The Solutions to Traveling Salesman Problem

Shuran Liao

Stony Brook University, 300 Circle Road, Stony Brook, NY, USA 11794

Shuran.Liao@stonybrook.edu

Abstract. This paper presents solutions to symmetric, asymmetric, and multiple traveling salesman problems. In the symmetric traveling salesman problem, one salesperson must go to five cities, making precisely one stop at each location. A matrix with the distances between each city is provided. Using the branch and bound algorithm and expressing it in Python, the final result is obtained. The formulations of the asymmetric traveling salesman problem and the multiple traveling salesman problem are demonstrated in the paper. The asymmetric problem in the paper is solved by transforming the asymmetric traveling salesman problem into a symmetric traveling salesman problem; then, the branch and bound algorithm has been applied to solve the problem. In the multiple traveling salesman problem, three salesmen are required to visit seven cities in total, each of which is only visited by one salesperson once. Each city is a point in an x-y plane, and the coordinates of all points are given in a graph. By formulating the MTSP problem with an assignment-based double-index integer programming and applying the constraints, the outcome is derived from Python code within a few seconds.

Keywords: Symmetric Traveling Salesman Problem, asymmetric Traveling Salesman Problem, Multiple Traveling Salesman Problem, branch and bound, Python.

1. Introduction

The Traveling Salesman Problem, or TSP for short, is a famous combinatorial optimization problem that was first proposed in 1930. TSP is a tremendously interesting problem for mathematicians and computer scientists since it is simple to express yet challenging to solve. The objective of the TSP is to identify the quickest path for a salesperson that travels from city to city precisely once before returning to the starting point. The problem is working out the ideal arrangement of the stated cities since the sequence in which the cities are visited during a journey impacts the distance that is traveled. The TSP is categorized as an NP-hard issue because finding the optimum route becomes more difficult as the number of cities rises. There are several approaches to solving the TSP, among which heuristics, linear programming, and branch and bound are examples of successful approaches.

There are symmetric and asymmetric divisions in the TSP. The symmetric Traveling Salesman Problem (STSP) seeks to discover the shortest Hamiltonian cycle in a weighted, finite, and undirected network without loops. There are a variety of approaches to solving the symmetric traveling salesman problem. Padberg and Rinaldi utilized the branch and cut technique to solve a symmetric traveling salesman problem with 532 cities [1]. John applied tabu search (TS), a metaheuristic search that is a metaheuristics algorithm used for optimization, to solve the problem [2]. An enhanced tabu search strategy was proposed by Lim et al. for the symmetric traveling salesman problem. They incorporated the two heuristic methods of tabu search and simulated annealing, and they tested the technique on five chosen symmetric TSP benchmark problems [3].

The distance function in the asymmetric traveling salesman problem is not symmetric, which implies that the distance from city u to city v differs from that from city v to city u. Many ways to solve the asymmetric traveling salesman issue have been established and developed. One effective algorithm is the branch and bound. Carpaneto et al. presented a lowest-first, branch-and-bound method, which is based on the assignment problem relaxation and a sub-tour elimination branching strategy. A large number of vertices is randomly generated, and the effectiveness of the algorithm is tested to be high [4]. Similarly, the branch-and-cut algorithm has also been proven to be effective in optimally solving problems with more than 200 vertices in a few minutes of CPU time [5]. Jonker

and Volgenant transformed asymmetric traveling salesman problems into symmetric ones. They converted an asymmetric matrix of size n×n to a symmetric matrix of size 2n×2n, and then replicated each mode in the network to generate a second ghost node with a very low weight. The optimal solution was obtained by merging the original and ghost nodes. Branch and bound is a successful approach to solving the TSP. The approach employs a branching process that divides the set of all tours (possible solutions) into progressively compact subsets. The length of the trips inside each subgroup is determined as a lower bound. Eventually, a subset of each trip that is shorter than or equal to a predetermined lower limit for each tour is identified. [6].

The original TSP only has one salesman; however, in order to solve some real problems, multiple salesmen are needed. As a result, the Traveling Salesman Problem (TSP) is expanded into the Multiple Traveling Salesman Problem (MTSP), where more than one salesman may be utilized in the situation. In the MTSP, a list of locations is visited by several salesmen such that each city is only traveled once by a single salesperson. The objective is to reduce the length of the longest sub-tour or the sum of all sub-tours taken by all salesmen as much as possible. The MTSP is an important combinatorial optimization problem with a great variety of real-like applications, such as robotics, transportation, networking, and so forth. The branch and bound approach may be used to solve MTSP, just like it can for symmetric and asymmetric TSP. Gavish and Srikanth created lower limits by a Lagrangean relaxation and updated the Lagrange multipliers using a sub-gradient optimization technique. They examined their algorithm using 500 cities and 10 salesmen on 410 cases and developed an efficient branch-and-bound based approach [7]. Additionally, another exact-solution method was suggested by Gromicho and other researchers, who created a novel method and demonstrated its effectiveness using 120 nodes and 2 to 12 salespeople. A quasi-assignment relaxation produced by relaxing the SECs serves as the system's foundation. Wacholder et al. converted the multiple salesmen problem into the conventional one salesman problem in order to construct an effective neutral network method for solving the MTSP. Their method was tested on problems with as many as 30 cities and 5 salesmen and gained valid solutions [8]. More accurate answers can be swiftly acquired and more real-world issues may be handled with a large number of scholars formulating and improving the solutions to the TSP.

In this paper, the exact solution to symmetric, asymmetric, and multiple Traveling Salesman Problem will be demonstrated, and the final result will be gained from Python.

2. Methodology

2.1. Branch and Bound Algorithm

In the presence of a complete weighted undirected graph $G = (\{1, \cdots, n\}, E)$ and a matrix A on G that has $a_{ij} = a_{ji}$ for all costs in A. T is the path across G that only visits each vertex once.
The branch-and-bound technique is divided into four parts: a method for computing the lower limit through relaxation, a branching method, heuristic solutions for bounding, and techniques to condense the solution space [9]. The following are the specifics of each procedure.

The Lagrange technique of 1-tree relaxation for the traveling salesperson problem was first published by Held and Karp [10]. In the graph, one cycle constitutes one tree, which is a network that consists of an extra edge. Constructing a minimal 1-tree while adhering to the requirement that every node with degree 2 bears a strong similarity to the TSP is a challenge. Relaxing the degree constraints yields the analogous Lagrange problem. Before branching a subproblem, a sub-gradient approach is utilized to boost the lower bound as every solution to a Lagrange problem is a lower limit of the TSP. After dividing the current set of feasible solutions into subsets, A branching approach sets a lower bound on the expense of solutions in each subset. Although this improvement is not always achievable, the branching should boost the lower bounds for each of the new subproblems that occur. A derived subproblem SP denotes the collection of all 1-tree TB. F that contains the edges of R but omits the edges of F is represented by a derived subproblem SP (R, F). The cost of each tour for the derived

subproblem SP is thus power limit by the weight WR, F(K) of the minimal 1-tree of $T_{R,F}$. For the derived subproblem SP, F is a lower constraint on the price of any tour (R, F). Volgenant and Jonker's heuristic is used to choose needed or prohibited edges. Branching splits the subproblem SP (R, F) in the scenario below into two or three separate subproblems.

A big 0-1 integer program, the traveling salesman issue. Every approach to solving zero-one integer programs aims to condense the solution space or get rid of variables that can be proved to have either a value of 0 or 1 in any ideal solution. This entails the TSP discovering duplicate edges which should not be included in an optional tour and detecting superfluous edges that are required to be included in any ideal tour. It is obvious that although unnecessary edges can belong in the set F of prohibited edges, required edges can go in the set R of necessary edges. We utilized two techniques (edge exchange and non-optimal edge identification) to boost the proportion of banned and necessary edges provided by Volgenant arid Jonker [11].

The branch and bound algorithm will be employed in solving the symmetric TSP, which aims to determine the quickest path for a salesperson who must travel across five cities.

2.2. Formulation of Asymmetric Traveling Salesman Problem

If $d_{rs} \neq d_{sr}$, the TSP problem becomes asymmetric. The following is a possible way to formulate the asymmetric traveling salesman problem.

Minimize

$$\sum_{i \neq j} c_{ij} x_{ij} \tag{1}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \ (i \in \mathrm{V},\ i \neq j) \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \ (j \in \mathrm{V},\ j \neq i) \tag{3}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \ (\mathrm{S} \subset V,\ 2 \leq |S| \leq n-2) \tag{4}$$

$$x_{ij} = 0 \text{ or } 1 (i,\ j) \in A \tag{5}$$

A specific question within the range of the asymmetric traveling salesman problem will be described in more detail below and will be solved using the formulation.

2.3. Formulations of Multiple Traveling Salesman Problem

The assignment-based double-index integer linear programming method is typically used to develop MTSP. Firstly, the following binary variable is defined:

$$x_{ij} = \begin{cases} 1 \ If\ edge\ (i,j)\ is\ used\ during\ travel \\ 0 \ Otherwise \end{cases} \tag{6}$$

The following is a generic formula for the assignment-based directed integer linear programming formulation of the MTSP:

Minimize

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\, x_{ij} \tag{7}$$

Subject to

$$\sum_{j=2}^{n} x_{1j} = m \tag{8}$$

$$\sum_{j=2}^{n} x_{j1} = m \tag{9}$$

$$\sum_{i=1}^{n} x_{ij} = 1,\ j = 2, \cdots\cdots, n \tag{10}$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 2, \cdots\cdots, n \tag{11}$$

+ other restrictions on elimination

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A$$

Based on the formulation of the Multiple Traveling Salesman Problem, Python code will be generated to solve the problem of three salesmen visiting seven cities discussed below.

3. Results and discussion

For a few cities, exact options that offer a worldwide ideal route already exist. The needed computation time increases more quickly as the number of cities increases. When there are few cities, branch and bound integer programming can produce precise results. Heuristic algorithms that approach the correct solution are typically used to handle problems involving a large number of cities. The following illustrates the exact solutions to symmetric, asymmetric, and multiple traveling salesman problems.

3.1. Solving Symmetric Traveling Salesman Problem

The following definition applies to the symmetric traveling salesman problem. Let V = $\{v_1, v_2, \cdots, v_n\}$ be the collection of locations, A = {(r, s): r, s $\in$ V} be the set of the edges, d_{rs} and d_{sr} be the cost measures connected to the edge (r, s) $\in$ A. If $d_{rs} = d_{sr}$ holds in every (r, s), The problem is symmetric.

In the symmetric problem, cities serve as the vertices of the graph, and the weight of each edge between them determines how the symmetric problem may be represented as an undirected weighted graph. The problem can be solved using Python in several ways.

Consider the following question. There are five cities, and the following matrix expresses their distance from one another. After visiting all four locations, a salesman starts in city 0 and ends up back there. The goal is to locate the shortest path.

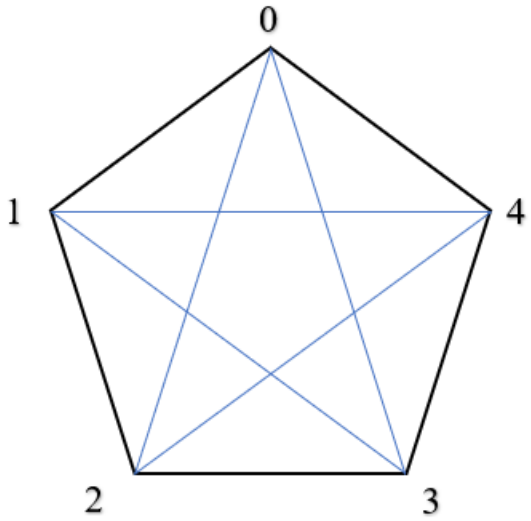

Figure 1. This graph demonstrates five vertexes and the existing nodes between the five vertexes

The distances between each pair of the city are given in the matrix below. The row above the matric indicates the starting cities and the column beside the matrix indicates the destination city. The matrix below is symmetric.

$$A = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 20 & 10 & 15 & 25 \\ 20 & 0 & 12 & 30 & 35 \\ 10 & 12 & 0 & 25 & 15 \\ 15 & 30 & 25 & 0 & 20 \\ 25 & 35 & 15 & 20 & 0 \end{bmatrix} \end{array} \quad (12)$$

This problem can be solved using the branch and bound method, which can be briefly explained using a state-space tree depicted below. The tree illustrates all the possible routes that the salesman can travel, and if adding up all the distances and comparing them, we can find the shortest one.

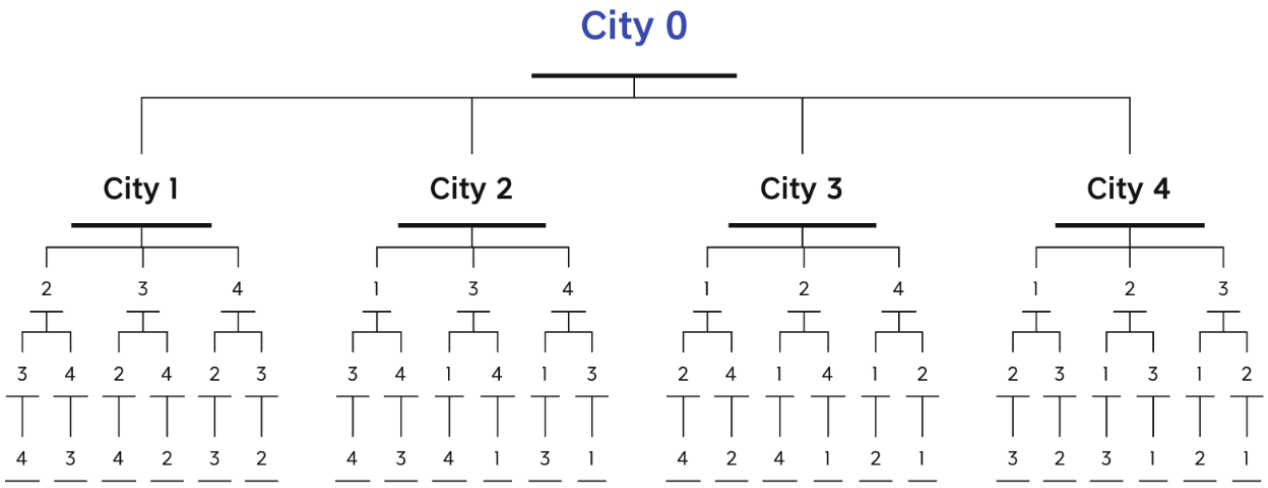

Figure 2. The state-space tree of the problem

In the picture above, the top level is level 1, with "City 0" as the starting point. Level 2 is the four branches representing the salesman's four routes after city 0. Level 3 is the collection of 12 branches, and levels 4 and 5 are the collection of 24 branches. Each branch represents a route between two adjacent cities. Below is an illustration of the problem's solution, which was constructed using the state-space tree concept.

The branch and bound approach compute a bound on the best practical solution for the current node in the tree. The subtree rooted with the node is omitted if the bound on the best feasible solution is worse than the current best. On a node, two costs are taken into account: the cost of getting to the node from the root and the cost of getting to a leaf from the present node. To decide whether to disregard the subtree containing this node, a constraint on the second was determined. The tour's price may be expressed as follows.

$$Cost\ of\ a\ tour\ T\ = \frac{1}{2} * \sum\ (cost\ of\ two\ edges\ adjacent\ to\ u\ and\ in\ T) \quad (13)$$
$$where\ u\ \in\ V$$

If two edges traveling through each vertex (u) were taken into consideration, the overall cost for all vertices would be double the price of the journey. Thus,

$$(Sum\ of\ two\ edges\ adjacent\ to\ u) \geq (Sum\ of\ minimum\ weight\ two\ edges\ adjacent\ to\ u) \quad (14)$$

$$Cost\ of\ any\ tour \geq \frac{1}{2} * \sum(Sum\ of\ cost\ of\ two\ minimum\ weight\ edges, adjacent\ to\ u) \quad (15)$$
$$where\ u\ \in\ V$$

Table 1. Minimum costs of two edges adjacent to all nodes

Vertex	Lowest weight edges	Total cost
0	(0, 2), (0. 3)	25
1	(0, 1), (1. 2)	32
2	(0, 2), (1, 2)	22
3	(0, 3), (3, 4)	35
4	(2, 4), (3, 4)	35

Thus, the lowest possible cost of each tour is equal to ½ (25 + 32 + 22 + 35 + 35) = 74.5.

Computing the problem using Python, it has been found that the optimal route is 0 – 1 – 2 – 4 – 3 – 0, and the minimum distance is 82.

3.2. Solving Asymmetric Traveling Salesman Problem

In the asymmetric traveling salesman problem, the distance from city i to city j differs from the distance from j to i. The following 3×3 matrix shows the weight between the cities 1, 2, and 3.

$$A = \begin{matrix} \\ 1 \\ 2 \\ 3 \end{matrix} \begin{matrix} \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{bmatrix} 0 & 10 & 2 \\ 32 & 0 & 22 \\ 15 & 32 & 0 \end{bmatrix} \end{matrix} \tag{16}$$

Making the asymmetric situation symmetric is one approach to solving this problem. The size of the matrix will be doubled, and there will be 8 nodes in the problem. $1'$ is the ghost node of node 1, $2'$ is the ghost node of node 2, and $3'$ is the ghost node of node 3. The ghost node is located at a distance of -w from the original node, which needs to be small enough to guarantee that all ghost edges are part of the ideal symmetric traveling salesman problem solution on the new graph.

$$A' = \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 1' \\ 2' \\ 3' \end{matrix} \begin{matrix} \begin{matrix} 1 & 2 & 3 & 1' & 2' & 3' \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & -w & 32 & 15 \\ 0 & 0 & 0 & 10 & -w & 32 \\ 0 & 0 & 0 & 12 & 22 & -w \\ -w & 10 & 12 & 0 & 0 & 0 \\ 32 & -w & 22 & 0 & 0 & 0 \\ 15 & 32 & -w & 0 & 0 & 0 \end{bmatrix} \end{matrix} \tag{17}$$

The branch and bound strategy, as mentioned above, may be utilized to determine the solution to this issue in the symmetric multiple traveling salesman problem.

After running Python code, the optimal symmetric tour is $1 \rightarrow 1' \rightarrow 3 \rightarrow 3' \rightarrow 2 \rightarrow 2' \rightarrow 1$. By merging the original and ghost nodes, the solution to the asymmetric problem is $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

3.3. Solving Multiple Traveling Salesman Problem

Find the shortest distance in an x-y plane for three salesmen who begin at point O and travel through points A to F. The points and coordinates of the points are given in the chart below.

Table 2. Points and coordinates of points of 3 traveling salesmen problem

Points	Coordinates
O	(0, 0)
A	(12, 57)
B	(14, 26)
C	(15, 50)
D	(10, 35)
E	(9, 60)
F	(20, 45)

Below is the pseudocode of the solution. The first part of the pseudocode defines the objective function and the constraints. The second part of the pseudocode obtains the path of the three salesmen.

```
Set points to coordinates of seven points

Set n to 7
Initialize C to a 7 × 7 zero matrix
For i = 0 To n
        For j = 0 To n
                Set i-th row, j-th column in C to the distance between point i and point j
        Endfor
Endfor
Set A to a 7 × 7 matrix with all Boolean entries
Set u to a vector with seven integer entries
Set m to 3
Set ones to a n × 1 matrix with all entries of 1
Set objective to minimize the sum of the multiplication of matrices C and A
Set constraints to an empty list
constraints ← constraints + array of whether the sum of all entries of first row in X is equal to m
constraints ← constraints + array of whether the sum of all entries of first column in X is equal to m
constraints ← constraints + array of whether the individual sum of all entries in each row except the first row is
equal to 1
constraints ← constraints + array of whether the individual sum of all entries in each column except the first
column is equal to 1
constraints ← constraints + array of whether each diagonal entry of X is equal to 0
constraints ← constraints + array of whether each entry except the first entry in u is greater than 2
constraints ← constraints + array of whether each entry except the first entry in u is less than n
constraints ← constraints + array of whether the first entry in u is equal to 1
For i = 0 To n
        For j = 0 To n
                If i ≠ j
                        constraints ← constraints + array of whether (u[i] - u[j] + 1) is no more than (n - 1) *
(1 - X[i, j]) ]
    Endif
        Endfor
Endfor
```

Figure 3. Pseudocode part I

```
Set A_1 to an array of the indices of entries in A that are equal to 1
Set path to an empty library
For i = 0 To m
        Initialize Salesman (i + 1) to 0
        Set j to i
        Set a to nearly infinity
        while a ≠ 0
                Set a to the first column in A_1
                Path of Salesman (i + 1) append a
                Set j to an array that satisfies the first column in A_1 is equal to a
                Set j to the first entry in j
                Set a to j
        Endwhile
                                        Endfor
Output path
```

Figure 4. Pseudocode part II

After executing the Python code, the result is that the route of salesperson 1 is $0 \rightarrow 3 \rightarrow 0$, the route of salesperson 2 is $0 \rightarrow 4 \rightarrow 0$, and route of salesperson 3 is $0 \rightarrow 6 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 0$. Although the Multiple Traveling Salesman Problem needs more time to compute, all problems are easily solved by Python within seconds.

4. Conclusion

There are two types of the Traveling Salesman Problem: symmetric TSP and asymmetric TSP. The Multiple Traveling Salesman Problem is a variation of the original TSP. The symmetric TSP is relatively easy to solve using branch and bound algorithms and Python. The asymmetric TSP is more difficult to solve, but we can transform the asymmetric TSP into symmetric TSP and then solve it. The Multiple Traveling Salesman Problem is also difficult to solve, but with the help of Python, we can gain the results within seconds. TSP has a wide range of applications, including vehicle routing problems, job-shop scheduling, and computer wiring. To acquire accurate or approximate answers to particular traveling salesman issues, many strategies have been presented. Among all the methods, the branch and bound approach has been shown to accurately solve symmetric, asymmetric, and multiple traveling salesman problems with a large number of cities. Also, transforming asymmetric problems into symmetric problems is also an effective approach to solving relatively difficult asymmetric problems. Additionally, computer programing now helps people solve the Traveling Salesman Problem much more quickly and accurately. It is hopeful that it will be easier and easier to solve problems that can be converted into the Traveling Salesman Problem in the future with more advanced approaches and effective computing tools.

References

[1] Padberg, M. and Rinaldi, G., “Optimization of a 532-City Symmetric Traveling Salesman Problem by Branch and Cut.” Operations Research Letters, vol. 6, no. 1, 1–7 (1087).

[2] Knox, J. “Tabu Search Performance on the Symmetric Traveling Salesman Problem.” Computers & Operations Research, vol. 21, no. 8, 867–876 (1984).

[3] Lim, Y. F., Hong, P. Y., Ramli, R., and Khalid, R., "An improved tabu search for solving symmetric traveling salesman problems," 2011 IEEE Colloquium on Humanities, Science and Engineering, 851-854 (2011).

[4] Carpaneto, G., et al. “Exact Solution of Large-Scale, Asymmetric Traveling Salesman Problems.” ACM Transactions on Mathematical Software, vol. 21, no. 4, 394–409 (1995).

[5] Ascheuer, N., Jünger, M. and Reinelt, G. “A Branch & Cut Algorithm for the Asymmetric Traveling Salesman Problem with Precedence Constraints.” Computational Optimization and Applications 17, 61–84 (2000).

[6] Jonker, R., and Volgenant. T., “Transforming Asymmetric into Symmetric Traveling Salesman Problems.” Operations Research Letters, vol. 2, no. 4, 161–163 (1983).

[7] Gavish, B., and Srikanth, T., "An optimal solution method for large-scale multiple traveling salesmen problems." Operations Research 34.5 (1986).

[8] Gromicho, J., Paixão, J. and Bronco, I., "Exact solution of multiple traveling salesman problems." Combinatorial optimization. Springer, Berlin, Heidelberg, 291-292 (1992).

[9] Held, M. and Karp, R.M., “The travelingsalesmanproblem and minimum spanning trees: part ZI.” Mathematical Programming 1, 6-25 (1971).

[10] Volgenant, T., Jonker, R., “Nonoptimal Edges for the Symmetric Traveling Salesman Problem. Operations Research” vol, 32, no. 4, 65-74 (1984).

[11] Matai, R., Singh, S. P., and Mittal, M. L., "Traveling salesman problem: an overview of applications, formulations, and solution approaches." Traveling salesman problem, theory and applications 1 (2010).