A Comparative Study of Genetic Algorithms, Simulated Annealing and Tabu Search for Non-fixed Destination Multi-depot Multiple Traveling Salesman Problem

Fereydoun Adbesh, Kamran Kardel

Department of Industrial Engineering, Mazandaran University of Science and Technology, Babol, P.O. Box: 743, P.O. Code: 47166-85635, Tabarsi st., Babol, Iran

Abstract

The non-fixed destination multi depot multiple traveling salesmen problem (MmTSP) in which more than one salesmen depart from several starting cities (depots) and having returned to the one of starting city (depot) that the number of cities in each depot remain the same at the end as it was in the beginning, form tours so that each city is visited with exactly one salesman and the tour lengths stay within certain limits. This problem is of a great complexity and few investigations have been done on it before. In this paper we apply three meta-heuristics algorithms as genetic algorithms (GAs), tabu search (TS) and simulated annealing (SA) for this problem and compare the theoretical properties and computational performance of these algorithms with the optimal answers obtained by solving the problems by Lingo 8. Computational analysis shows that the SA and TS have results very close to optimal solution in medium size and large/very large-sized problems respectively.

Keywords: Multiple traveling salesmen problem; Multi depot; Non-fixed destination; Genetic algorithms; Simulated annealing; Tabu search

1. Introduction

A generalization of the well-known Traveling Salesman Problem is the standard multiple Traveling Salesman Problem (mTSP). The problem can be defined simply as the determination of a set of tours for *m* salesmen who start from and return to a single home city (depot). The mTSP finds applications in printing press scheduling [1], crew scheduling [2], mission planning for autonomous mobile robots [3], hot rolling scheduling [4] and interview scheduling [5].

The multi-depot mTSP (MmTSP) is a generalization of the single depot mTSP, such that there is more than one depot (*d* depot) and a number of salesmen at each depot (m_k). If the problem is to determine a total of *m* tours such that the *m* salesmen must return to their original depots, this is referred to as the fixed destination MmTSP. On the other hand, if the salesmen do not have to return to their original depots but the number of salesmen at each depot should remain the same at the end as it was in the beginning, we have the non-fixed destination MmTSP [6].

In Fig. 1 (a) we illustrate the mTSP with 5 cities and 2 salesmen. Also in Fig. 1 (b) we illustrate the MmTSP with 10 cities, 4 salesmen, 2 depots and 2 salesmen in each depot.

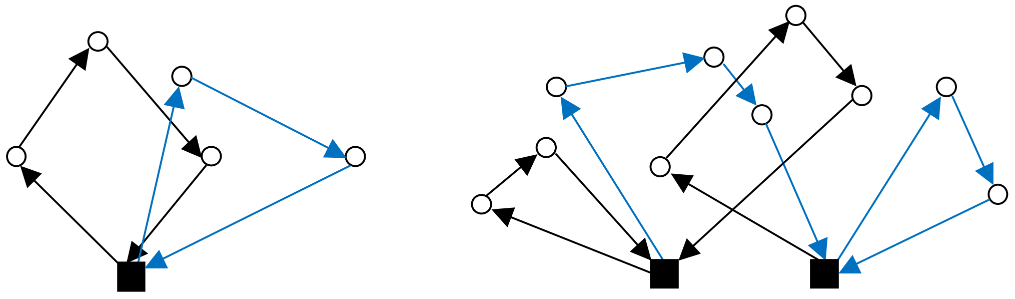


Fig.1: (a): mTSP (*n = 5, m = 2*) and (b): MmTSP (*n = 10, m = 4, d = 2,* $m_1 = 2$*,* $m_2 = 2$)

This problem can be transferred to a well-known mTSP by considering one city as single depot. MmTSP also can be transferred to a vehicle routing problem (VRP) by considering each salesman as a vehicle with a certain capacity [7]. Bektas [7] has addressed many applications of mTSP. Despite the many applications of this problem, it has not been duly attempted in the literature.

In this research two aspects are investigated. First, the non-fixed MmTSP problem is solved. Second, the effectiveness and efficiency of three well-known meta-heuristic methods in solving MmTSP are compared. These meta-heuristics are genetic algorithms, simulated annealing and tabu search.

2. Literature review

A number of different methods have been proposed for obtaining either optimal or near optimal solutions for the TSP. The methods used to solve the TSP range from classic methodologies based on linear programming [8] and branch-and-bound [9] to artificial intelligence methods such as neural networks [10], tabu search [11], and GAs [12]. For a good overview of the TSP and various solution methodologies for this problem see Lawler et al. [13]. Given the combinatorial complexity of TSPs of realistic size, a solution methodology that efficiently solves TSPs to global optimality remains elusive. While advances have been made in solving the TSP, those advances have come at the expense of increasingly complex computer code [14]. Most of the work on solving mTSPs using GAs have focused on vehicle scheduling problem (VSP) [15]. The VSP consists of scheduling a fleet of *m* vehicles to visit *n* cities where each city has to be visited by one and only one vehicle. The VSP typically includes constraints on the number of cities each vehicle can visit due to the capacity of each vehicle and the size of the load to be picked up at each city. In some cases, the cities must be visited within specific time windows. These issues lead to a number of different possible configurations for the VSP: those with or without time windows, those with heterogeneous or homogeneous vehicle capacities, and those with travel distance and/or fleet size restrictions. A variety of objectives can also be considered, including minimum or maximum total distance, minimum number of vehicles required, and minimum number of time window violations.

3. Non-fixed destination MmTSP

3.1. Problem definition and notation

Consider a complete directed graph $G = (V, A)$, where V is the set of n nodes (vertices), A is the set of arcs and $C = (c_{ij})$ is the cost (distance) matrix associated with each arc $(i, j) \in A$. The cost matrix C can be symmetric, asymmetric or Euclidean. Now let the node set be partitioned such that $V = V' \cup D$, where the first d nodes of V are depot set D. There are m_i salesmen located at depot i initially and the total number of salesmen is m. Also let $V' = \{d+1, d+2, \ldots, n\}$ be the set of customer nodes.

The MmTSP consists of finding tours for all the salesmen such that all customers are visited exactly once, the number of customers visited by a salesman lies between a interval and predetermined salesmen (m_k) depart from each depot and the total cost of all the tours is minimized.

Let us define x_{ij} as a binary variable equal to 1 if arc (i, j) is in the optimal solution and 0 otherwise. For any traveler, u_i is the number of nodes visited on the traveler's path from the origin up to node i (i.e., the visit number of the ith node). L is the maximum number of nodes a salesman may visit; thus, $1 \le u_i \le L$ for all $i \ge 2$. In addition, let K be the minimum number of nodes a salesman must visit, i.e., if $x_{i1} = 1$, then $K \le u_i \le L$ must be satisfied.

3.2. Formulation

The formulation for the non-fixed destination MmTSP proposed by Kara and Bektas [6] is adopted here with few changes:

Minimize $$\sum_{(i,j)\in A} c_{ij} x_{ij}$$

s.t.
$$\sum_{j\in V'} x_{ij} = m_i, \quad i \in D, \qquad (1)$$

$$\sum_{i\in V'} x_{ij} = m_j, \quad j \in D, \qquad (2)$$

$$\sum_{i\in V} x_{ij} = 1, \quad j \in V', \qquad (3)$$

$$\sum_{j\in V} x_{ij} = 1, \quad i \in V', \qquad (4)$$

$$u_i + (L-2)\sum_{k\in D} x_{ki} - \sum_{k\in D} x_{ik} \le L - 1, \quad i \in V' \qquad (5)$$

$$u_i + \sum_{k\in D} x_{ki} + (2-K)\sum_{k\in D} x_{ik} \ge 2, \quad i \in V' \qquad (6)$$

$$x_{ki} + x_{ik} \le 1, \quad k \in D, \quad i \in V', \qquad (7)$$

$$u_i - u_j + Lx_{ij} + (L-2)x_{ji} \le L-1, \quad i \ne j, \quad i,j \in V', \tag{8}$$

$$x_{ij} \in \{0,1\}, \quad i,j \in V. \tag{9}$$

In this formulation, for each $i \in D$, m_i outward and m_i inward arcs are guaranteed by (1) and (2). Eqs. (3) and (4) are the degree constraints for the customer nodes. Constraints (5) and (6) impose bounds on the number of nodes a salesman visits together with initializing the value of the u_i's as 1 if i is the first node visited on the tour. Constraint (7) prohibits a salesman from serving only a single customer. Finally, constraint (8) is sub-tour elimination constraints (SECs) in that they break all sub-tours between customer nodes.

Observe that there are $O(n^2)$ binary variables and $O(n^2)$ constraints in this formulation. It is easily seen that the MmTSP model reduces to that of the mTSP when D contains only one node, i.e., when D is a singleton. Consider that in this formulation one arbitrary tour in one solution must include at least one city (without depots), but in [6] each tour must include two cities (without depots).

4. Genetic algorithms

Genetic algorithms (GAs) are a relatively new optimization technique that can be applied to TSPs. The basic ideas behind GAs evolved in the mind of John Holland at the University of Michigan in the early 1970s [16]. GAs were not originally intended for highly constrained optimization problems but were soon adapted to order-based problems like the TSP [12], [17] and [18]. The development of effective GA operators for TSPs led to a great deal of interest and research to improve the performance of GAs for this type of problem [19], [20] and [21]. Several summaries of solving TSPs with GAs have been published that provide comprehensive reviews of the operators and associated issues [22], [23] and [24].

Concisely GAs work by generating a population of numeric vectors (called chromosomes), each representing a possible solution to a problem. The individual components (numeric values) within a chromosome are called genes. New chromosomes are created by crossover (the probabilistic exchange of values between vectors) or mutation (the random replacement of values in a vector). Mutation provides randomness within the chromosomes to increase coverage of the search space and help prevent premature convergence on a local optimum. Chromosomes are then evaluated according to a fitness (or objective) function, with the fittest surviving and the less fit being eliminated. The result is a gene pool that evolves over time to produce better and better solutions to a problem [25]. The search process in GAs typically continues until a pre-specified fitness value is reached, a set amount of computing time passes or until no significant improvement occurs in the population for a given number of iterations. The key to find a good solution using a GA lies in developing a good chromosome representation of solutions to the problem. A good GA chromosome design should reduce or eliminate redundant chromosomes from the population. Redundancy refers to a solution being able to be represented by a chromosome in more than one way and appearing in the population multiple times. Multiple representations of the same solution increase the search space and slow the search process.

Solving the TSP using GAs has generated a great deal of research on how best to perform the action of ''evolving'' an optimal (or good) solution to the problem. A number of different crossover methods have been proposed in the literature to solve the TSP using a GA. Some of the most commonly used operators include: Order Crossover [26], [27], [28] and [29], Partially Mapped Crossover [26], [28], [12] and [29], Cycle Crossover [28] and [29] and Asexual Crossover [14], [23] and [30]. Other proposed TSP operators include utilizing a matrix chromosome representation [19] and [20], hybrid operators [31], simple crossover [32], ordered crossover #2 [33] and moon crossover [34].

4.1. *Chromosome representation*

A well-designed chromosome should reduce or eliminate redundancy, accurately represent a solution to the problem, and allow the GA operators to work effectively to generate better solutions as the iterative evolutionary process continues.

Fig. 2 illustrates a method for representing solutions to a non-fixed MmTSP (where $n = 14, m = 3, d = 2, m_1 = 2, m_2 = 1$). This technique involves using a single chromosome of length $n+2m$ and is similar to the ‘‘one chromosome” technique [35]. In this technique, the n cities are represented by a permutation of the integers from 1 to n. This permutation is partitioned into m sub-tours by the insertion of $2m$ negative integers from -1 to $–k$ that kth depot must appear m_k times in starting depot and m_k times in ending depot. This attitude guarantees the condition that the numbers of salesmen start from and finish to depot k must be equal to m_k. This chromosome represents change from one salesperson to the next by a negative number as starting and ending depot of each sub-tour. Any permutation of these $n+2m$ integers represents a possible solution to the problem.

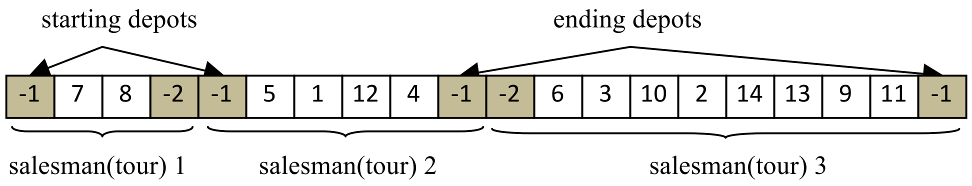


Fig.2: Chromosome representation for a 14 city MmTSP with four salesmen and two depots.

In the solution represented in Fig. 2, the first salesman starts from depot 1 and visits cities *7* and *8* (in that order) and then returns to depot 2. The second salesman starts from depot 1 and visits cities *5, 1, 12* and *4* (in that order) and then returns to depot 2 and so on for salesman 3.

4.2. *Initial population*

Chromosomes in initial population are generated randomly with this condition that there are no similar chromosomes in the initial population pool. Each chromosome is produced as follow: First a permutation of the n cities is randomly generated, then $2m$ negative integers in $\{-1,\ldots,-i,\ldots,-k\}$ will insert randomly in first and last position and between the cities of the permutation under this condition that each tour must have at least one city between two starting and ending depots. In addition each $-i$ must appear m_i times as starting depot and m_i times as ending depot. These aspects guarantee that each chromosome will be a feasible solution in initial population pool.

4.3. *Crossover*

In this chromosome we use one type of crossover that commonly used in previous researches, named modified 2-point crossover that are proposed from well-known 2-point crossover.

In this crossover first we select *2* arbitrary points (r_1 and r_2) between *1* and $n+2m$ of two parents (P1 and P2) that must show a position between two positive number, then copy cities before first point and after second point from P1 to offspring1 (O1), after that fill cities between 2 points from P2 to O1 considering that they aren't repeating, if there is negative number as starting/finishing depot we should go on to receive next valid (with considering m_is) negative number as starting/finishing depot; because two negative numbers should settle besides each other, then go back after first negative point and check other numbers respectively. To fill cities in O2 we do this conversely, copy cities between 2 points from P1 to O2, after that fill cities from beginning of O2 with cities of P2 until first point (r_1) and fill cities from the end of O2 with cities in P2 until the second point (r_2), considering that they aren't repeating before. Accordingly in O3 first copy cities before first point and after second point from P2 then fill cities between two points from P1 and in O4 first copy cities between two points from P2 then fill cities before first point and after second point from P1.

For example in Fig.3 parent 1 (P1) and Parent 2 (P2) are two distinct solutions for a problem with $n=12, m=3, d=2, m_1=2, m_2=1$. First we select $r_1 = 6$ (between *5* and *1*) and $r_2 = 13$ (between *3* and *10*).

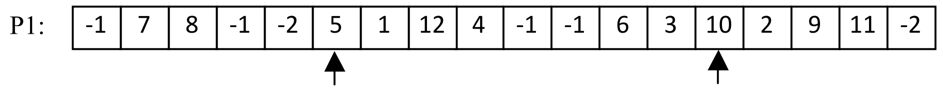
In O1, first we copy the numbers from first node to r_1 and from r_2 to the last node in P1 in same position in O1, i.e. "*-1, 7, 8, -1, -2* and *5"* from first to r_1 and "*10, 2, 9, 11* and *-2"* from r_2 to the end. For the numbers between r_1 and r_2, we start from first node of P2, if the numbers were not in the O1, copy them to the P1 respectively until O1 is completed, i.e. copy "*6, -1, -1, 12, 3, 4* and *1"* between r_1 and r_2.

In O2, first we must copy the numbers between r_1 and r_2 from P1 to the same position in O2, i.e. copy "*1, 12, 4, -1, -1, 6* and *3*" from P1 to O2. Then start from beginning of P2 and copy the non-existing number to O2 to fill the all blanks before r_1, i.e. copy "*-2, 9, 5, -1, -1* and *11*" from P2 to the O2's nodes before r_1. After that start from the end of P2 and copy the non-existing numbers from the end of O2 respectively until receive to r_2 to complete the O2.

In O3 we do like O1 with consider P2 as P1, i.e. copy "*-2, 9, 5, 6, -1* and *-1*" before r_1 and "*4, 10, 1, 7* and *-2*" after r_2 from P2 to the same position in O3, then copy the non-existing number "*8, 12, -2, -1, 3, 2* and *11*" from P1 to the blanks in O3 to complete it.

In O4 we do like O2 with consider P2 as P1, i.e. copy "*11, 12, 8, 2, 3, -2* and *-1*" from P2 to same position in O4, then start from beginning of P1 and copy the non-existing number until r_1, after that start from the end of P2 and copy the non-existing number from the end of O4 until r_2 to complete O4.

Consider that in O4 first tour has only 1 city, so this offspring is infeasible and can't be accepted, because each tour must include at least 2 cities.



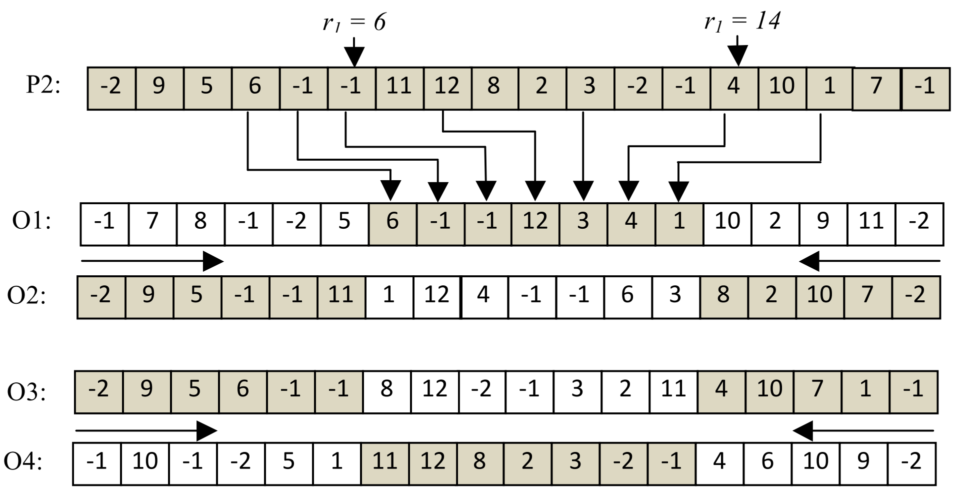

$r_1 = 6$
$r_1 = 14$
P2: -2 9 5 6 -1 -1 11 12 8 2 3 -2 -1 4 10 1 7 -1
O1: -1 7 8 -1 -2 5 6 -1 -1 12 3 4 1 10 2 9 11 -2
O2: -2 9 5 -1 -1 11 1 12 4 -1 -1 6 3 8 2 10 7 -2
O3: -2 9 5 6 -1 -1 8 12 -2 -1 3 2 11 4 10 7 1 -1
O4: -1 10 -1 -2 5 1 11 12 8 2 3 -2 -1 4 6 10 9 -2

Fig.3: modified 2-point crossover

4.4. Mutation

The mutation mechanism is based on three common move operators called: 1-bit, 2-opt and intra2-opt. These are described in the following.

1. 1-bit operator (see Fig. 4). In this operator one arbitrary city is removed from one tour and is added to another tour in the parent.

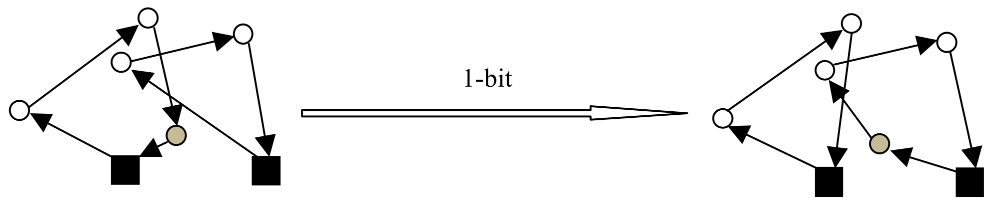


Fig.4: 1-bit operator

2. 2-opt operator (see Fig. 5). In this operator two arbitrary cities of two different tours of a parent are exchanged.

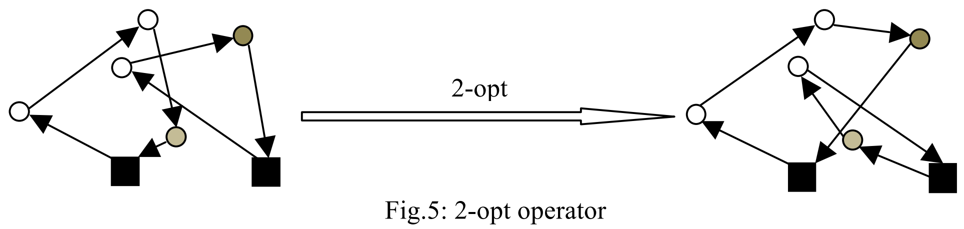


Fig.5: 2-opt operator

2. *intra2-opt operator* (see Fig. 6). In this operator two arbitrary cities of one tour of a parent are exchanged.

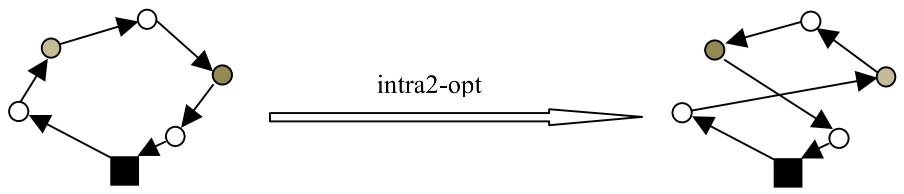


Fig.6: intra2-opt operator

4.5. Choosing parents' strategies

Three strategies for choosing parents are often used in GAs. These strategies are elitism, roulette wheel [26] and binary tournament. In elitism strategy the chromosomes with a fitness better than others in the preceding generation are chosen as parents in the current generation. In roulette wheel strategy all chromosomes have a chance to be chosen as parent, considering their fitness functions (i.e. the chromosome with lower fitness has more chance to be chosen). In binary tournament, two chromosomes are chosen from the population at random. The one with the better fitness value is chosen as a parent. The process is repeated to obtain a second parent with this condition that there is no similar chromosomes in the parents' population pool.

4.6. The GA algorithm

The following parameters are used in the proposed genetic algorithm:

G_{max}: the maximum number of generations
σ_{min}: minimum diversity of current generation's fitness function
pop-size: the number of population size of each generation
c_r: crossover rate
m_r: mutation rate
r_r: reproduction rate

The steps of the proposed GA are as follows:

```
Set $g = 0$, $t = 0$, $c = c_r \times$ pop-size, $m = m_r \times$ pop-size and $r = r_r \times$ pop-size;
Generate initial population randomly;
Repeat
    Select the r best chromosomes of the current population and copy them to the next generation without
        any changes;
    Select two chromosomes as parents;
    Repeat
        Perform crossover operator on them and produce 4 offspring;
            If any offspring has better fitness value than parent(s), copy the child to the next generation;
        Select one mutation operator randomly and create an offspring from the new child produced in
            previous step;
            If the new offspring has a better fitness value than its parent, copy it to the next generation;
    Until new population has been created;
Until $G_{max}$ generation has been passed or diversity of the current generation's fitness is smaller than $\sigma_{min}$;
```

5. Tabu Search

Tabu Search is a memory-based search strategy, originally proposed by Glover (see [37]), to guide the local search method to continue its search beyond a local optimum. One way of achieving this is to keep track of recent moves or solutions made in the past. A tabu list records recently made moves or visited solutions. Whenever the algorithm attempts to make a move listed in the tabu list, the move is banned. By this, the algorithm forces other solutions to be explored. However, this feature is not strict; it can be overridden when some aspiration criteria are satisfied. A popular aspiration criterion is that the fitness function value be the best ever seen. If this is the case, it is obvious that this solution has never been encountered before.

The basic idea of TS is the following. Begin with an initial solution. At each iteration, a neighborhood of the current solution is created through different classes of moves. Then, the best admissible solution in the neighborhood is selected as the new current solution, and the procedure is repeated until a stopping criterion is satisfied. A move is always admissible if it is not tabu. If it is tabu, then it may still be admissible if the move produces a solution strictly better than the best solution obtained so far. In both cases, the accepted move is recorded, and the tabu restrictions are updated.

Over the last 10 years, several meta-heuristics have been proposed for the VRP, such as simulated annealing, tabu search, genetic algorithms, ant algorithms, and neural networks. A comprehensive survey is provided by Gendreau et.al [38] and Cordeau et al. [39]. According to their analyses, in general, TS emerges as the most effective approach for the VRP, both in solution quality and computing time. Furthermore, there has been a tendency toward the development of fast, simple (with fewer parameters), and more robust algorithms.

In this paper, tabu search is considered as a meta-heuristic method for solving the problem; because VRP can be transferred to the MmTSP by ignoring its capacity constraints.

5.1. Initial solution

An initial solution is required for any TS algorithm to start the local search process. With respect to the TS meta-heuristic for the VRP, Van Breedam [40] states that the performance of TS heuristic is highly dependent on the quality of the initial solutions. Brandao [41] also shows that the initial solution can give an important contribution to enhance the final solution, and uses two methods for obtaining the initial solutions: a nearest neighbor heuristic and a pseudo lower bound. Based on the nearest neighbor method, a heuristic is proposed in the following to generate an initial solution.

In this paper we use a heuristic algorithm (NNS) based on nearest neighborhood search to find initial solution. For each tour first a non-allocated starting depot is randomly selected then a city with minimum cost is allocated to that until one city is allocated to all starting depots. Then the non-allocated nearest neighbor city will search continually until all cities are allocated to the tours. At the end of each tour, nearest finishing depot is allocated to it. The steps of the NNS are as follows.

1. Let $D = \{d_1,...,d_k\}$ be the set of depots and $V' = \{1,...,n\}$ the set of cities, $H = \{h_1, h_2,..., h_m\}$ be the last city in each tour, $M = \{m_1,...,m_i,...,m_k\}$ be the set of tour's number that should depart from depot *i* and $M' = \{m'_1,...,m'_i,...,m'_k\}$ be the set of tour's number that should finish in depot *i*.
2. Select depot *i* at random where $i \in D$ and $M_{d_i} > 0$. Set $M_{d_i} = M_{d_i} - 1$.
3. Allocate nearest city *j* to depot *i* and let $V' = V' - \{j\}$ and $h_i = j$.
4. Repeat steps 2 and 3 until $m_{d_i} > 0$, $\forall i \in D$, i.e. each depot has been connected to one city (each tour has one city).
5. Find the nearest city h_i to city *j* where $j \in V'$. Set $h_i = j$ and $V' = V' - \{j\}$.
6. Repeat step 5 until $V' = \phi$, i.e. all cities are allocated to one tour.
7. Allocate city h_i, the last city in each tour, to the nearest finishing depot that $m'_{d_i} > 0$, $i \in D$ and let $m'_{d_i} = m'_{d_i} - 1$.
8. Repeat step 7 until all last cities are allocated to one finishing depot.

5.2. Neighborhood Structure

Various neighborhood structures have been presented for the application of TS to the TSP and VRP by several authors. Pureza and Franc'a [42] defined the neighbors of a solution by moving a vertex to a different tour or by swapping vertices between two tours while preserving feasibility. In Osman [43] neighborhood is defined by means of a l-interchange generation mechanism, with $\lambda = 2$. This includes a combination of 2-opt moves, vertex reassignments to different tours, and vertex interchanges between two tours. The neighborhood structure proposed by Gendreau et al [44] is defined by removing a vertex from its current tour and inserting it into another tour containing one of its *p* nearest neighbors using GENI, a Generalized Insertion procedure developed by them for the TSP. Taillard et al [45] introduced another exchange heuristic, called as CROSS exchange, to generalize two edge exchange heuristics for problems with time windows. In TS heuristic for the vehicle routing problem with backhauls and time windows, Duhamel et al [46] presented a stochastic mechanism in which the current neighborhood is randomly selected among 2-opt, Or-opt, and swap. Brandao [41] defines the neighborhood by two types of trial moves, insert and swap. An insert move consists of taking a customer from one tour and inserting it into another tour, and a swap move consists of exchanging two customers belonging to two different tours.

In our implementation, different classes of neighborhood moves are applied to the current solution. These moves are based on the 2-interchange generation mechanism but with a combination of vertex reassignment, vertex swap, 2-opt and 'tails swap', between two tours and not within the same tour. For same tours we use intra exchanging phase as a local search to improve the current solution. These move operators are discussed below.

1. Relocate operator (see Fig. 7). For city $i \in tour_k$ and city $j \in tour_l$, place *i* after *j* in $tour_l$. The new tours are $tour'_k = (-d_1,...,i-1, i+1,...,-d_2)$ and $tour'_l = (-d_3,..., j, i, j+1,...,-d_4)$.

2 . *Exchange operator* (see Fig. 8). Cities $i \in tour_k$ and $j \in tour_l$ are exchanged between tours $tour_k$ and $tour_l$. The new tours are $tour_k^{'} = (-d_1,\ldots,i-1,j,i+1,\ldots,-d_2)$ and $tour_l^{'} = (-d_3,\ldots,j-1,i,j+1,\ldots,-d_4)$.

3. *tail-swap operator* (see Fig. 9). For city $i \in tour_k$ and city $j \in tour_l$, let the new tours be $tour_k^{'} = (-d_1,\ldots,i,j+1,j+2,\ldots,-d_2)$ and $tour_l^{'} = (-d_3,\ldots,i,i+1,i+2,\ldots,-d_4)$.

4. *depot-exchange operator* (see Fig. 10). This move operator exchanges two arbitrary starting/ending depots.

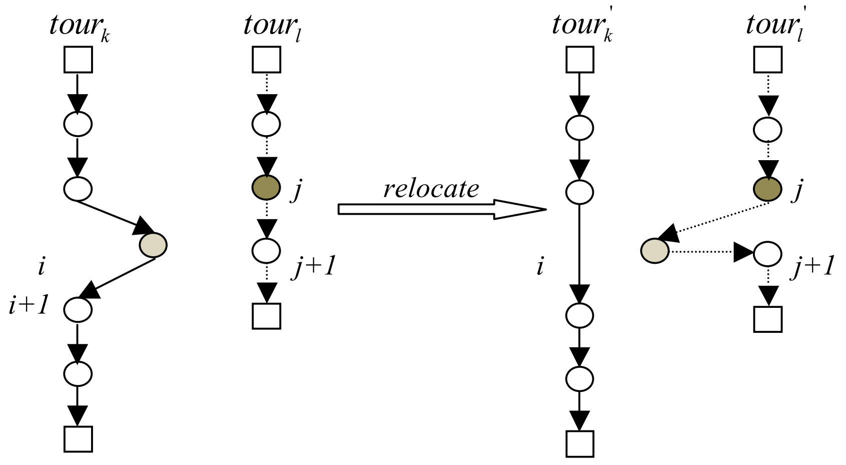


Fig.7: relocate operator

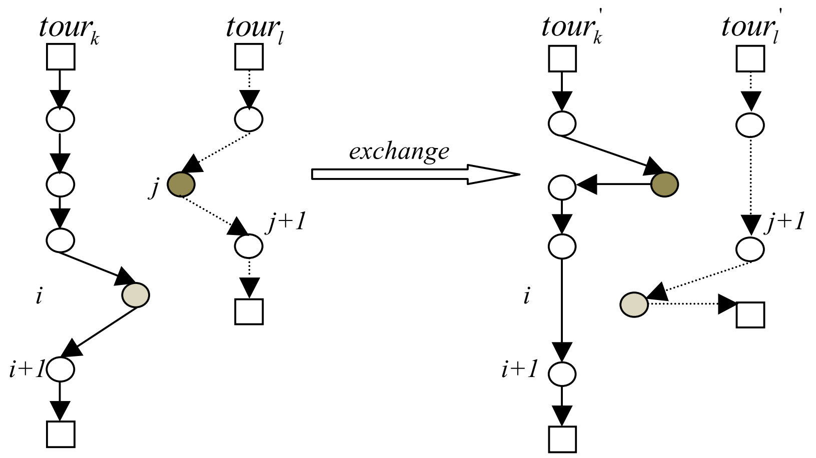


Fig.8: exchange operator

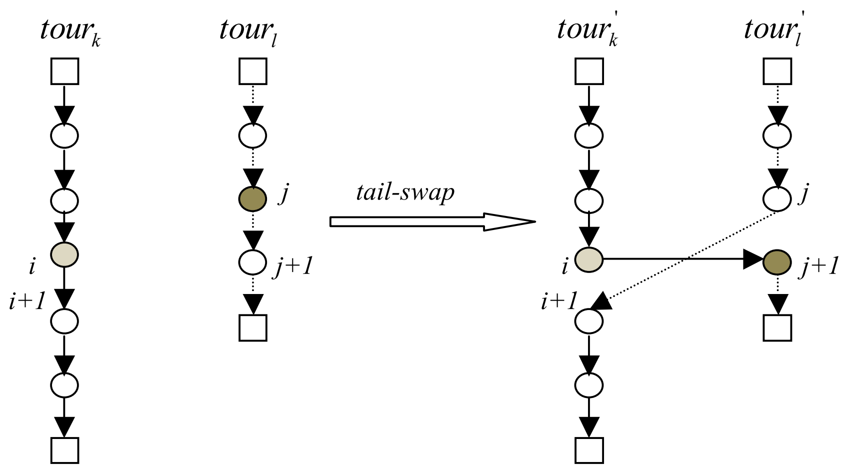


Fig.9: tail-swap operator

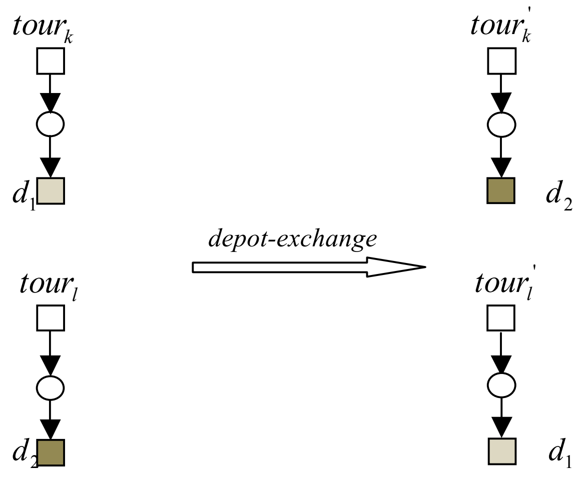


Fig.10: depot-exchange operator

5.3. Tabu List

The tabu list contains the “fitness function” attributes of solutions during the last 10–15 (selected randomly) iterations. A set of (min-fitness… max-fitness) integers can be constructed for the record of tabu status; min-fitness is the fitness of best solution or near it that we wish to obtain and max-fitness is the fitness of initial solution obtained by NNS or random generation. At each iteration, the tabu status of the last fitness function performed is added to the list, while the others are decreased by one until zero.

5.4. Stopping criterion

The search is terminated if either a specified number of iterations have elapsed in total or since the last best solution was found.

5.5. Intra exchanging phase

Operations in this phase are similar to operations in exchange operator. Instead of exchanging two arbitrary nodes, it compares all pairs of nodes in that tour and then chooses the best exchange between two nodes in all individual tours.

5.6. The TS algorithm

Pareto principle is used in the TS algorithm proposed in this paper for acceptance of neighbor solution. A solution in the neighborhood of the current one is accepted if it is not dominated by all non-dominated solutions found so far in the search. This local search algorithm is called Pareto Local Search (PLS). The local search algorithm then works, from a general perspective, as follows.

1. It starts from a randomly generated tour, which is added to the archive.
2. It picks a solution x randomly from the archive and explores its neighborhood.
3. Once a non-dominated solution x' in the neighborhood of x is found, x' is added to the archive.
4. After the full examination of the neighborhood of x, it is flagged as *visited* and we continue at step 2.

The local search procedure terminates if all the neighbors of all solutions in the archive were explored, i.e. every solution in the archive is flagged as visited. In this case, the archive is a Pareto local optimum set.

A brief description of the TS algorithm used in this paper is as follow. in the first step, a set of good solutions are generated as Pareto list. The worst solution in the list is always selected as current solution. Then a neighbor solution is generated from current solution with one neighbor operator. This new solution is accepted if it is better than the current one, so it is replaced with current solution in Pareto list. If there is no change in Pareto list after v consecutive neighbor generation, the current solution is eliminated from the list and the worst solution in the list is considered as the current solution. The algorithm will be continued until all solutions in the list are eliminated.

The following variables are used in the description of TS heuristic.

i, index of current iteration;

m, maximum number of iterations;

c, number of consecutive iterations without any change in Pareto list;

mc, maximum number of consecutive iterations without any change in Pareto list;

l, number of solutions in the Pareto list;

p, length of the Pareto list.

v, number of consecutive iterations without any change in current solution that must performing intra exchange phase.

A pseudo code of the proposed TS algorithm is given below:

```
Set i and c equal to 0;
Generate one initial solution, $X_0$, with NNS;
Generate $p - 1$ solution from $X_0$ with relocate operator and set these p solutions as Pareto list;
Set the best solution in the list as $X_{best}$ and the worst solution as the current solution;
Repeat
    Select one of the four types of neighborhood move operator randomly;
    Generate one solution by the selected move operator from current solution;
    Set the new solution as current solution if it is not tabu and better than the current solution, update
        tabu list and set i = i + 1 and c = 0. Otherwise set c = c + 1;
    If (c mod v = 0) then perform intra exchanging phase to current solution;
    If (c > mc) then eliminate current solution from the list and set the worst solution in the list as current
        solution;
Until (i <= m) and tabu list is not empty;
```

The distinctive feature of this TS heuristic is the use of a simple but powerful neighborhood structure. During the search, the current neighborhood is randomly selected among four types of neighborhood move, and the tabu length is randomly selected between [20...35]. This mechanism introduces a stochastic type of diversification. Furthermore, by applying a different transformation at each iteration, a larger neighborhood is explored over a few iterations, without a computational burden associated with an extensive search in a unified neighborhood. The proposed tabu search heuristic improves the current solution by performing intra exchanging phase in *v* consecutive iterations without any change in current solution. This is considered as an intensification mechanism in this tabu search heuristic.

6. Simulated Annealing

Simulated annealing (SA) is a stochastic relaxation technique that has its origin in statistical mechanics [36]. The SA methodology draws its analogy from the annealing process of solids. In the annealing process, a solid is heated to a high temperature and gradually is cooled to allow it to crystallize. As the heating process allows the atoms to move randomly, if the cooling is done too rapidly, it gives the atoms enough time to align themselves in order to reach a minimum energy state. This analogy can be used in combinatorial optimization in which the state of solid corresponding to the feasible solution, the energy at each state corresponding to the improvement in the objective function and the minimum energy state will be the optimal solution.

SA uses a stochastic approach to direct the search process. It allows the search process to proceed to a neighboring state even if the move causes the value of the objective function to become worse. SA guides the original local search method in the following way. If a move to a neighbor X_0 in the neighborhood N(X) decreases the objective function value, or leaves it unchanged, then the move is always accepted. More precisely, the solution X_0 is accepted as the new current solution if $\Delta \leq 0$, where $\Delta = C(X^{'}) - C(X)$ and $C(X)$ is the value of objective function. Moves, which increase the objective function value, are accepted with a probability of $e^{(-\Delta/T)}$ to allow the search to escape from a local optimum; where T is a parameter called temperature. The value of T varies from a relatively large value to a small value close to zero. These values are controlled by a cooling schedule that specifies the initial and incremental temperature values at each stage of the algorithm.

The SA has two inside and outside loops. The inside loop controls the achievement to an equilibrium state in the current temperature and the outside loop controls the rate of temperature decrease.

6.1. Initial solution

A heuristic based on the nearest neighbor search (NNS) is used for generating initial solution as described in section 5.1.

6.2. Neighborhood generation

The operators in relocate, exchange, tail-swap and depot-exchange methods as discussed in section 5.2, are used for the move in feasible space to obtain the neighbor solutions. In addition, one more operator called intra-exchange is used for searching in one tour of the current solution. The performance of this operator over one tour is depicted below.

intra-exchange operator (see Fig. 11). Compared to the four move operators listed above, this operator happens in only one tour. For city $i \in tour_k$ and city $j \in tour_k$, let the new tour be $tour_k^{'} = (-d_1,...,i-1,j,i+1,...,j-1,i,j+1,...,-d_2)$.

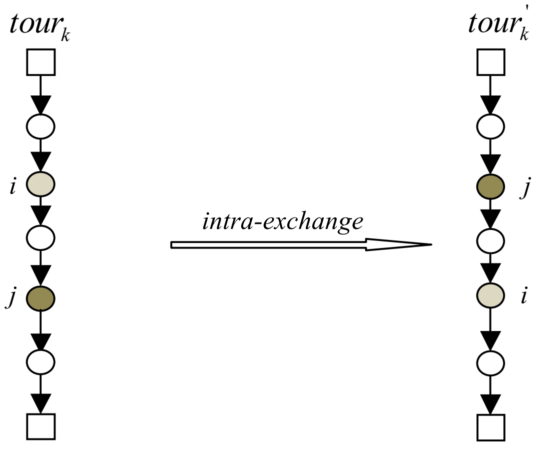


Fig.11: intra-exchange operator

6.3. The SA Algorithm

The SA parameters are introduced as follows.

E (Epoch length), number of accepted solutions in each temperature to achieve an equilibrium;
M, maximum number of consecutive temperature trails;
T_0, initial temperature;
α, rate of the current temperature decrease (cooling schedule);
X, a feasible solution;
C(X), the value of objective function for *X*;
n, counter for the number of accepted solutions in each temperature;
k, counter for the number of consecutive iteration in each temperature;

r, counter for the number of consecutive temperature trails, where T_r is equal to the temperature in iteration *r*.

The pseudo code of the proposed SA algorithm is as follow:

```
Initialize parameters $T_0, M, E, \alpha, T_f$
Generate initial solution by NNS algorithm $X^0$ and set $X^{best} = X^0$
Set r = 0 and n = 0
Do (outside loop)
  Set k = 0
  Do (inside loop)
    Generate neighboring solution $X^{n+1}$ by one randomly selected neighborhood generator: $X^n \longrightarrow X^{n+1}$
    if $F(X^{n+1}) \prec F(X^n)$ then
      $X^n = X^{n+1}$ : set k = k+1
    else
      Generate random $y \longrightarrow u(0,1)$
      if $y \prec e^{-\Delta F / T_k}$ then $X^n = X^{n+1}$ : set k = k+1
    End if
    Update $X^{best}$
  Loop until $k \le E$
  $T_{r+1} = \alpha . T_r$
Loop until $r \le M$
```

7. Experimental Results

In this section, we describe our experience with some computational experiments done using the genetic algorithms, simulated annealing and tabu search described in previous sections. In the two following subsections, the computational testing methodology and the computational results on the non-fixed destination MmTSP are presented, respectively.

7.1. Computational testing methodology

In order to evaluate the practical benefits of the genetic algorithms, simulated annealing and tabu search described in previous sections, computational tests were conducted to compare the performance of all three meta-heuristics on a set of problems created for the non-fixed destination MmTSP.

Most of methods presented in the literature are exact methods solving small problems. However, we generated test problems in medium ($n = 50$), large ($n = 100$) and very large ($n = 150$) sizes to examine the performance of the proposed heuristics.

Table 1 shows the size of the tested problems in terms of the number of cities (n), salesmen (m), depots (d) and number of salesmen in each depot (m_i) for each problem. For each level of n, all salesmen started and ended their individual tours in an arbitrary depot. In each problem, the number of salesmen in each depot should remain the same at the end as it was at the beginning. The fitness function minimizes the total distance traveled by all of the salesmen. For these runs, each salesman visits at least one city (other than the depot cities). This objective would represent a situation where there is a set of salesmen and with no constraints on the maximum number of cities to be visited by each salesman. Thus, any salesman can visit at most $n-2(m-1)$ cities (other than the depot cities).

Table 1
Computational test problems' conditions

ID	Number of cities(n)	Number of salesman(m)	Number of depots(d)	Number of salesman in each depots(m_k)		
				m_1	m_2	m_3
M_1	50	2	2	1	1	
M_2	50	3	2	2	1	
M_3	50	4	2	2	2	
L_1	100	2	2	1	1	
L_2	100	4	2	2	2	
L_3	100	5	2	3	2	
L_4	100	6	3	2	2	2
E_1	150	2	2	2	2	
E_2	150	4	2	2	2	
E_3	150	6	2	3	3	
E_4	150	8	3	3	3	2
EX	200	4	2	2	2	

In the following, we have considered two parameters, namely fitness function and CPU time to compare the accuracy and efficiency of the algorithms. We compare the best result obtained by these meta-heuristics and the optimal solution obtained by Lingo 8.0.

Each of twelve problems was run 30 times. The tests were run on a Celeron CPU 2.0 GHz 256 MB of RAM using computer programs written in Turbo Delphi.

7.2. Parameter tuning

To find the best value of the parameters used in these algorithms, we have done some experimentations for each size of test problems (i.e. for $n = 50$, $n = 100$ and $n = 150$). For each size, four salesmen and two salesmen in each depot are considered. The problems are run 30 times for different values of all parameters and the best result is selected for each parameter and is used in all problems with the same size (i.e. the results of $n = 50$, $m = 4$, $m_1 = 2$ and $m_2 = 2$ are used for all test problems with $n = 50$ and so on for $n = 100$ and $n = 150$).

7.2.1. Genetic algorithms

The parameters tuned in the proposed GA are as follow: population size of each generation (*pop-size*), operators rate (crossover rate (c_r), mutation rate (m_r) and reproduction rate (r_r)) and maximum number of generations (G_{max}).

To find the best value of *pop-size,* we use 11 different values for each size of problem. For example Fig.12 illustrates the values of *pop-size* in term of fitness function for a problem of size n=100. As shown in this figure, population size of 110 gives the best fitness value. Therefore, we set *pop-size*=110 for all problems with n = 100.

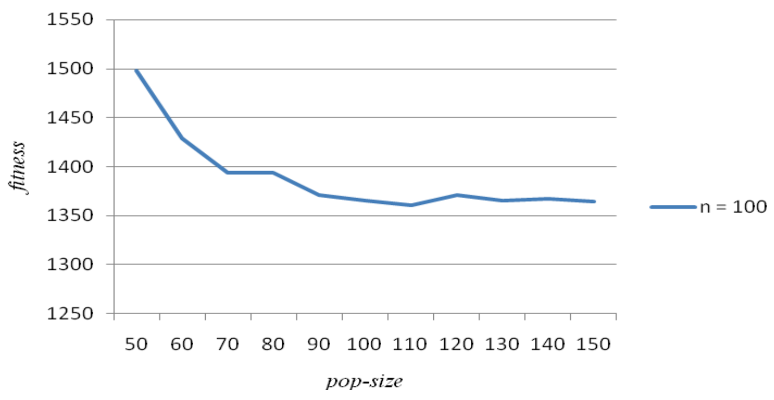


Fig.12: The *pop-size* tuning for a problem with *n =100* cities (*m = 4*, $m_1 = 2$ and $m_2 = 2$)

The values set for each parameter in each problem size are shown in Table 2.

Table 2
The values of different parameters tuned for GA

Problem size (*n*)	*pop-size*	Operator rates %			G_{max}
		c_r	m_r	r_r	
50	80	75	20	5	3000
100	110	65	30	5	7000
150	100	65	30	5	9000

7.2.2. Tabu search

The parameters tuned in the proposed TS are maximum number of consecutive iterations without any change in Pareto list (*mc*), length of the Pareto list (*p*) and tabu-tenure (*t*).

To find the best value of *mc*, we use 11 different values for each size of problem. For example, Fig.13 illustrates the values of *mc* in terms of fitness function for a problem with *n*=100 cities. As shown in this figure, a maximum number of consecutive iterations without any change in Pareto list of 20000 gives the best fitness value. Therefore, we set *mc*= 20000 for all problems with *n* = 100 cities.

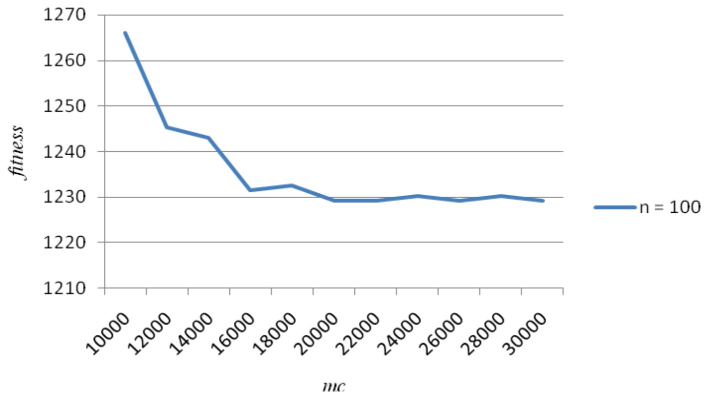


Fig.13: The *mc* tuning for *n =100* (*m = 4*, $m_1 = 2$ and $m_2 = 2$)

With a similar analysis, the values of other parameters are tuned as shown in Table 3.

Table 3
The results of parameter tuning for TS

Problem size (n)	mc	p	t
50	16000	10	25…35
100	20000	20	30…40
150	26000	20	35…55

7.2.3. Simulated annealing

The parameters to be tuned in the proposed SA are the initial temperature (T_0), the rate of current temperature decrease (α), the number of accepted solutions in each temperature to achieve the equilibrium state (E) and the maximum number of consecutive temperature trails (M).

To find the best value of T_0, five different values are tested for each size of problem. For example, Fig.14 illustrates the values of T_0 in terms of fitness function for a problem with n=100 cities. As shown in this figure, an initial temperature of 10 gives the best fitness value. Therefore, we set T_0 = 10 for all problems with n=100 cities.

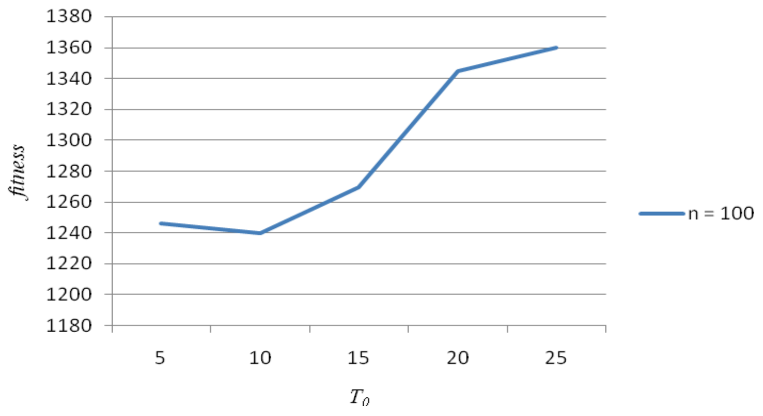


Fig.14: The T_0 tuning for $n = 100$ ($m = 4$, $m_1 = 2$ and $m_2 = 2$)

Similarly, the values of other parameters for SA are tuned as shown in Table 4.

Table 4
The results of parameter tuning for SA

Problem size (n)	T_0	α	E	M
50	10	0.9	70	45
100	10	0.85	100	30
150	5	0.9	200	20

7.3. Computational results

To the best knowledge of the authors, there were no standard test problems for the MmTSP on the accessible libraries. Therefore, we produced the two dimensional cost (distance) matrix $C_{(d+n)(d+n)}$ between all nodes (depots and cities) where c_{ij}s are random integral numbers generated uniformly in the span $10 \le c_{ij} \le 100$. The cost matrix for all problems with the same number of cities (n) and depots (d) is the same.

When minimizing the total distance of all the salesmen in the non-fixed MmTSP, as the number of salesmen in each depot increases, the total distance of all of the trips also tends to increase. This results from the fact that each salesman must start and return to the home city. So as the number of salesmen increases, the number of trips into and out of the home city increases, thereby increasing the total distance traveled. The results of each algorithm are described as follows.

7.3.1. Genetic Algorithms

If any new solution had a better fitness than its parents, then the new solution is inserted in the current new population. The algorithm stops after a maximum number of generations (G_{max}) or the diversity of chromosomes in the last generation be smaller than 10 (i.e. $\sigma < 10$). The parameter σ is calculated as follows in which f_i is the fitness of chromosome i in the current population and $\bar{f}$ is the mean fitness of all chromosomes in the current population.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{pop-size} (f_i - \bar{f})^2}{pop - size}}$$

The results obtained by the proposed genetic algorithm and the optimal solution of each problem obtained by the Lingo 8.0 Software are shown in Table 5. The last column underneath the GA presents the percentage of difference between the GA solution and the optimal solution. This criterion is calculated by

$$gap\% = \frac{GA(fitness) - Lingo(fitness)}{Lingo(fitness)} \times 100 \cdot$$

7.3.2. Tabu search

The proposed TS algorithm uses a random selection of four types of move operators at uniform interval [1…4] (see section 5.2): relocate, exchange, tail-swap and depot-exchange. All these operators do their actions in two different tours in the current solution. To improving the current solution, we use intra exchanging phase (see section 5.5), that performs in one tour, in $v = 40$ consecutive iterations without any change in the current solution.

The results obtained by the proposed TS algorithm is compared with the optimal solutions obtained by the Lingo 8.0 Software in Table 5. The last column underneath the TS presents the percentage of difference between the TS solution and the optimal solution. This criterion is calculated by

$$gap\% = \frac{TS(fitness) - Lingo(fitness)}{Lingo(fitness)} \times 100$$

7.3.3 Simulated annealing

A simulated annealing algorithm generally needs to be designed carefully as the choice of its parameters might affect the quality of solution and computational time. Our SA algorithm uses a random selection of five types of move operators (see section 6.2) at uniform interval [1… 5] where its fitness is controlled by the cooling schedule.

The SA parameters are set in advance according to Table 4. The results obtained by the proposed SA algorithm is compared with the optimal solutions obtained by the Lingo 8.0 Software in Table 5. The last column underneath the SA presents the percentage of difference between the SA solution and the optimal solution. This criterion is calculated by

$$gap\% = \frac{SA(fitness) - Lingo(fitness)}{Lingo(fitness)} \times 100$$

7.4 Comparison of heuristics

In this section, the performance of the proposed GA, TS and SA algorithms are compared in terms of the final solution fitness and computational time (Table 5).

Table 5
Computational Results

Problem ID	**Optimal results**		**GA results**			**TS results**			**SA results**		
	fitness	*CPU time(sec)*	*fitness*	*CPU time*	*gap%*	*fitness*	*CPU time*	*gap%*	*fitness*	*CPU time*	*gap%*
M_1	633	22	712	90	12.5	671	19	6	670	31	**5.8**
M_2	647	22	733	98	13.3	684	18	5.7	633	28	**2.4**
M_3	664	21	745	88	12.2	701	21	5.6	694	9	**4.5**
L_1	1125	454	1308	118	16.2	1182	81	**5**	1199	13	6.5
L_2	1152	191	1326	114	15.1	1208	89	**4.9**	1221	12	6
L_3	1166	193	1363	120	16.9	1228	95	**5.3**	1248	14	7
L_4	1175	900	1368	108	16.4	1246	99	**6**	1276	12	8.5
E_1	1583	770	1868	156	18	1683	212	**6.3**	1707	47	7.8
E_2	1606	811	1892	146	17.8	1702	201	**6**	1737	38	8.1
E_3	1630	1077	1944	154	19.3	1729	220	**6.1**	1765	40	8.2
E_4	1676	783	2037	160	21.5	1782	229	**6.3**	1819	44	8.5
EX	-	-	2783	246	-	**2248**	441	-	2355	127	-

Fig.15 shows the fitness values of the test problems obtained by GA, TS, SA and optimal solution. For the *EX* problem, Lingo can not give optimal solution after 48 hour, so the results for GA, TS and SA have been reported.

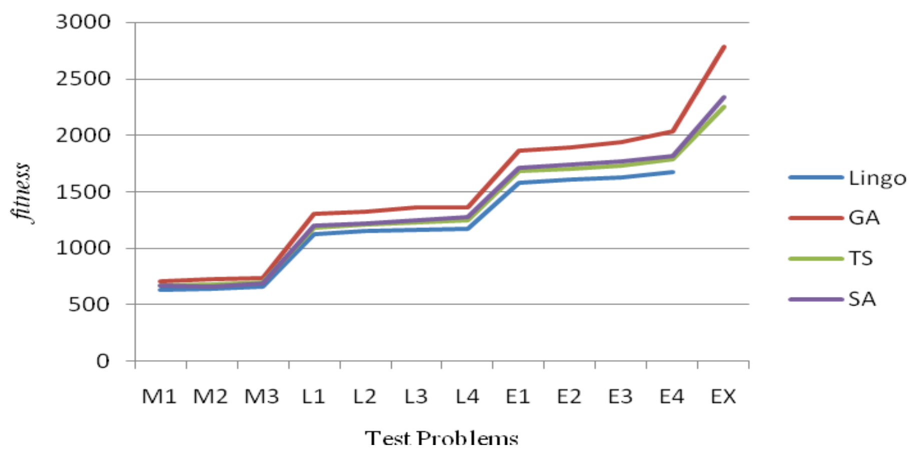


Fig. 15: Comparison between the fitness of GA, TS, SA and optimal solution in test problems

As seen in Table 5 and Fig. 15, in medium-sized problems with a $n = 50$ cities, fitness value of the SA is fairly better than the TS. In large- and very large-sized problems, however, the TS algorithm has obtained better solutions. In all problems, the fitness value of GA is obviously worse than others.

Fig. 16 illustrates the computational time of GA, TS, SA and Lingo 8.0.

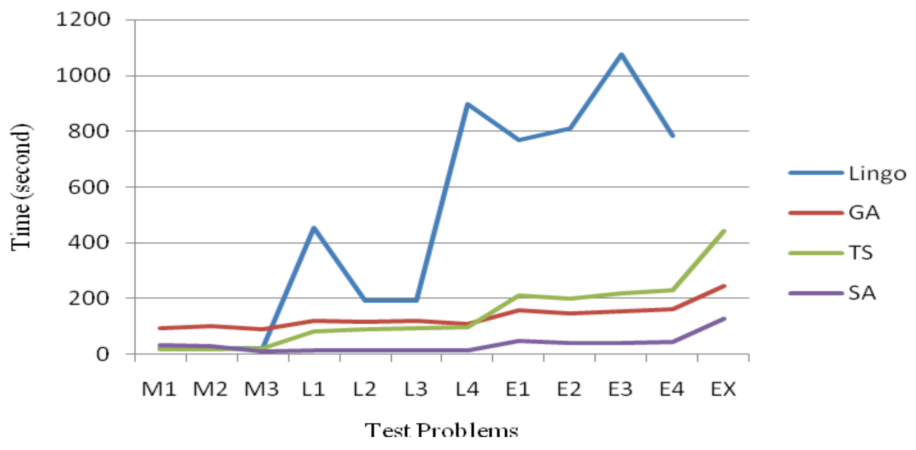


Fig. 16: Comparison between the computational time of GA, TS, SA and Lingo 8.0

Fig. 16 shows that the SA has the least computational time compared to GA, TS and Lingo 8.0. The GA is the next and the TS has the greatest computational time among the meta-heuristic algorithms. Only in medium-sized problems, Lingo 8.0 can compete with meta-heuristic algorithms and in large- and very large-sized problems the difference between Lingo 8.0 and others is very remarkable.

In summary, even for large-sized problems with n=200 cities, the computational time of TS is 441 seconds, which is a promising time. Therefore, TS is the best algorithm for solving non-fixed destination MmTSP.

8. Conclusion

In this paper, we present three meta-heuristic methods, namely genetic algorithm, tabu search and simulated annealing, for solving the non-fixed destination multi-depot multiple traveling salesmen problem. Compared to Lingo 8.0, the experimental results demonstrate the effectiveness and efficiency of the SA algorithm which obtains answers very close to the optimal solution in medium-sized problems. For large- and very large-sized problems, the TS algorithm obtains effective solutions than other meta-heuristics in an acceptable computational time. The genetic algorithm gives inferior solutions in all size of problems as compared to the TS and the SA algorithms. Also in an extremely large-sized problem with 200 cities, the Lingo 8.0 Software cannot obtain an exact solution even after 48 hour. For this problem, the TS algorithm obtained a very promising solution compared to GA and SA in only 441 seconds.

In addition to the MmTSP presented here, there are other variations of the MmTSP with very few meta-heuristics designed that can be investigated, like multiple departure single destination mTSP (MDmTSP) and multiple Hamiltonian path problem (mHPP) [6]. In MDmTSP, there are salesmen that depart from multiple depots and arrive at a single destination, while each remaining node is visited exactly once by any salesman. In mHPP, there is only one depot and all salesmen start from node 1 and finish their tours in node 0 (ending node).

9. Reference

[1] Gorenstein S, (1970), Printing press scheduling for multi-edition periodicals, *Management Science* 16 (6) B373–B383.

[2] Lenstra JK, Rinnooy Kan AHG, (1975), Some simple applications of the traveling salesman problem, *Operations Research Quarterly* 26 717–733.

[3] Brummit B, Stentz A, (April 1996), Dynamic mission planning for multiple mobile robots, in: *Proceedings of the IEEE International Conference on Robotics and Automation*.

[4] Tang L, Liu J, Rong A, Yang Z, (2000), A multiple traveling salesman problem model for hot rolling scheduling in Shangai Baoshan Iron & Steel Complex, *European Journal of Operational Research* 124, 267–282.

[5] Gilbert KC, Hofstra RB, (1992), A new multiperiod multiple traveling salesman problem with heuristic and application to a scheduling problem, *Decision Sciences* 23, 250–259.

[6] Kara I, Bektas T, (2006), Integer Linear Programming Formulations of Multiple Salesmen Problems and its Variations, *European Journal of Operational Research* 174, 1449–1458.

[7] Bektas T, (2006), The multiple traveling salesman problems: an overview of formulations and solution procedures, *Omega* 34, 209 – 219

[8] Wong R, (1980), Integer programming formulations of the traveling salesman problem. In: *Proceedings of the IEEE International Conference of Circuits and Computers*, pp. 149–152.

[9] Little J, Murty D, Sweeney D, Karel C, (1963), An algorithm for the traveling salesman problem. *Operations Research* 11 (6), 972–989.

[10] Shirrish B, Nigel J, Kabuka MR, (1993), A Boolean neural network approach for the traveling salesman problem. *IEEE Transactions on Computers* 42 (10), 1271–1278.

[11] Glover F, (1990). Artificial intelligence, heuristic frameworks and tabu search. *Managerial and Decision Economics* 11 (5), 365–378.

[12] Goldberg DE, Lingle RJ, (1985), Alleles, loci, and the traveling salesman problem. In: *Proceedings of the International Conference on Genetic Algorithms, London*, pp. 154–159.

[13] Lawler EL, Lenstra JK, Rinnooy Kan A, Shimoys D, (1985), The Traveling Salesman Problem. *John Wiley & Sons, New York*.

[14] Chatterjee S, Carrera C, Lynch L, (1996), Genetic algorithms and traveling salesman problems. *European Journal of Operational Research* 93 (3), 490–510.

[15] Malmborg C, (1996), A genetic algorithm for service level based vehicle scheduling. *European Journal of Operational Research* 93 (1), 121–134.

[16] Holland JH, (1975), Adaptation in Natural and Artificial Systems. *MIT Press, Cambridge*, MA.

[17] Jog P, Suh JY, Van Gucht D, (1989), The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. In: *Proceedings of the Third International Conference on Genetic Algorithms, Los Altos, CA*, pp. 110– 115.

[18] Whitley D, Starkweather T, Fuquay D, (1989), Scheduling problems and traveling salesmen: the genetic edge recombination operator. In: *Proceedings of the Third International Conference on Genetic Algorithms, Los Altos, CA*, pp. 133–140.

[19] Poon PW, Carter JN, (1995), Genetic algorithm crossover operators for ordering applications. *Computers and Operations Research* 22 (1), 135–147.

[20] Qu L, Sun R, (1999), A synergetic approach to genetic algorithms for solving traveling salesman problem. *Information Sciences* 117 (3–4), 267–283.

[21] Katayama K, Sakamoto H, Narihisa H, (2000), The efficiency of hybrid mutation genetic algorithm for the traveling salesman problem. *Mathematical and Computer Modeling* 31 (10–12), 197–203.

[22] Potvin J, (1996), Genetic algorithms for the traveling salesman problems. *Annals of Operations Research* 63, 330–370.

[23] Schmitt L, Amini M, (1998), Performance characteristics of alternative genetic algorithmic approaches to the traveling salesman problem using path representation: An empirical study. *European Journal of Operational Research* 108 (3), 551–570.

[24] Schaffer JD, Eshelman LJ, Offutt D, (1991), Spurious correlations and premature convergence in genetic algorithms. In: Rawlings, G.J.E. (Ed.), Foundations of genetic algorithms. *Morgan Kaugmann Publishers, San Mateo, CA*, pp. 102–112.

[25] Ragsdale CT, (2001), Spreadsheet modeling and decision analysis, third ed. *South-Western College Publishing, Cincinnati, OH*.

[26] Goldberg DE, (1989), Genetic Algorithms in Search, Optimization and Machine Learning. *Addison-Wesley, Reading, MA*.

[27] Davis L, (1985), Job shop scheduling with genetic algorithms. In: *Proceedings of the International Conference on Genetic Algorithms, London*, pp. 136–140.

[28] Starkweather T, Whitley D, Whitley C, Mathial K, (1991), A comparison of genetic sequencing operators. In: *Proceedings of the Fourth International Conference on Genetic Algorithms, Los Altos, CA*, pp. 69–76.

[29] Oliver I, Smith D, Holland J, (1987), A study of permutation crossover operators on the traveling salesman problem. In: *Proceedings of the Second International Conference on Genetic Algorithms, London*, pp. 224–230.

[30] Fox BR, McMahon MB, (1991), Genetic operators for sequencing problems. In: Rawlings, G.J.E. (Ed.), Foundations of genetic algorithms. *Morgan Kaugmann Publishers, San Mateo, CA*, pp. 284–300.

[31] Liaw C, (2000), A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research* 124 (1), 28–42.

[32] Knosala R, Wal T, (2001), A production scheduling problem using genetic algorithm. *Journal of Materials Processing Technology* 109 (1–2), 90–95.

[33] Syswerda G, (1989), Uniform crossover in genetic algorithms. In: *Proceedings of the Third International Conference on Genetic Algorithms, Los Altos*, pp. 502– 508.

[34] Moon C, Kim J, Choi G, Seo Y, (2002), An efficient genetic algorithm for the traveling salesman problem with precedence constraints. *European Journal of Operational Research* 140 (3), 606–617.

[35] Tang L, Liu J, Rong A, Yang Z, (2000). A multiple traveling salesman problem model for hot rolling schedule in Shanghai Baoshan Iron and Steel Complex. *European Journal of Operational Research* 124 (2), 267–282.

[36] Kirkpatrick S, Gelatt Jr, Vecchi M, (1983), Optimization by simulated annealing. *Science 220,* 671–680.

[37] Glover F, Laguna M, (1997), Tabu search, *Kluwer Academic*, Boston.

[38] Gendreau M, Laporte G, Potvin JY, (2002), Metaheuristics for the Capacitated VRP. In: Toth P and Vigo D (eds). The Vehicle Routing Problem. *SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA*, pp 129–154.

[39] Cordeau JF, Gendreau M, Laporte G, Potvin JY, Semet F, (2002), A guide to vehicle routing heuristics. *JOPL Res Soc* 53: 512–522.

[40] Van Breedam A, (2001). Comparing descent heuristics and metaheuristics for the vehicle routing problem. *Comput Opns Res* 28: 289–315.

[41] Brandao J (2004), A tabu search algorithm for the open vehicle routing problem. *Eur J Opl Res* 157: 552–564.

[42] Pureza VM, Franca PM, (1991), Vehicle routing problems via tabu search metaheuristic. *Technical Report CRT-347, Centre for Research on Transportation, Montreal, Canada*.

[43] Osman LH, (1993), Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann ofOp ns Res* 41: 421–451.

[44] Gendreau M, Hertz A, Laporte G, (1994), A tabu search heuristic for the vehicle routing problem. *Mngt Sci* 40: 1276– 1290.

[45] Taillard E, Badeau P, Gendreau M, Guertin F, Potvin JY, (1997), A tabu search heuristic for the vehicle routing problem with soft time windows. *Transp Sci* 31: 170–186.

[46] Duhamel C, Potvin JY, Rousseau JM, (1997), A tabu search heuristic for the vehicle routing problem with backhauls and time windows. *Transp Sci* 31: 49–59.