Travel Salesman Problem Algorithm Using Proposed Algorithm Development with K-Smallest City Method

Arba Robbani
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Indonesia
arbarobbani@students.itb.ac.id
23220346@mahasiswa.itb.ac.id

Ayu Latifah
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Indonesia
33220010@std.stei.itb.ac.id
ayulatifah@sttgarut.ac.id

Devani Claudia
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Indonesia
devaniclg@gmail.com

Nana Rahmana Syambas
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Indonesia
nana@stei.itb.ac.id

***Abstract*—. The Traveling Salesman Problem (TSP), classified as an NP-hard problem in combinatorial optimization, requires a lot of time to optimize large-scale data. In this paper, the proposed algorithm is designed to produce the smallest value and the fastest time in the computation process to find the shortest path. The results of the proposed algorithm design will be compared with algorithms that are generally widely used to solve TSP, including Ant Colony Optimization (ACO) and Brute Force, but in addition to comparing the results of the proposed algorithm in terms of minimum value and computational speed developed with the two algorithms, it will also be compared to the OR-Tools algorithm created by Google because to be able to evaluate the level of accuracy on the value generated on the large matrix dimension, it cannot be done on Brute Force because it takes a long time. The results show that the proposed algorithm can provide better results than ACO and shorter computation time than Brute Force even can still compete with Google's OR-Tools algorithm for the data used.**

Keywords— traveling salesman problem, optimization, shortest path, brute force, ACO, OR-Tools, k-smallest city

I. INTRODUCTION

Traveling Salesman Problem (TSP) is a challenge to determine the shortest but effective route given in a topology. TSP is known as an algorithmic problem in the field of computer science. TSP belongs to a class of combinatorial optimization problems known as NP-complete. This means that TSP is classified as NP-hard because it does not have a fast solution, and the complexity of calculating the best route will increase with increasing destinations [1]–[4].

Many ways have been done to be able to complete this TSP, including several optimization algorithm developments such as Genetic Algorithm, Simulated Annealing (SA), Gravitational Search Algorithm (GSA), to the current trend using Machine Learning (ML) technology and Artificial Intelligence (AI) which is more intelligent but complex to get the best results that are more optimal from the various ways that have been done[3], [5], [6], [7]. Apart from that, the algorithms that are quite popular for use in TSP are Ant Colony Optimization (ACO) and Brute Force[1], [8], [9].

Algorithms such as Ant Colony Optimization (ACO) and Brute Force are widely used in solving TSP. ACO is used to determine the shortest path and the best cost in the minimum iteration, whereas Brute Force will calculate and compare all possibilities and determine the smallest minimum total cost. However, Brute Force has a computational time limit. The more nodes, the longer the computation time[9].

In this paper, the author develops an algorithm written in the Python programming language. This algorithm works by simulating to find the shortest distance from all the existing distances so that the result is a ring topology that starts from one point to several existing points and then, after all the points have been traversed, will return to the starting point. The results firstly will be compared with the ACO and Brute Force algorithms, with the hope that it can provide better accuracy than ACO but with a much shorter computation time compared to Brute Force.

In general, the Ant Colony Optimization (ACO) and Brute Force algorithms are used to determine the shortest distance and weight with the smallest value they have in the search process[2]. This algorithm is widely adopted in various applications such as the Traveling Salesman Problem (TSP), the quadratic task problem, and so on. These two algorithms are used to solve problems in finding the shortest distance that can be traveled through all existing cities so that they will eventually return to the starting point of the search [4],[12].

Moreover, in this paper, a comparison will also be made with the algorithm created by Google for related problems, namely OR -Tools[1]. OR-Tools is an algorithm that has been developed by the Google company to help solve this TSP. Besides having a good level of accuracy, this algorithm also has a short computation time in finding the best path. Therefore, this algorithm is used as a comparison to the proposed algorithm. Another goal is to be able to evaluate the accuracy and speed of time resulting from the algorithm that has been developed so that it can be used as a reference in developing further research on the results displayed.

II. RELATED WORK

In several previous studies, similar research has been carried out to be able to solve this tsp with the data used in the form of random matrix results obtained from the python code used. The number of dimensions of the matrix is 100.

The Fast Heuristic Algorithm is one of the studies that result in the development of an algorithm with an approach that is carried out by comparing the sum of the node values with two minimum distance values horizontally and vertically. The results of the development of this algorithm provide a solution for the shortest distance that is better than ACO with a shorter computation time, but when compared to brute force, there is still a shortage of the resulting level of accuracy[9].

Furthermore, the development of algorithms to form a ring topology with the least path value. The algorithm developed is designed by approaching it by connecting several nodes with a minimum distance value and then making a tree design from the lowest two minimum paths. The results obtained at nodes 20-50 of the algorithm that has been developed have an average percentage of 15% better than ACO in producing the most minimal solution in solving TSP[8].

Therefore, in this paper, the results of research in developing an algorithm that is more optimal than previous studies will be presented, while the design of the algorithm that has been developed has the aim of being able to produce more optimal time from ACO and an accuracy level that can compete with Brute Force, with evaluation comparing the results of algorithm using Google OR-Tools algorithm so that it is expected to provide ideas for further research opportunities.

III. PROPOSED ALGORITHM

This proposed algorithm is found by doing the experiment with a 5x5 and 6x6 cost matrix. These data are part of data 15x15 in this paper. There are many destination cities, as in figure 1 and figure 2. Determined origin city from A and will return to A again form ring topology.

	A	B	C	D	E
A	0	15	47	10	35
B	15	0	93	25	27
C	47	93	0	55	11
D	10	25	55	0	7
E	35	27	11	7	0

Fig. 1. Data 5x5 matrix

From figure 1, we used a brute force algorithm and obtained the shortest path is A-B-D-E-C-A. We can see the shortest path never choose destination city with the highest cost, but the path will lead to *k*-smallest city. Look at city A, and the path will choose B as the second smallest cost. From B, the path will choose D as the second smallest cost as original and first smallest cost because B can't choose to A before through all cities. From D, a path will choose E as the first smallest cost, and this city is not visited before and so on until path A-B-D-E-C-A is obtained.

	A	B	C	D	E	F
A	0	15	47	10	35	8
B	15	0	93	25	27	32
C	47	93	0	55	11	18
D	10	25	55	0	7	52
E	35	27	11	7	0	20
F	8	32	18	52	20	0

Fig. 2. Data 6x6 matrix

From figure 2, we also used a brute force algorithm and obtained the shortest path is A-F-C-E-D-B-A. With adding one column in matrix 5x5 and matrix is symmetric, it can be said that city F is placed after A-B-D-E-C, solution in 5x5 matrix. This city F is first smallest cost from city A and second from city C. So, this concept still valid for another dimension. However, we do not know where the smallest cost will be chosen whether first, second, three, and so on. We will try to select the smallest k that can be selected with prefix variable and try to connect between city to another city using all possible node to get the best shortest path. However, this will take time too long if the number of k is high, so we must limit the number of nodes that can try with free variable. In this search path, if we only need estimation shortest path, we can use take variable. Setting the prefix, free, and take values will give different computation times and accuracy, so we need to look for them to get optimal results. For more details, read below pseudocode.

Algorithm 1: K-smallest city

```
Data: Matrix of cost of nodes/cities
Result: Shortest route and its distances
Intialization prefix, free, take, routes
// prefix used to find k-smallest in each node
// free is limit of possible next node
// take is number of routes data taken
// routes is list of path with their distances
Function FindPaths(cities, matrix, node, path, distances):
    Add node to path
    if length of path < 1 then
        Calculate distance from one node to destination node
    else
        for i to number of cities do
            Sorting the cost of each node
            Take cost in prefix for each node and save in list first
            Sorting value in first
        end
        for i to number of cities do
            Create list real cost of each node without 0 and lower
              than the biggest of first
            Find node with values from real and save in list citylist
        end
    end
    if all cities in path then
        Add last node
        Calculate last distances
        if nothing data in routes then
            if last distance not zero then
                Print path, distances
                Save in list routes
        else
            if distances lower and not reverse path in routes then
                Print path, distances
                Save in list routes
        end
    end
    Create list sort value that not node and not in path of citylist
    // for limit iteration
    if 3 < length of sort < 10 then
        Take length of sort divide by 2
    else if length of sort > 10 then
        Take length of sort by free
    end
    for i to number of sort do
        city is i-th sort if city not in path then
            FindPaths (cities, matrix, city, path, distances)
        if length of routes = take then
            Print routes as musch as take
    end
```

IV. RESULTS AND DISCUSSIONS

The data used in the proposed algorithm testing is data in the form of a symmetric matrix with dimensions of 15x15, 50x50, and 100x100. This data will be used to validate the results of algorithms made with ACO and Google OR-Tools Algorithm.

The tools or platforms used in implementing and testing the proposed algorithm development that has been designed will use the Google Collaboratory platform, which is a coding environment for the Python programming language with a "notebook" format (similar to Jupyter notebook), or in other words, it's like Google lending us a computer for free. To create programs or perform data processing from Google. Considerations in using this platform are also based on the ease of using the library, collaborating, and practice sharing it with certain parties so to facilitate the discussion and collaboration process in designing and integrating the design results into python coding so that the desired results can be seen as intended. The purpose of the design that has been carried out on the development of the proposed algorithm [10], [11].

Testing was carried out on several related algorithms, namely ACO, Brute Force, and propose an algorithm to see the comparison of the accuracy of the results of the shortest distance displayed and the length of time taken to obtain the shortest distance from the three matrices data used during the testing process. The results for matrix dimensions up to 15x15 can be seen in Figure 3 and Figure 4.

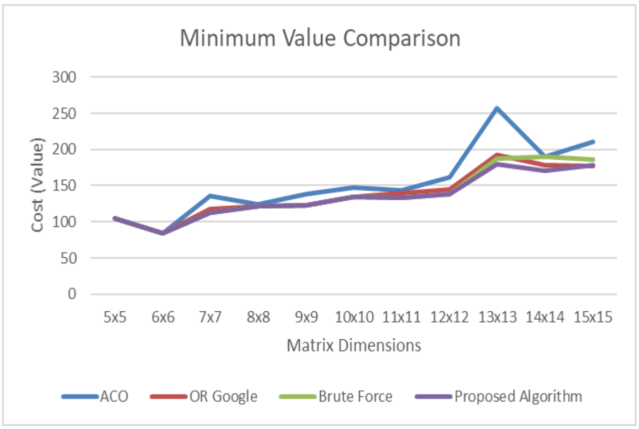

Fig. 3. Comparison minimum value using 15x15 matrix

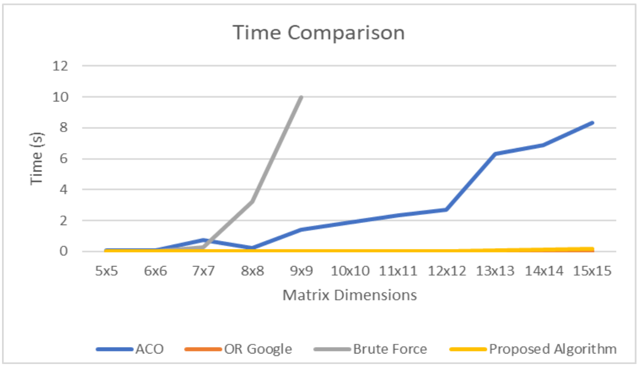

Fig. 4. Comparison time of proposed algorithm using 15x15 matrix

For the first data in the matrix up to 15x15, from the comparison results shown in Figure 3 and Figure 4, it can be seen that the test results of the proposed algorithm are able to produce accuracy and time duration in finding the shortest distance from the matrix data used more optimally than the ACO Algorithm and Brute Force, whereas when compared with OR Google for a matrix with matrix dimensions below 15x15 it can still give better results but when it reaches a 15x15 matrix both in terms of value and time, OR Google is superior. When compared with ACO, the proposed Algorithm 10.62% better in obtaining the shortest path and get 98.49% better in obtaining time. When compared with brute force, the proposed algorithm gets as well as a brute force but in better time. When compared with Google's OR, the proposed Algorithm 2.22% better in obtaining the shortest path but lose in obtaining time. The results for matrix dimensions up to 50x50 can be seen in Figure 5 and Figure 6.

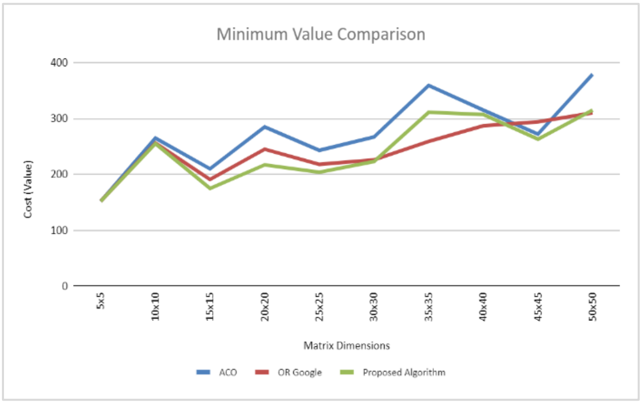

Fig. 5. Comparison minimum value using 50x50 matrix

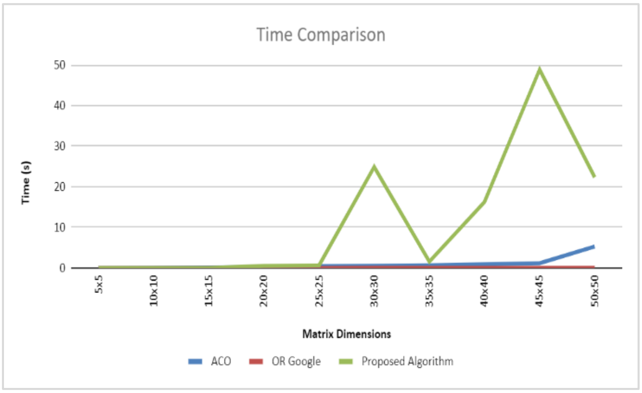

Fig.6. Comparison minimum time using 50x50 matrix

The testing of the second data, namely with matrix dimensions up to 50x50, from the comparison results shown in Figure 7 and Figure 8. The test results of the proposed algorithm that when the matrix dimensions are below 35x35 are able to provide better results than on ACO and Google's OR, but when it reaches a matrix larger than 35x35, the proposed algorithm developed at a certain dimension point cannot compete with ACO or OR-Tool to get the minimum value. When compared with ACO, the proposed Algorithm 11.29% better in obtaining the shortest path, but if used matrix below 35x35 obtained 12.89% better. When compared with Google's OR, the proposed algorithm only 1% better in obtaining the shortest path but in the 30x30 matrix, it obtained 4.66% better. Even in the time comparison, this proposed algorithm has a larger duration compared to the two algorithms. The comparison the results for matrix dimensions up to 100x100 can be seen in Figure 7 and Figure 8.

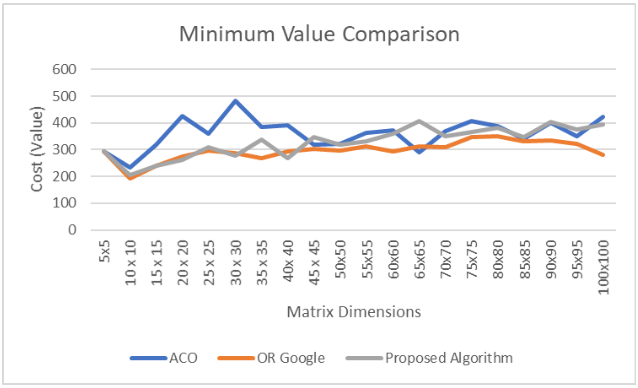

Fig. 7. Comparison minimum value using 100x100 matrix

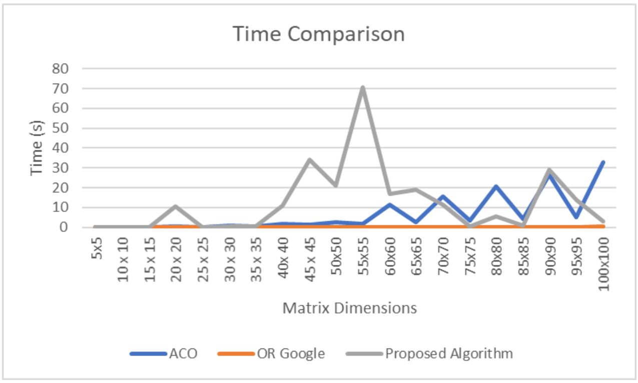

Fig. 8. Comparison minimum time using 100x100 matrix

In the last test data with a matrix dimension of 100x100. From the comparison results shown in Figure 7 and Figure 8, it can be seen that the test results of the proposed algorithm that when the matrix dimensions are below 50x50 are able to provide results that can compete even on some dimensions can provide results that better than Google's ACO and OR for comparison in achieving the minimum value, but when it reaches a matrix larger than 50x50 the proposed algorithm developed at a certain dimension point cannot compete with ACO or OR-Tool. Even the time part has a more extensive duration compared to the two comparison algorithms in the matrix dimensions below 70x70, although at some points, specific dimensions can still compete with the ACO algorithm. When compared with ACO, the proposed Algorithm 7.70% better in obtaining the shortest path, but if used matrix below 35x35 obtained 20.65%, below 40x40 obtained 22.03% better, and below 50x50 obtained 16.94%. When compared with OR's Google, proposed Algorithm 10.03% worse in getting the shortest path, but if used matrix below 35x35 got 4.03% worse.

In this testing, we know that this proposed algorithm has good accuracy to get the shortest path 100x100 dimension. This is basically caused by the process of searching for the shortest distance from the existing data where distances with a higher value than other distances will be passed automatically in the process of searching for the closest distance from existing data and only take the possible k-smallest city. This will certainly shorten the time in searching—the shortest distance without ignoring the accuracy of the search results that have been carried out.

From the results obtained, it is also seen that the larger the dimensions of the data matrix used, the longer the time needed to find the shortest distance, this is because the process of repetition in the search for distances as much as the number possible of k-smallest city and complexity of the type of cost matrix. An example of this complexity is matrix has centered city that has the shortest path, so we challenging to get number of k in k-smallest city.

From the results of the tests that have been carried out, several related algorithms have been compared, which are generally used to find the shortest distance from two data matrices. The algorithms used include Ant Colony Optimization (ACO), Brute Force, OR Google and the proposed algorithm development according to the concept that has been developed, but in testing up to 50x50 and100x100 brute force matrix, it cannot be used as a comparison again because the computation process is too long.

V. CONCLUSION

One of the most popular combinatorial problems is TSP. Until now, there are various kinds of new techniques proposed to solve it. In this paper, a new method is proposed by designing a proposed algorithm development to be able to provide solutions in solving TSP. The development of the proposed algorithm is a design result to be able to optimize functions that are superior to previous research by developing an algorithm based on finding the shortest distance that can be traversed in passing all existing paths from the starting point with a ring topology scheme after all existing paths have been passed. With matrix below 35x35, this proposed algorithm on average 16.77% better in shortest path if compared with ACO. With matrix below 50x50, this proposed algorithm on average 14.12% better in shortest path if compared with ACO. With matrix below 100x100, this proposed algorithm on average 7.70% better in shortest path if compared with ACO. This proposed, if testing with below 100x100, can still compete results compared by OR's Google.

The results show that the greater the dimensions of the matrix data used, the longer the duration of time needed to find the shortest distance. This is because the repetition process in the search to get shortest distances and filtering process. This duration will be influenced by setting of *prefix, free,* and *take* variable.

The following are suggestions for the development of algorithms that can be carried out in further research. It is necessary to find an algorithm concept for better further development so that the search results for the shortest distance for matrices with more dimensions do not have a significantly longer time duration. In this proposed algorithm, still need learning algorithm to estimate the shortest path without using repetition searching. Author has insight in this problem but make more time to finish it. Using this k-smallest city method, we can give rank in each city/node from other node from *citylist* data in this paper and be calculated mean of these rank. With below 15x15 matrix, these data can estimate the shortest path and less time consuming but with more dimensions, this insight cannot be produced optimal path although with less time consuming. This is because need more data to estimate shortest path. With the complexity in solving scheduling and routing problems, it will continue to increase to continue to carry out studies and continuous research in the future on increasingly challenging CSR issues.

REFERENCES

[1] I. J. Rahman, A. R. Zain, and N. R. Syambas, "A new heuristic method for optical network topology optimization," *Proc. - ICWT 2016 2nd Int. Conf. Wirel. Telemat. 2016*, pp. 73–77, 2017, doi: 10.1109/ICWT.2016.7870855.

[2] S. Ebadinezhad, "DEACO: Adopting dynamic evaporation strategy to enhance ACO algorithm for the traveling salesman problem," *Eng. Appl. Artif. Intell.*, vol. 92, no. November 2019, p. 103649, 2020, doi: 10.1016/j.engappai.2020.103649.

[3] R. Jain, M. L. Meena, and K. P. Singh, "Travelling Salesman Problem Optimization Using Hybrid Genetic Algorithm," *Lect. Notes Networks Syst.*, vol. 175, pp. 350–356, 2021, doi: 10.1007/978-3-030-67187-7_36.

[4] M. A. H. Akhand, S. I. Ayon, S. A. Shahriyar, N. Siddique, and H. Adeli, "Discrete Spider Monkey Optimization for Travelling Salesman Problem," *Appl. Soft Comput. J.*, vol. 86, p. 105887, 2020, doi: 10.1016/j.asoc.2019.105887.

[5] A. H. Halim and I. Ismail, "Combinatorial Optimization: Comparison of Heuristic Algorithms in Travelling Salesman Problem," *Arch. Comput. Methods Eng.*, vol. 26, no. 2, pp. 367–380, 2019, doi: 10.1007/s11831-017-9247-y.

[6] P. Kerschke, L. Kotthoff, J. Bossek, H. H. Hoos, and H. Trautmann, "Leveraging TSP solver complementarity through machine learning," *Evol. Comput.*, vol. 26, no. 4, pp. 597–620, Dec. 2018, doi: 10.1162/EVCO_a_00215.

[7] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L. M. Rousseau, "Learning heuristics for the tsp by policy gradient," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Jun. 2018, vol. 10848 LNCS, pp. 170–181, doi: 10.1007/978-3-319-93031-2_12.

[8] B. Supriadi, B. Budi, and N. R. Syambas, "Comparison of proposed algorithm to ant colony optimization algorithm to form a ring topology with minimum path value," *Proc. - ICWT*

2016 2nd Int. Conf. Wirel. Telemat. 2016, pp. 67–72, 2017, doi: 10.1109/ICWT.2016.7870854.

[9] Christian, W. M. Ashari, and N. R. Syambas, "Proposed new heuristic algorithm for travelling salesman problem based from minimum link," *Proc. - ICWT 2016 2nd Int. Conf. Wirel. Telemat. 2016*, pp. 78–83, 2017, doi: 10.1109/ICWT.2016.7870856.

[10] T. Carneiro, R. V. M. Da Nobrega, T. Nepomuceno, G. Bin Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications," *IEEE Access*, vol. 6, pp. 61677–61685, 2018, doi: 10.1109/ACCESS.2018.2874767.

[11] E. Bisong, "Google Colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, Apress, 2019, pp. 59–64.

[12] E. Es Yurek and H. C. Ozmutlu, "A decomposition-based iterative optimization algorithm for traveling salesman problem with drone," *Transp. Res. Part C Emerg. Technol.*, vol. 91, no. July 2017, pp. 249–262, 2018, doi: 10.1016/j.trc.2018.04.009.