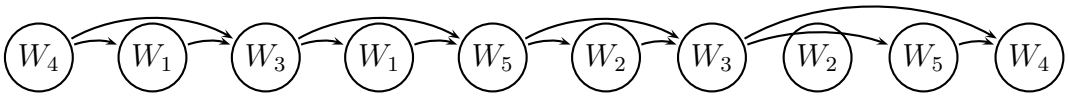


Figure 4. Example of the move operator and corresponding reduced graph : aggregated view of clusters

### *3.7 Stopping criteria*

The heuristic stops when $Nb1$ generations have been computed or when no improvement has been made during $Nb2$ generations.

### *3.8 Memetic Algorithm heuristic*

Figure 1 presents a synthetic view of our algorithm.

**Algorithm 1** Memetic Algorithm

```
Compute an initial population of N random individuals
while the number of iterations is lower than Nb1 or no improvement has
occurred for Nb2 iterations do
  for i = 0 to k do
    Choose 2 individuals randomly
    Construct 2 children with crossover
    Apply local search on both children
    Add children to the population
  end for
  Keep the N best individuals in the population
  Apply mutation with a 5% probability.
end while
```

## 4 Computational Results

The algorithm was coded in C++ and run on an Intel Pentium IV 2.00 Ghz under Linux/Debian. Instances used are part of the GTSPLIB library [1] which proposes a set of 65 instances. Among these instances, we have selected 41 instances used in several papers ((27; 24; 9)) in order to compare our algorithm to others. Presented results are the mean results obtained through 5 attempts for each instance.

Fischetti *et al.* (9) proposed a Branch & Cut algorithm to solve the symmetric GTSP. In their work, they derive test problems by applying a deterministic partitioning method to 46 standard TSP instances from the TSPLIB library. For a given instance, the number of clusters is fixed to $m = \lceil n/5 \rceil$. Then, $m$ centers are determined by considering $m$ nodes as far as possible from each other. The clusters are finally obtained by assigning each node to its nearest center. They provide optimal objective values for each of the problems.

[1] GTSPLIB is available at the address http://www.cs.rhul.ac.uk/home/zvero/GTSPLIB/

For all computational tests, we set the number of individuals per population ($N$) to 50, the number of crossovers ($2 * k$) equal to 30, the maximum number of iterations ($Nb1$) to 100 and the maximum number while the best solution cost stays unchanged ($Nb2$) equal to 5.

Table 1 presents the performances of our algorithm regarding the gap with optimal solutions provided by the Branch & Cut algorithm from Fischetti *et al.* (9). The column headings are defined as follows:

- instance: the name of the test problem; the digits at the beginning of the name give the number of clusters, those at the end give the number of nodes;
- opt: the optimal objective value for the problem;
- best: the number of trials, out of five, for which our algorithm found the optimal solution;
- mean gap: the mean gap of our algorithm above the optimum (percentage);
- min gap: the minimal gap of our algorithm above the optimum (percentage);
- max gap: the maximal gap of our algorithm above the optimum (percentage).

Results written in bold represent cases for which the solution we found is equal to the optimal one.

Table 1
Experimental results : quality of the solutions

| instance | opt | best | mean gap | min gap | max gap |
|---|---|---|---|---|---|
| 10att48.gtsp | 5394 | 5 | **0.00** | **0.00** | **0.00** |
| 10gr48.gtsp | 1834 | 5 | **0.00** | **0.00** | **0.00** |
| 10hk48.gtsp | 6386 | 5 | **0.00** | **0.00** | **0.00** |
| 11eil51.gtsp | 174 | 5 | **0.00** | **0.00** | **0.00** |
| 12brazil58.gtsp | 15332 | 5 | **0.00** | **0.00** | **0.00** |
| 14st70.gtsp | 316 | 5 | **0.00** | **0.00** | **0.00** |
| 16eil76.gtsp | 209 | 5 | **0.00** | **0.00** | **0.00** |
| 16pr76.gtsp | 64925 | 5 | **0.00** | **0.00** | **0.00** |
| 20kroA100.gtsp | 9711 | 5 | **0.00** | **0.00** | **0.00** |
| 20kroB100.gtsp | 10328 | 5 | **0.00** | **0.00** | **0.00** |
| 20kroC100.gtsp | 9554 | 5 | **0.00** | **0.00** | **0.00** |
| 20kroD100.gtsp | 9450 | 5 | **0.00** | **0.00** | **0.00** |
| 20kroE100.gtsp | 9523 | 5 | **0.00** | **0.00** | **0.00** |
| 20rat99.gtsp | 497 | 5 | **0.00** | **0.00** | **0.00** |
| 20rd100.gtsp | 3650 | 5 | **0.00** | **0.00** | **0.00** |
| 21eil101.gtsp | 249 | 5 | **0.00** | **0.00** | **0.00** |
| 21lin105.gtsp | 8213 | 5 | **0.00** | **0.00** | **0.00** |
| 22pr107.gtsp | 27898 | 5 | **0.00** | **0.00** | **0.00** |
| 24gr120.gtsp | 2769 | 5 | **0.00** | **0.00** | **0.00** |
| 25pr124.gtsp | 36605 | 5 | **0.00** | **0.00** | **0.00** |
| 26bier127.gtsp | 72418 | 5 | **0.00** | **0.00** | **0.00** |
| 28pr136.gtsp | 42570 | 5 | **0.00** | **0.00** | **0.00** |
| 29pr144.gtsp | 45886 | 5 | **0.00** | **0.00** | **0.00** |
| 30kroA150.gtsp | 11018 | 5 | **0.00** | **0.00** | **0.00** |
| 30kroB150.gtsp | 12196 | 5 | **0.00** | **0.00** | **0.00** |
| 31pr152.gtsp | 51576 | 5 | **0.00** | **0.00** | **0.00** |
| 32u159.gtsp | 22664 | 5 | **0.00** | **0.00** | **0.00** |
| 39rat195.gtsp | 854 | 5 | **0.00** | **0.00** | **0.00** |
| 40d198.gtsp | 10557 | 5 | **0.00** | **0.00** | **0.00** |
| 40kroA200.gtsp | 13406 | 5 | **0.00** | **0.00** | **0.00** |
| 40kroB200.gtsp | 13111 | 5 | **0.00** | **0.00** | **0.00** |
| 45ts225.gtsp | 68340 | 5 | **0.00** | **0.00** | **0.00** |
| 46pr226.gtsp | 64007 | 5 | **0.00** | **0.00** | **0.00** |
| 53gil262.gtsp | 1013 | 4 | 0.02 | **0.00** | 0.10 |
| 53pr264.gtsp | 29549 | 5 | **0.00** | **0.00** | **0.00** |
| 60pr299.gtsp | 22615 | 5 | **0.00** | **0.00** | **0.00** |
| 64lin318.gtsp | 20765 | 3 | 0.38 | **0.00** | 1.02 |
| 80rd400.gtsp | 6361 | 2 | 0.78 | **0.00** | 2.09 |
| 84fl417.gtsp | 9651 | 5 | **0.00** | **0.00** | **0.00** |
| 88pr439.gtsp | 60099 | 4 | 0.10 | **0.00** | 0.50 |
| 89pcb442.gtsp | 21657 | 1 | 0.80 | **0.00** | 1.99 |

Table 1 shows that, with 5 attempts, 41 instances out of 41 are optimally solved and that for 36 of these instances, the optimal solution is found at each run of the memetic algorithm. The difference between the best and the worst solution returned from the 5 trials always remains small , which seems to indicate that our algorithm is robust: 38 instance are solved with a maximal gap lesser than 1 %, the gap never exceeds 2.09 % even on large instances.

Table 2 gives information about running times and iterations. The column headings are as follows :

- instance: the name of the test problem;
- best: the number of trials, out of five, for which our algorithm found the optimal solution;
- mean time: the mean CPU time in seconds;
- min time: the minimum CPU time in seconds;
- max time: the maximum CPU time in seconds;
- iterations: the mean numbers of iterations.

Table 2
Experimental results: CPU time and iterations numbers

| instance | best | mean time | min time | max time | iterations |
|---|---|---|---|---|---|
| 10att48.gtsp | 5 | 0.29 | 0.21 | 0.44 | 6 |
| 10gr48.gtsp | 5 | 0.22 | 0.2 | 0.24 | 6 |
| 10hk48.gtsp | 5 | 0.25 | 0.17 | 0.37 | 6 |
| 11eil51.gtsp | 5 | 0.29 | 0.26 | 0.33 | 6 |
| 12brazil58.gtsp | 5 | 0.25 | 0.23 | 0.28 | 6.2 |
| 14st70.gtsp | 5 | 0.32 | 0.26 | 0.45 | 6.2 |
| 16eil76.gtsp | 5 | 0.36 | 0.32 | 0.44 | 6 |
| 16pr76.gtsp | 5 | 0.52 | 0.37 | 0.61 | 6.2 |
| 20kroA100.gtsp | 5 | 0.95 | 0.58 | 1.56 | 6.2 |
| 20kroB100.gtsp | 5 | 1.41 | 1.09 | 1.9 | 6.4 |
| 20kroC100.gtsp | 5 | 1.07 | 0.69 | 1.33 | 6 |
| 20kroD100.gtsp | 5 | 1.09 | 0.84 | 1.35 | 6 |
| 20kroE100.gtsp | 5 | 1.3 | 0.79 | 1.72 | 6.6 |
| 20rat99.gtsp | 5 | 2.85 | 2.25 | 3.58 | 6.8 |
| 20rd100.gtsp | 5 | 3.1 | 2.12 | 3.71 | 7.8 |
| 21eil101.gtsp | 5 | 1.54 | 1.36 | 1.67 | 7.4 |
| 21lin105.gtsp | 5 | 1.09 | 0.89 | 1.48 | 6 |
| 22pr107.gtsp | 5 | 2.6 | 2.22 | 3.17 | 6.2 |
| 24gr120.gtsp | 5 | 3.35 | 2.32 | 4.41 | 7.4 |
| 25pr124.gtsp | 5 | 3.21 | 2.29 | 4.05 | 6.6 |
| 26bier127.gtsp | 5 | 4.04 | 2.6 | 5.79 | 6.6 |
| 28pr136.gtsp | 5 | 6.61 | 4.97 | 8.62 | 7.6 |
| 29pr144.gtsp | 5 | 6.47 | 5.01 | 8.24 | 8.4 |
| 30kroA150.gtsp | 5 | 11.57 | 8.37 | 13.64 | 7 |
| 30kroB150.gtsp | 5 | 12.35 | 11.49 | 13.2 | 7.6 |
| 31pr152.gtsp | 5 | 7.03 | 5.03 | 8.84 | 7.6 |
| 32u159.gtsp | 5 | 10.22 | 6.68 | 12.95 | 6.8 |
| 39rat195.gtsp | 5 | 43.07 | 25.29 | 61.8 | 8.4 |
| 40d198.gtsp | 5 | 22 | 15.11 | 28.1 | 7.4 |
| 40kroA200.gtsp | 5 | 39.42 | 31.5 | 45.66 | 9 |
| 40kroB200.gtsp | 5 | 36.88 | 31.93 | 41.37 | 8.4 |
| 45ts225.gtsp | 5 | 108.77 | 52.2 | 168.9 | 11.8 |
| 46pr226.gtsp | 5 | 10.16 | 8.12 | 13.26 | 6.4 |
| 53pr264.gtsp | 5 | 75.44 | 58.99 | 94.29 | 8.4 |
| 53gil262.gtsp | 4 | 131.4 | 108.22 | 157.9 | 11.6 |
| 60pr299.gtsp | 5 | 139.77 | 109.64 | 163.15 | 9.4 |
| 64lin318.gtsp | 3 | 142.63 | 71.53 | 183.01 | 12 |
| 80rd400.gtsp | 2 | 260.48 | 220.78 | 305.19 | 9.2 |
| 84fl417.gtsp | 5 | 41.95 | 21.21 | 55.94 | 8.2 |
| 88pr439.gtsp | 4 | 352.55 | 302.01 | 360.86 | 14.8 |
| 89pcb442.gtsp | 1 | 307.53 | 292.23 | 387.46 | 14.4 |

The results presented in Table 2 show that our algorithm can solve optimally 24 out of the 41 instances with a running time below 10 seconds (at each run). Actually, in most cases, the optimal solution is found during the very first iterations of the algorithm and most of the time is spent to trigger the stopping criteria (5 iterations without improvement). The CPU time to solve the instances is more significant for large instances, whitout exceeding 400 secondes.

Table 3 presents a comparison between the memetic algorithm from Snyder (27), the GI$^3$ of Renaud *et al.* (24), the 2 heuristics (Lagrangian and "root-node") presented by Fischetti, Salazar-Gonzáles and Toth (9) and the Branch & Cut algorithme from Fischetti *et al.* presented in (9).

The results have been obtained on the following computers:

- Snyder : Pentium IV 3.2 GHz processor and 1 GB RAM.
- GI$^3$ and NN: Sun Sparc Station LX.
- FST-Lagr. , FST-Root. and B& C. : HP 9000/720.

For each algorithm, two columns are presented in the table:

- Gap: the mean gap of the algorithm above the optimal solutions (percentage);
- CPU: the CPU time in seconds.

Table 3
Results for several algorithms

| | Bontoux | | Snyder | | GI | | Noon | | FST-Lagr | | FST-Root | | BC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| instance | Gap | CPU | Gap | CPU | Gap | CPU | Gap | CPU | Gap | CPU | Gap | CPU | CPU |
| 10att48.gtsp | **0.00** | 0.29 | **0.00** | **0.00** | | | | | **0.00** | 0.90 | **0.00** | 2.10 | 2.10 |
| 10gr48.gtsp | **0.00** | 0.22 | **0.00** | 0.50 | | | | | **0.00** | 0.50 | **0.00** | 1.90 | 1.90 |
| 10hk48.gtsp | **0.00** | 0.25 | **0.00** | 0.20 | | | | | **0.00** | 1.10 | **0.00** | 3.80 | 3.80 |
| 11eil51.gtsp | **0.00** | 0.29 | **0.00** | 0.10 | **0.00** | 0.30 | **0.00** | 0.40 | **0.00** | 0.40 | **0.00** | 2.90 | 2.90 |
| 12brazil58.gtsp | **0.00** | 0.25 | **0.00** | 0.30 | | | | | **0.00** | 1.40 | **0.00** | 3.00 | 3.00 |
| 14st70.gtsp | **0.00** | 0.32 | **0.00** | 0.20 | **0.00** | 1.70 | **0.00** | 0.80 | **0.00** | 1.20 | **0.00** | 7.30 | 7.30 |
| 16eil76.gtsp | **0.00** | 0.36 | **0.00** | 0.20 | **0.00** | 2.20 | **0.00** | 1.10 | **0.00** | 1.40 | **0.00** | 9.40 | 9.40 |
| 16pr76.gtsp | **0.00** | 0.52 | **0.00** | 0.20 | **0.00** | 2.50 | **0.00** | 1.90 | **0.00** | 0.60 | **0.00** | 12.90 | 12.90 |
| 20kroA100.gtsp | **0.00** | 0.95 | **0.00** | 0.40 | **0.00** | 6.80 | **0.00** | 3.80 | **0.00** | 2.40 | **0.00** | 18.30 | 18.40 |
| 20kroB100.gtsp | **0.00** | 1.41 | **0.00** | 0.40 | **0.00** | 6.40 | **0.00** | 2.40 | **0.00** | 3.10 | **0.00** | 22.10 | 22.20 |
| 20kroC100.gtsp | **0.00** | 1.07 | **0.00** | 0.30 | **0.00** | 6.50 | **0.00** | 3.60 | **0.00** | 2.20 | **0.00** | 14.30 | 14.40 |
| 20kroD100.gtsp | **0.00** | 1.09 | **0.00** | 0.40 | **0.00** | 8.60 | **0.00** | 5.40 | **0.00** | 2.50 | **0.00** | 14.20 | 14.30 |
| 20kroE100.gtsp | **0.00** | 1.3 | **0.00** | 0.80 | **0.00** | 6.70 | **0.00** | 2.80 | **0.00** | 0.90 | **0.00** | 12.90 | 13.00 |
| 20rat99.gtsp | **0.00** | 2.85 | **0.00** | 0.70 | **0.00** | 5.00 | **0.00** | 7.30 | **0.00** | 3.10 | **0.00** | 51.40 | 54.50 |
| 20rd100.gtsp | **0.00** | 3.1 | **0.00** | 0.30 | 0.80 | 7.30 | 0.08 | 8.30 | **0.00** | 2.60 | **0.00** | 16.50 | 16.60 |
| 21eil101.gtsp | **0.00** | 1.54 | **0.00** | 0.20 | 0.40 | 5.20 | 0.40 | 3.00 | **0.00** | 1.70 | **0.00** | 25.50 | 25.60 |
| 21lin105.gtsp | **0.00** | 1.09 | **0.00** | 0.30 | **0.00** | 14.40 | **0.00** | 3.70 | **0.00** | 2.00 | **0.00** | 16.20 | 16.40 |
| 22pr107.gtsp | **0.00** | 2.6 | **0.00** | 0.40 | **0.00** | 8.70 | **0.00** | 5.20 | **0.00** | 2.10 | **0.00** | 7.30 | 7.40 |
| 24gr120.gtsp | **0.00** | 3.35 | **0.00** | 0.50 | | | | | 1.99 | 4.90 | **0.00** | 41.80 | 41.90 |
| 25pr124.gtsp | **0.00** | 3.21 | **0.00** | 0.60 | 0.43 | 12.20 | **0.00** | 12.00 | **0.00** | 3.70 | **0.00** | 25.70 | 25.90 |
| 26bier127.gtsp | **0.00** | 4.04 | **0.00** | 0.50 | 5.55 | 36.10 | 9.86 | 7.80 | **0.00** | 11.20 | **0.00** | 23.30 | 23.60 |
| 28pr136.gtsp | **0.00** | 6.61 | **0.00** | 0.50 | 1.28 | 12.50 | 5.54 | 9.60 | 0.82 | 7.20 | **0.00** | 42.80 | 43.00 |
| 29pr144.gtsp | **0.00** | 6.47 | **0.00** | 0.30 | **0.00** | 16.30 | **0.00** | 11.80 | **0.00** | 2.30 | **0.00** | 8.00 | 8.20 |
| 30kroA150.gtsp | **0.00** | 11.57 | **0.00** | 1.30 | **0.00** | 17.80 | **0.00** | 22.90 | **0.00** | 7.60 | **0.00** | 10**0.00** | 100.30 |
| 30kroB150.gtsp | **0.00** | 12.35 | **0.00** | 1.00 | **0.00** | 14.20 | **0.00** | 20.10 | **0.00** | 9.90 | **0.00** | 60.30 | 60.60 |
| 31pr152.gtsp | **0.00** | 7.03 | **0.00** | 1.50 | 0.47 | 17.60 | 1.80 | 10.30 | **0.00** | 9.60 | **0.00** | 51.40 | 94.80 |
| 32u159.gtsp | **0.00** | 10.22 | **0.00** | 0.60 | 2.60 | 18.50 | 2.79 | 26.50 | **0.00** | 10.90 | **0.00** | 139.60 | 146.40 |
| 39rat195.gtsp | **0.00** | 43.07 | **0.00** | 0.70 | **0.00** | 37.20 | 1.29 | 86.00 | 1.87 | 8.20 | **0.00** | 245.50 | 245.90 |
| 40d198.gtsp | **0.00** | 22 | **0.00** | 1.20 | 0.60 | 60.40 | 0.60 | 118.80 | 0.48 | 12.00 | **0.00** | 762.50 | 763.10 |
| 40kroA200.gtsp | **0.00** | 39.42 | **0.00** | 2.70 | **0.00** | 29.70 | 5.25 | 53.00 | **0.00** | 15.30 | **0.00** | 183.30 | 187.40 |
| 40kroB200.gtsp | **0.00** | 36.88 | **0.00** | 1.40 | **0.00** | 35.80 | **0.00** | 135.20 | 0.05 | 19.10 | **0.00** | 268.00 | 268.50 |
| 45ts225.gtsp | **0.00** | 108.77 | **0.00** | 2.40 | 0.61 | 89.00 | **0.00** | 117.80 | 0.09 | 19.40 | 0.09 | 1298.40 | 37875.90 |
| 46pr226.gtsp | **0.00** | 10.16 | **0.00** | 1.00 | **0.00** | 25.50 | 2.17 | 67.60 | **0.00** | 14.60 | **0.00** | 106.20 | 106.90 |
| 53gil262.gtsp | 0.02 | 131.4 | 0.79 | 1.90 | 5.03 | 115.40 | 1.88 | 122.70 | 3.75 | 15.80 | 0.89 | 1443.50 | 6624.10 |
| 53pr264.gtsp | **0.00** | 75.44 | **0.00** | 1.30 | 0.36 | 64.40 | 5.76 | 147.20 | 0.33 | 24.30 | **0.00** | 336.00 | 337.00 |
| 60pr299.gtsp | **0.00** | 139.77 | 0.02 | 6.10 | 2.23 | 90.30 | 2.01 | 281.80 | **0.00** | 33.20 | **0.00** | 811.40 | 812.80 |
| 64lin318.gtsp | 0.38 | 142.63 | **0.00** | 3.50 | 4.59 | 206.80 | 4.92 | 317.00 | 0.36 | 52.50 | 0.36 | 847.80 | 1671.90 |
| 80rd400.gtsp | 0.78 | 260.48 | 1.37 | 3.50 | 1.23 | 103.50 | 3.98 | 1137.10 | 3.16 | 59.80 | 2.97 | 5031.50 | 7021.40 |
| 84fl417.gtsp | **0.00** | 41.95 | 0.07 | 2.40 | 0.48 | 427.10 | 1.07 | 1341.00 | 0.13 | 77.20 | **0.00** | 16714.40 | 16719.40 |
| 88pr439.gtsp | 0.10 | 352.55 | 0.23 | 9.10 | 3.52 | 611.00 | 4.02 | 1238.90 | 1.42 | 146.60 | **0.00** | 5418.90 | 5422.80 |
| 89pcb442.gtsp | 0.80 | 307.53 | 1.31 | 10.10 | 5.91 | 567.70 | 0.22 | 838.40 | 4.22 | 78.80 | 0.29 | 5353.90 | 58770.50 |

The results presented in the table 3 show that our algorithm seems at least as efficient as the genetic algorithm proposed by Snyder. Snyder algorithm is much more faster, but returns solutions with higher gaps when the instance size increases. Others algorithms seems to be slower or to return worse solutions. We can also notice that for the smallest instances, optimal solutions are quickly reached whatever the algorithm used.

## 5 Conclusions and future research

In this paper, we proposed to solve the GTSP using a memetic algorithm where the crossover operator relies on large neighborhood search. Our main contribution is the originality of our crossover procedure. Experimental results show that our algorithm is robust and presents a good balance between CPU time and quality of the solutions. 40 of the 41 problems are solved optimally and the gap between returned and optimal solutions never exceeds 2.09 %.

However, experimental results do not seem completely relevant in order to discuss the efficiency of our algorithm and to compare to other approaches proposed in the literature. Many instances may be easily solved to the optimum. However, the crossover procedure should be quickened to be efficient on larger instances.

## References


[1] Ahuja R. K., Ergun Ö., and Orlin J. B., Punnen A. P.: A survey of very large-scale neighborhood search techniques. Discrete Applied Mathematics, vol.123, p. 75102 (2002)

[2] Ben-Arieh D., Gutin G., Penn M., Yeo A., Zverovitch A.: Transformations of generalized ATSP into ATSP. Operations Research Letters, vol. 31, p.357-365 (2003)

[3] Bontoux B., Feillet D.: Ant Colony Optimization for the Traveling Purchaser Problem. Computer & Operations Research, vol. 35, no 2, p. 628-637 (2008)

[4] Chentsov A.G., Korotayeva L.N.,: The Dynamic Programming Method in the Generalized Traveling Salesman Problem. Math. Comput. Modelling, vol. 25, no. 1, p. 93-105, (1997)

[5] Dauzère-Pérès S., Sevaux M.: An exact method to minimize the number of tardy jobs in single machine scheduling. Journal of Scheduling, vol. 7, p. 405-420 (2004)

[6] Dimitrijevic V., Saric Z.: An Efficient Transformation of the Generalized

Traveling Salesman Problem into the Traveling Salesman Problem on Digraphs. Information Sciences, 102, p. 105-110, (1997)
[7] Feillet D., Dejax P., Gendreau M., Guegen C.: An exact algorithm for the Elementary Shortest Path Problem with Ressource Constraints : application to some vehicle routing problems. Networks, vol. 44, p. 216-229 (2004)
[8] Fischetti M., Salazar-Gonzáles J.J., Toth P.: The Symmetric Generalized Traveling Salesman Polytope. Networks, vol. 26(2), p. 113-123 (1995)
[9] Fischetti M., Salazar-Gonzáles J.J., Toth P.: A Branch-and-Cut algorithm for the Symmetric Generalized Traveling Salesman Problem. Operationas Research, vol. 45(3), p. 378-394 (1997)
[10] Garey M.R., Johnson D.: Computer and Intractability : A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, (1979)
[11] Hart W. E., Krasnogor N., Smith J. E.: Recent Advances in Memetic Algorithms. Springer, (2005)
[12] Henry-Labardere A.L.: The record balancing problem : A dynamic programming solution of a generalized traveling salesman problem RAIRO, vol. B2, p.43-49 (1969)
[13] Laporte G., Nobert Y.: Generalized Travelling Salesman Problem through n Sets of Nodes : An integer programming approach. INFOR 21, vol. 1, p. 61-75 (1984)
[14] Laporte G., Mercure H., Nobert Y.: Generalized Travelling Salesman Problem through n Sets of Nodes : the Asymmetrical Case. Discrete Applied Mathematics, vol. 18, p. 185-197 (1984)
[15] Laporte G., Semet F.: Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. INFOR 37, vol. 2, p.114-120 (1999)
[16] Lien Y-N., Ma E., Wah B.: Transformation of the Generalized Traveling-Salesman Problem into the Standard Traveling-Salesman Problem Information Sciences, 74, p. 177-189 (1993)
[17] Lin S.: Computer solutions of the traveling salesman problem. Bell Systems Journal, 44, p. 2245-2269 (1965)
[18] Man K.F., Tang K.S. , Kwong S.: Genetic Algorithms: Concepts and Designs Springer (1999)
[19] Noon C.E., Bean J.C.: A Lagrangian Based Approach for the Asymmetric Generalized Traveling Salesman Problem. Operations Research, vol. 39, no. 4, (1991)
[20] Noon C.E., Bean J.C.: An Efficient Transformation of the Generalized Traveling Salesman Problem. INFOR, vol. 31, no. 1 (1993)
[21] Prins C.: A simple and effective evolutionary algorithm for the VRP. Computers & Operations Research, vol. 31, no. 12, p. 1985-2002 (2004)
[22] Reinelt G.: TSPLIB - A Traveling Salesman Problem Library. ORSA Journal on Computing 3, p.376-384 (1991)
[23] Renaud, J., Boctor, F.F., Laporte, G.: A fast composite heuristic for the symmetric traveling salesman problem. INFORMS, Journal on Comput-

ing, vol. 8, p. 134-143 (1996)
[24] Renaud J., Boctor F.F.: An Efficient Composite Heuristic for the Symmetric Generalized Traveling Salesman Problem. European Journal of Operational Research, vol. 108, p. 571-584 (1998)
[25] Shi X.H. , Liang Y.C., Lee H.P., Lu C., Wang Q.X.: Particle swarm optimization-based algorithms for TSP and generalized TSP. Information Processing Letters, vol. 103(5), p. 169-176 (2007)
[26] Silberholz J., Golden B.: The Generalized Traveling Salesman Problem: A New Genetic Algorithm Approach. Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies , p. 165-181 (2007)
[27] Snyder L.V., Daskin M.S.: A Random-Key Genetic Algorithm for the Generalized Traveling Salesman Problem. European Journal of Operational Research, vol. 174, p. 38-53 (2006)
[28] Srivastava S.S.S., Kumar R.C.G., Sen P.: Generalized Traveling Salesman Problem through $n$ sets of nodes. CORS Journal, vol. 7, p. 97-101 (1969)