SSRG International Journal of Recent Engineering Science
ISSN: 2349–7157 / https://doi.org/10.14445/23497157/IJRES-V8I1P102

Volume 8 Issue 1, 6-10, Jan-Feb 2021
© 2021 Seventh Sense Research Group®

Original Article

A New Column-Row Method for Traveling Salesman Problem: The Dhouib-Matrix-TSP1

Souhail Dhouib

Higher Institute of Industrial Management, Sfax University.

Received: 05 January 2021 Revised: 07 February 2021 Accepted: 16 February 2021 Published: 26 February 2021

Abstract *- In this paper, a new column-row method named Dhouib-Matrix-TSP1 is designed to solve the Traveling Salesman Problem (TSP) in polynomial time. At first, the distance matrix is defined, and then four steps are launched: 1) Selecting the starting position, 2) Choosing Rows, 3) Discarding by column 3) Transforming the route to a tour.*

Some numerical examples are presented to illustrate the effectiveness of the proposed method. It can be concluded that the Dhouib-Matrix-TSP1 method consumes a small number of iterations (just n iterations, where n represents the number of cities) to solve the TSP, and its result is the closest to the optimum solution.

Keywords *- Traveling Salesman Problem, Combinatorial opptimization, Polynomial time.*

1. Introduction

The Traveling Salesman Problem (TSP) is a fundamental combinatorial optimization model studied in the decision-making theory. It deals with finding the shortest tour that visits each city in a given list exactly once and then returns to the starting city. The TSP is formulated mathematically as flows:

$$\text{Optimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \tag{1}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1\,,\ \ i = 1,\dots,n$$
$$\sum_{i=1}^{n} x_{ij} = 1\,,\ \ j = 1,\dots,n$$
$$x_{ij} = 0 \ or\ 1,\ i = 1,\dots,n\,,\ \ \mathrm{j} = 1,\dots,n. \tag{2}$$

Where d_{ij} is the distance (the cost or the time) from city i to city j and x_{ij} is used to be the binary value (if the route from i to j is chosen, then x_{ij} =1; otherwise, x_{ij} =0). Eq1. Represents the function that minimizes the distance between the cities, whereas Eq2. Deals with the restriction of no city are visited twice.

In recent years, many papers have dealt with TSP. [1] provides a comprehensive review of existing applications, approaches, and taxonomy on multiple TSP. [2] presents a new kind of problem by combining the scheduling problem with the TSP, where a traveler moves through n locations (nodes), visits all the locations and each location exactly once to assign and initiate one of the n jobs, and then returns to the first location. [3] explores the Energy Minimization Traveling Salesman Problem (EMTSP), where the sum of the product of load (including curb weight of the vehicle) and traveled distances are minimized. [4] deals with maximizing the profit per unit of time: the profit is calculated by subtracting the route costs from the profit divided by the total time required to complete the tour. [5] explores the TSP with release dates and drone resupply; this paper consists of finding a minimum route time for a single truck that can receive newly available orders on route via a drone sent from the depot. [6] designs and codes a multi-thread-based fast algorithm with Delphi language for the medium and large-scale TSP instances from TSPLIB. [7] presents a comparative study of the alternative mathematical formulations for TSP with hotel selection. [8] develops a Variable Neighborhood Search-based heuristic solution method to solve the Hybrid Electric Vehicle TSP with time windows. [9] generates the exact Pareto set in Multi-Objective Traveling Salesman and Set Covering Problems. [10] proposes a new decomposition-based technique and probabilistic model-based methods to tackle a newly designed multi-objective TSP test suite. [11] applies the honey bees mating optimization metaheuristic for the Euclidean TSP.

This paper attempts to propose an original column and row generation method, named the Dhouib-Matrix-TSP1 method, to optimize the TSP in polynomial time (exactly in n iterations). The next section describes the proposed method, where section three gives a detailed example, computational results will be presented in section four, and finally, a conclusion will be given.

2. The Proposed Method: Dhouib-Matrix-TSP1

The Dhouib-Matrix-TSP1 method follows a very simple procedure composed of four steps :

Step 1: At first, find the minimal element of each row in the distance matrix and write it (a_i) on the right-hand side of the matrix. Then, select the smallest element in the list of a_i and determine its position in the matrix (a numerical example will be given in section 3). The position of the smallest element will specify two cities, *x*, and *y*, which will be inserted in the proposed route named *list-cities* {*x*,*y*}. To finish this step, discard the columns of the inserted cities (*x* and *y*).

Step 2: Select the left element in the proposed route, which is (*x*), and the right one, which is (*y*). Next, find the smallest element for each row: for the row of *x* and the row of *y*. Then, select the minimum distance; let it be the city *z*.

Step 3: Now, insert the last selected city *z* in the proposed *list-cities* at right if it was selected by *y* {*x*-*y*-*z*} or left if it was selected by *x* {*z*-*x*-*y*}. After that, discard the column of *z* and test: if all cities are in the list, so go to Step 4. Else return to Step 2.

Step 4: Transform the proposed route *list-cities in*to a tour by starting and ending with the same city. Accordingly, the starting city must be in the first position on the proposed route. Nevertheless, change the position of the first element to the end (one by one; a detailed example will be given in section 3) until the starting city is at the first position. Finally, add the starting city to the end of the proposed route, and it will be a tour.

3. The Proposed Method: Dhouib-Matrix-TSP1

This section presents an example that may be useful to explain the four steps of the proposed method: Dhouib-Matrix-TSP1.

From / To	Bev.	Sal.	Wey.	Bra.	Smi.	Dan.	Wob.	Spr.
Beverly	0	13	45	50	95	15	32	140
Salem	15	0	40	44	85	22	43	122
Weymouth	42	35	0	7	42	55	40	145
Braintree	48	39	10	0	47	58	45	155
Smithfield	105	90	35	42	0	98	100	202
Danvers	12	24	52	55	102	0	35	148
Woburn	35	42	32	42	102	33	0	112
Springfield	135	125	142	150	212	150	110	0

Let's consider the distances matrix from [12] in time between eight cities (Beverly, Salem, Weymouth, Braintree, Smithfield, Danvers, Woburn, and Springfield). We are looking to design a tour for a salesman to minimise the total distance.

Step 1: At first, find the minimal element of each row in the distance matrix and write it (a_i) on the right-hand side of the matrix as follows:

$$\begin{bmatrix} - & \boxed{13} & 45 & 50 & 95 & 15 & 32 & 140 \\ \boxed{15} & - & 40 & 44 & 85 & 22 & 43 & 122 \\ 42 & 35 & - & \boxed{7} & 42 & 55 & 40 & 145 \\ 48 & 39 & \boxed{10} & - & 47 & 58 & 45 & 155 \\ 105 & 90 & \boxed{35} & 42 & - & 98 & 100 & 202 \\ \boxed{12} & 24 & 52 & 55 & 102 & - & 35 & 148 \\ 35 & 42 & 32 & 42 & 102 & \boxed{33} & - & 112 \\ 135 & 125 & 142 & 150 & 212 & 150 & \boxed{110} & - \end{bmatrix} \quad \begin{matrix} 13 \\ 15 \\ \textcircled{7} \\ 10 \\ 35 \\ 12 \\ 32 \\ 110 \end{matrix}$$

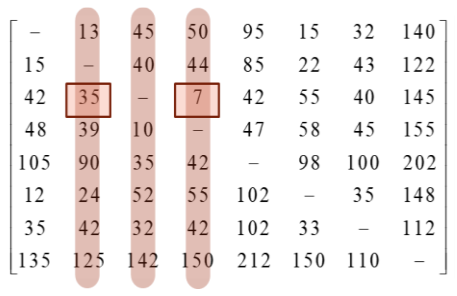
Then, select the smallest element in the list of a_i; in this example, it is 7. Next, determine the position of the smallest element (7) in the corresponding row, which is (d_{34}), and insert these cities in the proposed route named *list-cities* {3-4}. In reality, the distance (d_{34}) represents the minimum distance between all cities in the matrix. Instead of eliminating the visit of these cities in the next iterations, we discard their columns from the matrix: column 3 and column 4.

$$\begin{bmatrix} - & 13 & 45 & 50 & 95 & 15 & 32 & 140 \\ 15 & - & 40 & 44 & 85 & 22 & 43 & 122 \\ 42 & 35 & - & \boxed{7} & 42 & 55 & 40 & 145 \\ 48 & 39 & 10 & - & 47 & 58 & 45 & 155 \\ 105 & 90 & 35 & 42 & - & 98 & 100 & 202 \\ 12 & 24 & 52 & 55 & 102 & - & 35 & 148 \\ 35 & 42 & 32 & 42 & 102 & 33 & - & 112 \\ 135 & 125 & 142 & 150 & 212 & 150 & 110 & - \end{bmatrix}$$

3-4
Distance: 7
Discard columns 3 et 4

Step 2: Select the left element in the proposed route *list-cities*, which is (3) and the right one which is (4). Then, find the minimum element for each row: for row 3, it is (35), and for row 4, it is (39). After that, select the smallest distance, which is 35, between cities 3 and 2.

Step 3: Now, insert the last select city, which is 2, in the proposed route *list-cities* {2-3-4} and discard its column, then return to Step 2.

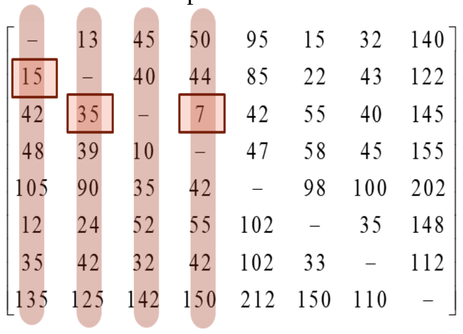


$$\begin{bmatrix} - & 13 & 45 & 50 & 95 & 15 & 32 & 140 \\ 15 & - & 40 & 44 & 85 & 22 & 43 & 122 \\ 42 & \boxed{35} & - & \boxed{7} & 42 & 55 & 40 & 145 \\ 48 & 39 & 10 & - & 47 & 58 & 45 & 155 \\ 105 & 90 & 35 & 42 & - & 98 & 100 & 202 \\ 12 & 24 & 52 & 55 & 102 & - & 35 & 148 \\ 35 & 42 & 32 & 42 & 102 & 33 & - & 112 \\ 135 & 125 & 142 & 150 & 212 & 150 & 110 & - \end{bmatrix}$$

2-3-4
Distance: 42
Discard column 2

Step 2: Select the left element in the proposed route *list-cities*, which is (2) and the right one which is (4). Then, find the minimal element for each row: for row 2, it is (15), and for row 4, it is (45). Then, select the smallest distance, which is 15, between City 2 and City 1.

Step 3: Insert the last selected city, which is 1 in the proposed route *list-cities* {1-2-3-4} and discard its column, then return to Step 2.

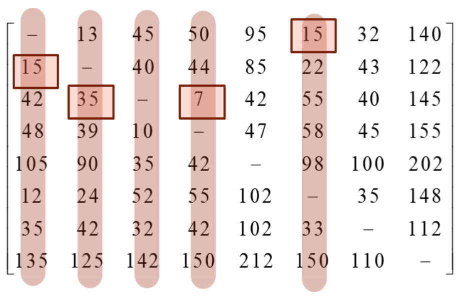


$$\begin{bmatrix} - & 13 & 45 & 50 & 95 & 15 & 32 & 140 \\ \boxed{15} & - & 40 & 44 & 85 & 22 & 43 & 122 \\ 42 & \boxed{35} & - & \boxed{7} & 42 & 55 & 40 & 145 \\ 48 & 39 & 10 & - & 47 & 58 & 45 & 155 \\ 105 & 90 & 35 & 42 & - & 98 & 100 & 202 \\ 12 & 24 & 52 & 55 & 102 & - & 35 & 148 \\ 35 & 42 & 32 & 42 & 102 & 33 & - & 112 \\ 135 & 125 & 142 & 150 & 212 & 150 & 110 & - \end{bmatrix}$$

1-2-3-4
Distance: 57
Discard column 1

Step 2: Select the left element in the proposed route *list-cities*, which is (1) and the right one which is (4). Then, find the minimal element for each row: for row 1, it is (15), and for row 4, it is (45). After that, select the smallest distance, which is 15, between City 1 and City 6.

Step 3: Insert the last selected city, which is 6, in the proposed route *list-cities* {6-1-2-3-4} and discard its column, then return to Step 2.

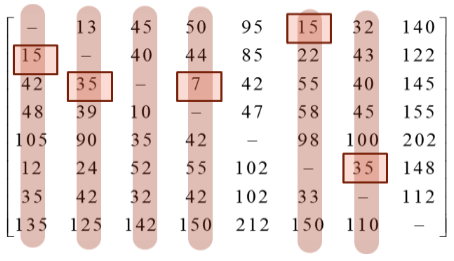


$$\begin{bmatrix} - & 13 & 45 & 50 & 95 & \boxed{15} & 32 & 140 \\ \boxed{15} & - & 40 & 44 & 85 & 22 & 43 & 122 \\ 42 & \boxed{35} & - & \boxed{7} & 42 & 55 & 40 & 145 \\ 48 & 39 & 10 & - & 47 & 58 & 45 & 155 \\ 105 & 90 & 35 & 42 & - & 98 & 100 & 202 \\ 12 & 24 & 52 & 55 & 102 & - & 35 & 148 \\ 35 & 42 & 32 & 42 & 102 & 33 & - & 112 \\ 135 & 125 & 142 & 150 & 212 & 150 & 110 & - \end{bmatrix}$$

6-1-2-3-4
Distance: 72
Discard column 6

Step 2: Select the left element in the route *list-cities*, which is (6) and the right one which is (4). Then, find the minimum element for each row: for row 6, it is (35), and for row 4, it is (45). After that, select the smallest distance, which is 35, between City 7 and City 6.

Step 3: Insert the last selected city, which is 7 in the proposed route *list-cities* {7-6-1-2-3-4} and discard its column, then return to Step 2.

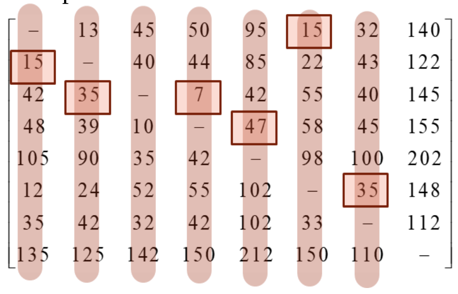


$$\begin{bmatrix} - & 13 & 45 & 50 & 95 & \boxed{15} & 32 & 140 \\ \boxed{15} & - & 40 & 44 & 85 & 22 & 43 & 122 \\ 42 & \boxed{35} & - & \boxed{7} & 42 & 55 & 40 & 145 \\ 48 & 39 & 10 & - & 47 & 58 & 45 & 155 \\ 105 & 90 & 35 & 42 & - & 98 & 100 & 202 \\ 12 & 24 & 52 & 55 & 102 & - & \boxed{35} & 148 \\ 35 & 42 & 32 & 42 & 102 & 33 & - & 112 \\ 135 & 125 & 142 & 150 & 212 & 150 & 110 & - \end{bmatrix}$$

7-6-1-2-3-4
Distance: 107
Discard column 7

Step 2: Select the left element in the proposed route *list-cities*, which is (7) and the right one which is (4). Then find the minimal element for each row: for row 7, it is (102), and for row 4, it is (47). Then, select the smallest distance, which is 47, between City 5 and City 4.

Step 3: Insert the last select city, which is 5 in the proposed *list-cities* {7-6-1-2-3-4-5} and discard its column, then return to Step 2.

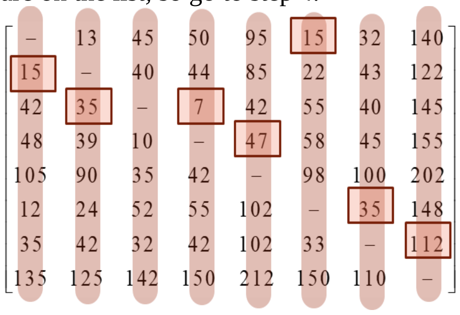


$$\begin{bmatrix} - & 13 & 45 & 50 & 95 & \boxed{15} & 32 & 140 \\ \boxed{15} & - & 40 & 44 & 85 & 22 & 43 & 122 \\ 42 & \boxed{35} & - & \boxed{7} & 42 & 55 & 40 & 145 \\ 48 & 39 & 10 & - & \boxed{47} & 58 & 45 & 155 \\ 105 & 90 & 35 & 42 & - & 98 & 100 & 202 \\ 12 & 24 & 52 & 55 & 102 & - & \boxed{35} & 148 \\ 35 & 42 & 32 & 42 & 102 & 33 & - & 112 \\ 135 & 125 & 142 & 150 & 212 & 150 & 110 & - \end{bmatrix}$$

7-6-1-2-3-4-5
Distance: 154
Discard column 5

Step 2: Select the left element in the proposed route *list-cities*, which is (7), and the right one, which is (5). Then, find the minimal element for each row: for row 7, it is (112), and for row 5, it is (202). Then, select the smallest distance, which is 112, between City 8 and City 7.

Step 3: Insert the last selected city, 8, in the proposed route *list-cities* {8-7-6-1-2-3-4-5} and discard its column. All cities are on the list, so go to step 4.



$$\begin{bmatrix} - & 13 & 45 & 50 & 95 & \boxed{15} & 32 & 140 \\ \boxed{15} & - & 40 & 44 & 85 & 22 & 43 & 122 \\ 42 & \boxed{35} & - & \boxed{7} & 42 & 55 & 40 & 145 \\ 48 & 39 & 10 & - & \boxed{47} & 58 & 45 & 155 \\ 105 & 90 & 35 & 42 & - & 98 & 100 & 202 \\ 12 & 24 & 52 & 55 & 102 & - & \boxed{35} & 148 \\ 35 & 42 & 32 & 42 & 102 & 33 & - & \boxed{112} \\ 135 & 125 & 142 & 150 & 212 & 150 & 110 & - \end{bmatrix}$$

8-7-6-1-2-3-4-5
Distance: 266
Discard column 8

Step 4: Transfer the proposed route *list-cities* {8-7-6-1-2-3-4-5} to a tour by starting and ending with the same city (1). Accordingly, change the position of the first element to the end of the list until city number 1 is at position 1. For example, change the position of city 8 to the last position after city 5, transfer city 7 after city 8, and so on. Then, we obtain the {1-2-3-4-5-8-7-6}. Finally, add city 1 to the end of the route {1-2-3-4-5-8-7-6-1} and compute the total distance, which will be equal to 464. Our method Dhouib-Matrix-TSP1 is better (19 units in less) than the Hungarian method (which is 483).

4. The Computational Results

The Python language is used to implement the proposed Dhouib-Matrix-TSP1 method, and some examples will be used to prove its efficiency in finding the optimal or the near-optimal solution in polynomial time (only *n* iterations, where *n* is the number of cities).

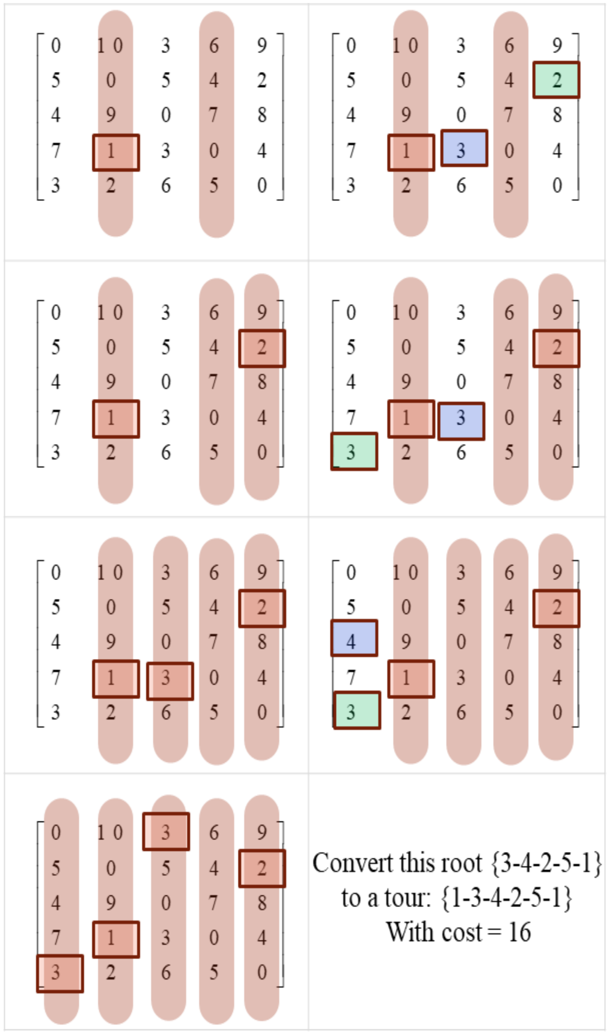


Fig. 1 Solving the instances of [14] by Dhouib-Matrix-TSP1

[14] designs a method named One's Assignment method to solve the TSP. Fig 2. shows that our method Dhouib-Matrix-TSP1 finds the optimal solution in exactly 5 iterations.

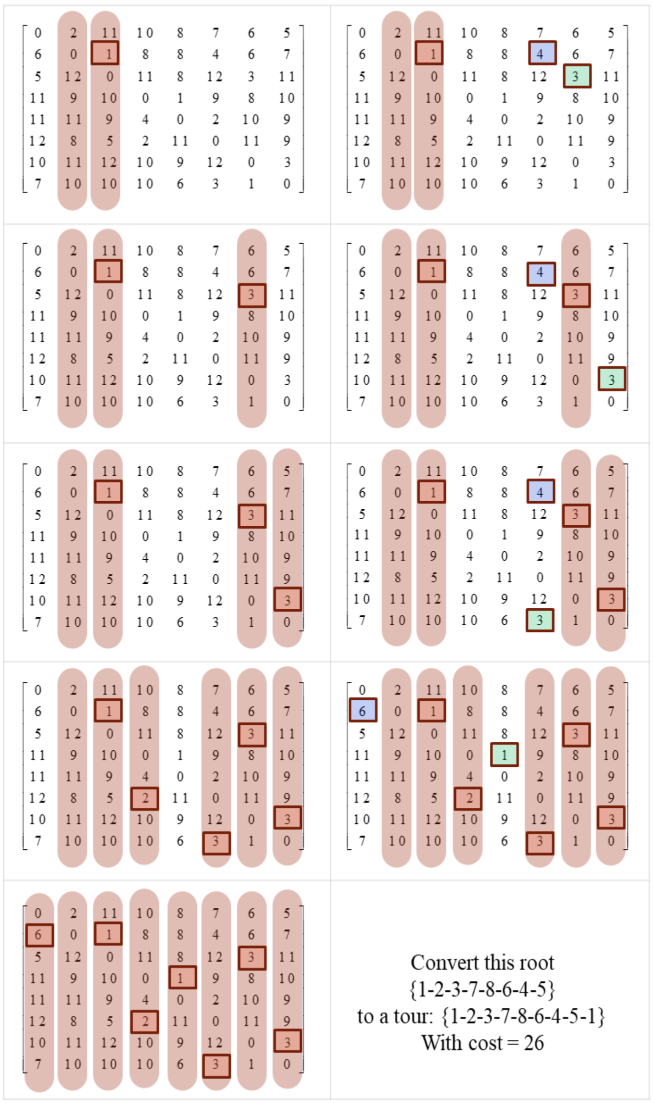


Fig. 2 Solving the instances of [15] by Dhouib-Matrix-TSP1

[15] applies the Hungarian method to find a tour for 8 cities with a distance of 34, whereas our method finds a better solution {1-2-3-7-8-6-4-5-1} with a distance of 26 (improvement of 24%).

5. Conclusion

In this paper, a new column-row method named Dhouib-Matrix-TSP1 and composed of four steps is designed. Then, several numerical examples are depicted, each example is simulated using the Python language, and the solution is compared to the optimum.

It can be concluded that the proposed Dhouib-Matrix-TSP1 method uses a simple technique and generates an optimal or a near-optimal solution in polynomial time (on only *n* iterations). An extension to this work is to apply our method to other combinatory optimization problems.

References

[1] Omar Cheikhrouhoua, and Ines Khoufi, “A Comprehensive Survey on the Multiple Traveling Salesman Problem: Applications, Approaches, and Taxonomy,” *Computer Science Review*, vol. 40, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[2] Mohsen Mosayebi, Manbir Sodhi, and Thomas A. Wettergren, “The Traveling Salesman Problem with Job-times (TSPJ),” *Computers & Operations Research*, vol. 129, 2021.[CrossRef] [Google Scholar] [Publisher Link]

[3] Shijin Wang, Ming Liu, and Feng Chu, “Approximate and Exact Algorithms for an Energy Minimization Traveling Salesman Problem,” *Journal of Cleaner Production*, vol. 249, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[4] Moshe Kaspi, Moshe Zofi, and Ron Teller, “Maximizing the Profit Per Unit Time for the Traveling Salesman Problem,” *Computers & Industrial Engineering*, vol. 135, pp. 702-710, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[5] Juan C. Pina-Pardo, Daniel F. Silva, and Alice E. Smith, “The Traveling Salesman Problem with Release Dates and Drone Resupply,” *Computers & Operations Research*, vol. 129, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] Xin Wei et al., “Multi-core-, Multi-Thread-Based Optimization Algorithm for Large-Scale Traveling Salesman Problem,” *Alexandria Engineering Journal*, vol. 60, no. 1, pp. 189-197, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Cemal Aykut Gencel, and Barış Keçeci, “Traveling Salesman Problem with Hotel Selection: Comparative Study of the Alternative Mathematical Formulations,” *Procedia Manufacturing*, vol. 39, pp. 1699-1708, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[8] Christian Doppstadt, Achim Koberstein, and Daniele Vigo, “The Hybrid Electric Vehicle—Traveling Salesman Problem with time windows,” *European Journal of Operational Research*, vol. 284, no. 2, pp. 675-692, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[9] Kostas Florios, and George Mavrotas, “Generation of the Exact Pareto set in Multi-Objective Traveling Salesman and Set Covering Problems,” *Applied Mathematics and Computation*, vol. 237, pp. 1-19, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[10] Aimin Zhou, Feng Gao, and Guixu Zhang, “A Decomposition-Based Estimation of Distribution Algorithm for Multi-Objective Traveling Salesman Problems,” *Computers & Mathematics with Applications*, vol. 66, no. 10, pp. 1857-1868, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[11] Yannis Marinakis, Magdalene Marinaki, and Georgios Dounias, “Honey Bees Were Mating Optimization Algorithm for the Euclidean Traveling Salesman Problem,” *Information Sciences*, vol. 181, no. 20, pp. 4684-4698, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[12] Briana Couto, “*Using Matrices And Hungarian Method To Solve The Traveling Salesman Problem*,” Mathematics Honors Theses, Ph.D. at Digital Commons at Salem State University, 2018. [Google Scholar] [Publisher Link]

[13] Sevgi Erdoğan, and Elise Miller-Hooks, “A Green Vehicle Routing Problem,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 100-114, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[14] Hadi Basirzadeh, “One's Assignment Method for Solving Traveling Salesman Problem,” *Journal of Mathematics and Computer Science*, vol. 10, pp. 258-265, 2014. [Google Scholar] [Publisher Link]

[15] Janusz Czopik, “An Application of the Hungarian Algorithm to Solve Traveling Salesman Problem,” *American Journal of Computational Mathematics*, vol. 9, pp. 61-67, 2019. [CrossRef] [Google Scholar] [Publisher Link]