(IJID) International Journal on Informatics for Development, *e-ISSN: 2549-7448*
*Vol. 7, No. 2, 2018, Pp. 54-58*

# Comparison of Dijkstra dan Floyd-Warshall Algorithm to Determine the Best Route of Train

Tri Setya Darmawan
Student of Department of Informatics, Graduate Program, Faculty of Science and Technology
State Islamic University Sunan Kalijaga Yogyakarta
tsdarmawan@gmail.com

***Abstract*-- This study aims to find out the comparison of Dijkstra and Floyd-Warshall algorithms in finding the best path on a train trip. The best route is the path, which has the minimum price of a train journey. The results of route discovery will be displayed in a web-based application using the PHP programming language and MySQL database. The results of these two algorithms are compared using four parameters: time complexity, memory complexity, level of completion and level of optimization. Based on our experiments, Dijkstra algorithm has better performances on those four parameters than Floyd-Warshall algorithm.**

***Keywords-Train; Searching for the Best Route; Dijkstra and Floyd-Warshall Algorithm.***

## I. INTRODUCTION

In a journey, it typically needs solution to find alternative routes or path best. Alternative routes could be the fastest, the shortest line, etc. The experimental work in this research will search the best route on transportation in the form of a train. The best path has low cost of price ticket of train. To find the best path, we use shortest path algorithms.

The shortest path algorithm is an algorithm used to locate the shortest. There are several types of shortest path algorithms, such as Dijkstra algorithm, Floyd-Warshall algorithm, Greedy algorithm, Bellman-Ford algorithm, etc. In this research, we compare Dijkstra algorithm and Floyd-Warshall algorithm. The goal of this comparison is to obtain performances of those algorithms in the four parameters: time complexity, memory complexity, completeness, and optimality.

Dijkstra algorithm is used to find the shortest path from the office of PT. Telkom Indonesi Regional IV Jateng-DIY to offices of PT. Telkom Indonesia in other regions to ease the mobility of staffs [1]. This algorithm is also implemented to provide the information of the estimation of fuel consumption and time. The system is a web-based system.

## II. SHORTEST PATH ALGORITHM

### A. Dijkstra Algorithm

Dijkstra algorithm found by Edsger Dijkstra in 1959. This algorithm is one algorithm to solve problems in the search for the shortest path from a source to a destination [2]. In addition, this algorithm is also a form of Greedy algorithm. In solving the problem, Dijkstra algorithm only resolves the problem for a path whose weight is not negative value. Dijkstra algorithm has a complexity time of $O(V*\log V+E)$ where $V$ is a vertex and $E$ is edges. The time complexity of this algorithm is $O(n^2)$. From these two values, it can be concluded that the total computing asymptotic time of Dijkstra algorithm is $T(n)=n$.

According to Siang [3] Dijkstra algorithm is as follows:

1 $L = \{\}$;
$V = \{v_2, v_3, \ldots, vn\}$.
2 While I = 2, ..., n, do D(i) = W(1,i)
3 For $v_n \notin L$ do:
a. Choose point $vk \in V-L$ with minimum D(k)
$L = L \cup \{v_k\}$
b. For each $vj \in V-L$ do:
If $D(j)>D(k)+W(k,j)$ then change D(j) with $D(k)+W(k,j)$
4 For each $vj \in V$, $w*(1,j) = D(j)$

Declarations:
$V(G)$ = $\{v_1, v_2, \ldots, v_n\}$.
$L$ = set of dots $\in V(G)$ which have been selected in the shortest path
$D(j)$ = amount of minimum path weight from $v_1$ to $v_j$
$w(i,j)$ = path weight from $v_i$ to $v_j$
$w*(1,j)$ = amount of minimum path weight from $v_1$ to $v_j$

Initial node is the node to start the searching. By using this algorithm, initial distance values will be improved step by step [4]. In [4], the classic Dijkstra's algorithm was designed to solve the single source shortest problem for a static graph.

### B. Floyd-Warshall Algorithm

Robert Floyd found Floyd-Warshall algorithm in 1962. This algorithm is one of algorithms to solve problems in the search of shortest path algorithm in addition to Dijkstra algorithm. According to Kriswanto [5], Floyd-Warshall algorithm is one variant of dynamic programming. It means that it solves the problems by looking at solutions and finding from that a decision that is mutually bound. Thus, these solutions are formed from the discovery of solutions in the previous stages and enables more than one solution is found.

For the implementation, Floyd-Warshall algorithm starts the iteration from first point. Then adding the track or path by evaluating all points to the destination point, which have minimum amount of weight.

For the example, $W_0$ is first adjacency matrix of a directed graph. $W*$ is minimum matrix adjacency with $Wij*$ is shortest path from $v_i$ to $v_j$.

Floyd-Warshall algorithm, which is referred in [5], is as follows:

1 $W = W_0$
2 For k = 1 to n, do:
For i = 1 to n, do:
For j = 1 to n, do:
If $W[i,j]>W[i,k]+W[k,j]$
Then replace $W[i,j]$ with $W[i,k]+W[k,j]$
3 $W* = W$

Declarations:
$W$ = matrix
$W_0$ = first adjacency matrix graph
$K$ = iteration 1 to n
$i$ = first point $v_i$
$j$ = first point $v_j$
$W*$ = result of matrix after comparison

For the search of shortest path, all iterations ($k$) will be used to make $n$ matrix. Therefore, the process used is slower than Dijkstra algorithm. Especially if value of $n$ is big. However, Floyd-Warshall algorithm is not rarely used to solve the problem in finding shortest path.

## III. RESULT

Before the implementation of two algorithmss, firstly all of data used are collected. The data used are train's name,

station's name, map of train, and schedule of train. These data are shown in Table 1-2, and Fig. 1-4.

TABLE I. TRAIN'S NAME AND GENRE

| No | Train's Name | Genre |
|---|---|---|
| 1 | Argo Bromo Anggrek | Executive |
| 2 | Argo Wilis | Executive |
| 3 | Argo Lawu | Executive |
| 4 | Argo Dwipangga | Executive |
| 5 | Argo Sindoro | Executive |
| 6 | Argo Muria | Executive |
| 7 | Argo Jati | Executive |
| 8 | Argo Parahyangan | Executive |
| 9 | Gajayana | Executive |
| 10 | Bima | Executive |
| 11 | Sembrani | Executive |
| 12 | Turangga | Executive |
| 13 | Taksaka | Executive |
| 14 | Bangunkarta | Executive |
| 15 | Purwojaya | Executive |
| 16 | Tegal Bahari | Executive |
| 17 | Cirebon Ekspres | Executive |
| 18 | Harina | Executive |
| 19 | Gumarang | Executive |
| 20 | Lodaya | Executive |
| 21 | Sancaka | Executive |
| 22 | Mutiara Timur Siang | Executive |
| 23 | Mutiara Timur Malam | Executive |
| 24 | Malabar | Executive |
| 25 | Malioboro Ekspres | Executive |
| 26 | Ciremai | Executive |
| 27 | Ranggajati | Executive |
| 28 | Mutiara Selatan | Executive |
| 29 | Sawunggalih | Executive |
| 30 | Bogowonto | Executive |
| 31 | Kamandaka | Executive |
| 32 | Bengawan | Economy |
| 33 | Bogowonto | Economy |
| 34 | Brantas | Economy |
| 35 | Gajahwong | Economy |
| 36 | Gaya Baru Malam Selatan | Economy |
| 37 | Jaka Tingkir | Economy |
| 38 | Jayabaya | Economy |
| 39 | Jayakarta Premium | Economy |
| 40 | Kahuripan | Economy |
| 41 | Kamandaka | Economy |
| 42 | Kertajaya | Economy |
| 43 | Kutojaya Utara | Economy |
| 44 | Logawa | Economy |
| 45 | Majapahit | Economy |
| 46 | Matarmaja | Economy |
| 47 | Menoreh | Economy |
| 48 | Pasundan | Economy |
| 49 | Probowangi | Economy |
| 50 | Progo | Economy |
| 51 | Serayu | Economy |
| 52 | Singasari | Economy |
| 53 | Sri Tanjung | Economy |
| 54 | Tawang Jaya | Economy |
| 55 | Tegal Ekspres | Economy |

TABLE II. STATION'S NAME

| No | Station's Name | No | Station's Name |
|---|---|---|---|
| 1 | Gambir | 8 | Surabaya Gubeng |
| 2 | Bandung | 9 | Jember |
| 3 | Cirebon | 10 | Pasar Senen |
| 4 | Purwokerto | 11 | Kiaracondong |
| 5 | Semarang Tawang | 12 | Cirebon Prujakan |
| 6 | Yogyakarta | 13 | Semarang Poncol |
| 7 | Madiun | 14 | Lempuyangan |

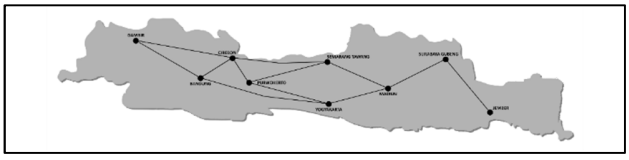

Figure 1. Map of train's track executive genre

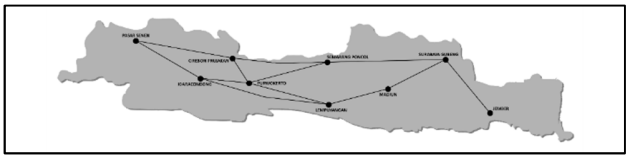

Figure 2. Map of train's track economy genre

| No KA | Nama KA | Jenis KA | Stasiun Awal | Stasiun Tujuan | Waktu Awal | Waktu Tiba | Waktu Tempuh | Harga |
|---|---|---|---|---|---|---|---|---|
| 6 | Argo Wilis | Eksekutif | Bandung | Yogyakarta | 8:30 | 15:52 | 7:22 | 370 |
| 6 | Argo Wilis | Eksekutif | Bandung | Yogyakarta | 8:30 | 15:52 | 7:22 | 340 |
| 6 | Argo Wilis | Eksekutif | Bandung | Yogyakarta | 8:30 | 15:52 | 7:22 | 310 |
| 6 | Argo Wilis | Eksekutif | Bandung | Yogyakarta | 8:30 | 15:52 | 7:22 | 280 |
| 19 | Argo Parahyangan | Eksekutif | Bandung | Gambir | 5:00 | 8:15 | 3:15 | 120 |
| 19 | Argo Parahyangan | Eksekutif | Bandung | Gambir | 5:00 | 8:15 | 3:15 | 115 |
| 19 | Argo Parahyangan | Eksekutif | Bandung | Gambir | 5:00 | 8:15 | 3:15 | 105 |

Figure 3. Schedule of train executive genre

| No KA | Nama KA | Jenis KA | Stasiun Awal | Stasiun Tujuan | Waktu Awal | Waktu Tiba | Waktu Tempuh | Harga |
|---|---|---|---|---|---|---|---|---|
| 147 | Jaka Tingkir | Ekonomi | Cirebon Prujakan | Pasar Senen | 0:26 | 3:30 | 3:04 | 175 |
| 147 | Jaka Tingkir | Ekonomi | Cirebon Prujakan | Pasar Senen | 0:26 | 3:30 | 3:04 | 165 |
| 147 | Jaka Tingkir | Ekonomi | Cirebon Prujakan | Pasar Senen | 0:26 | 3:30 | 3:04 | 130 |
| 175 | Brantas | Ekonomi | Cirebon Prujakan | Pasar Senen | 0:50 | 3:54 | 3:04 | 84 |
| 177 | Kertajaya | Ekonomi | Cirebon Prujakan | Pasar Senen | 5:30 | 8:40 | 3:10 | 100 |
| 177 | Kertajaya | Ekonomi | Cirebon Prujakan | Pasar Senen | 5:30 | 8:40 | 3:10 | 90 |
| 171 | Matarmaja | Ekonomi | Cirebon Prujakan | Pasar Senen | 6:10 | 9:20 | 3:10 | 109 |
| 141 | Majapahit | Ekonomi | Cirebon Prujakan | Pasar Senen | 7:12 | 10:08 | 2:56 | 160 |
| 141 | Majapahit | Ekonomi | Cirebon Prujakan | Pasar Senen | 7:12 | 10:08 | 2:56 | 135 |

Figure 4. Schedule of train economy genre

After the data are collected then implementation of two algorithms take place. Results are compared to one another. There are four parameters used in comparison such as time complexity, memory complexity, completeness, and optimal. The implementation of two algorithms described in the steps of those algorithms.

### *A. Dijkstra Algorithm*

The steps of Dijkstra algorithm are as follows:

1) Input data of station's departure and station's destination.
2) Station's departure is at choice point.
3) For the iteration, *n*, system looks for possible routes to be passed based on existing choice points and their closest relationships.
4) System finds possible routes with the lowest value.
5) If the route is obtained by a station, which is not a choice point, then the station is added to the choice point.
6) If the station is station's destination then process is break. In addition, if the station is not the same with station's destination then the process will repeat from point c.
7) A route has been found.

### *B. Floyd-Warshall Algorithm*

The steps to implement Floyd-Warshall algorithm is as follows:

1) Make the table with a size adjusting to the total number of stations in our database; rows are station's departure and columns are station's destination as seen in Fig. 5.
2) Input value by calling data from database as shown in Figure 5. If the row and the column have the same name then value of input is zero. If name of row and column is not same and there is not the value in *database* then value of input is *infinity*.

| d,k | BAN | CIR | GAM | JEM | MAD | PUR | SEM | SUR | YOG |
|---|---|---|---|---|---|---|---|---|---|
| BAN | 0 | 120 | 100 | INF | INF | INF | INF | INF | 230 |
| CIR | 120 | 0 | 120 | INF | INF | 145 | 120 | INF | INF |
| GAM | 100 | 120 | 0 | INF | INF | INF | INF | INF | INF |
| JEM | INF | INF | INF | 0 | INF | INF | INF | 120 | INF |
| MAD | INF | INF | INF | INF | 0 | INF | 285 | 125 | 125 |
| PUR | INF | 145 | INF | INF | INF | 0 | 140 | INF | 195 |
| SEM | INF | 120 | INF | INF | 285 | 140 | 0 | INF | INF |
| SUR | INF | INF | INF | 120 | 125 | INF | INF | 0 | INF |
| YOG | 230 | INF | INF | INF | 125 | 130 | INF | INF | 0 |

Figure 5 First table of Floyd-Warshall algorithm

3) Implementation formula of Floyd-Warshall algorithm.
4) Get the result from formula of Floyd-Warshall algorithm as shown in Figure 6.

| d,k | BAN | CIR | GAM | JEM | MAD | PUR | SEM | SUR | YOG |
|---|---|---|---|---|---|---|---|---|---|
| BAN | 0 | 120 | 100 | 600 | 355 | 265 | 240 | 480 | 230 |
| CIR | 120 | 0 | 120 | 650 | 405 | 145 | 120 | 530 | 340 |
| GAM | 100 | 120 | 0 | 700 | 455 | 265 | 240 | 580 | 330 |
| JEM | 600 | 645 | 700 | 0 | 245 | 500 | 530 | 120 | 370 |
| MAD | 355 | 400 | 455 | 245 | 0 | 255 | 285 | 125 | 125 |
| PUR | 265 | 145 | 265 | 565 | 320 | 0 | 140 | 445 | 195 |
| SEM | 240 | 120 | 240 | 530 | 285 | 140 | 0 | 410 | 335 |
| SUR | 480 | 525 | 580 | 120 | 125 | 380 | 410 | 0 | 250 |
| YOG | 230 | 275 | 330 | 370 | 125 | 130 | 270 | 250 | 0 |

Figure 6 Result Table of Floyd-Warshall Algorithm

5) Search the route from the table of result Floyd-Warshall algorithm as shown in Figure 7.

| d,k | BAN | CIR | GAM | JEM | MAD | PUR | SEM | SUR | YOG |
|---|---|---|---|---|---|---|---|---|---|
| BAN | RUTE TIDAK DITEMUKAN | bandung-cirebon | bandung-gambir | bandung-yogyakarta-madiun-surabaya-gubeng-jember | bandung-yogyakarta-madiun | bandung-cirebon-purwokerto | bandung-cirebon-semarang-tawang | bandung-yogyakarta-madiun-surabaya-gubeng | bandung-yogyakarta |
| CIR | cirebon-bandung | RUTE TIDAK DITEMUKAN | cirebon-gambir | cirebon-semarang-tawang-madiun-surabaya-gubeng-jember | cirebon-semarang-tawang-madiun | cirebon-purwokerto | cirebon-semarang-tawang | cirebon-semarang-tawang-madiun-surabaya-gubeng | cirebon-purwokerto-yogyakarta |
| GAM | gambir-bandung | gambir-cirebon | RUTE TIDAK DITEMUKAN | gambir-bandung-yogyakarta-madiun-surabaya-gubeng-jember | gambir-bandung-yogyakarta-madiun | gambir-cirebon-purwokerto | gambir-cirebon-semarang-tawang | gambir-bandung-yogyakarta-madiun-surabaya-gubeng | gambir-bandung-yogyakarta |
| JEM | jember-surabaya-gubeng-madiun-yogyakarta-bandung | jember-surabaya-gubeng-madiun-yogyakarta-purwokerto-cirebon | jember-surabaya-gubeng-madiun-yogyakarta-bandung-gambir | RUTE TIDAK DITEMUKAN | jember-surabaya-gubeng-madiun | jember-surabaya-gubeng-madiun-yogyakarta-purwokerto | jember-surabaya-gubeng-madiun-semarang-tawang | jember-surabaya-gubeng | jember-surabaya-gubeng-madiun-yogyakarta |
| MAD | madiun-yogyakarta-bandung | madiun-yogyakarta-purwokerto-cirebon | madiun-yogyakarta-bandung-gambir | madiun-surabaya-gubeng-jember | RUTE TIDAK DITEMUKAN | madiun-yogyakarta-purwokerto | madiun-semarang-tawang | madiun-surabaya-gubeng | madiun-yogyakarta |
| PUR | purwokerto-cirebon-bandung | purwokerto-cirebon | purwokerto-cirebon-gambir | purwokerto-yogyakarta-madiun-surabaya-gubeng-jember | purwokerto-yogyakarta-madiun | RUTE TIDAK DITEMUKAN | purwokerto-semarang-tawang | purwokerto-yogyakarta-madiun-surabaya-gubeng | purwokerto-yogyakarta |
| SEM | semarang-tawang-cirebon-bandung | semarang-tawang-cirebon | semarang-tawang-cirebon-gambir | semarang-tawang-madiun-surabaya-gubeng-jember | semarang-tawang-madiun | semarang-tawang-purwokerto | RUTE TIDAK DITEMUKAN | semarang-tawang-madiun-surabaya-gubeng | semarang-tawang-purwokerto-yogyakarta |
| SUR | surabaya-gubeng-madiun-yogyakarta-bandung | surabaya-gubeng-madiun-yogyakarta-purwokerto-cirebon | surabaya-gubeng-madiun-yogyakarta-bandung-gambir | surabaya-gubeng-jember | surabaya-gubeng-madiun | surabaya-gubeng-madiun-yogyakarta-purwokerto | surabaya-gubeng-madiun-semarang-tawang | RUTE TIDAK DITEMUKAN | surabaya-gubeng-madiun-yogyakarta |
| YOG | yogyakarta-bandung | yogyakarta-purwokerto-cirebon | yogyakarta-bandung-gambir | yogyakarta-madiun-surabaya-gubeng-jember | yogyakarta-madiun | yogyakarta-purwokerto | yogyakarta-purwokerto-semarang-tawang | yogyakarta-madiun-surabaya-gubeng | RUTE TIDAK DITEMUKAN |

Figure 7 Result Table of Search the Route

6) Input data of station's departure and station's destination.

7) Search the route in result table of search the route to find the route from station's departure to station's destination.
8) Route has been found.

After the implementation of those algorithms then we compare four parameters in those algorithms. The result of comparison is as follows:

1) Time Complexity
   For the time complexity of two algorithms, Dijkstra algorithm's time complexity is 81 and it is better around 88.89% than Floyd-Warshall algorithm, which has 729.
2) Memory Complexity
   For the memory complexity, Dijkstra algorithm uses 512616 *bytes* or less 46.04% than Floyd-Warshall algorithm, which uses 949968 *bytes* for the genre of executive. For the genre of economy, Dijkstra algorithm uses 482488 *bytes* or less 48.81% than Floyd-Warshall algorithm, which uses 942632 *bytes*. It is because the process of Dijkstra algorithm does not always use all data in *database*, whereas the process of Floyd-Warshall Algorithm uses all of data in *database*.
3) Completeness
   For the completeness, both algorithms have no error.
4) Optimality
   For the optimal between Dijkstra algorithm and Floyd-Warshall algorithm, both algorithms have advantages in implementation. The implementation of Dijkstra algorithm uses dynamic data for every iteration. Meanwhile, the implementation of Floyd-Warshall algorithm uses static data.

## IV. CONCLUSION

After doing research then we can conclude that Dijkstra algorithm has more advantages than Floyd-Warshall algorithm. These advantages are in less usage of time and memory.

## REFERENCES

[1] Andiany, F.E, and Hadikurniawati, W., "Implementasi Algoritma Dijkstra untuk Mencari Rute Terpendek Antar Kantor dan Estimasi Penggunaan Bahan Bakar Kendaraan (Studi kasusu PT. Telkom Indonesia Regional IV Jateng-DIY", *Proceeding of SENDI_U*, 2018.
[2] Javaid, M.A., "Understanding Dijkstra's Algorithm," *SSRN Electronic Journal*, 2013.
[3] Siang, J. J., *Matematika Diskrit dan Aplikasinya pada Ilmu Komputer.* 4nd ed. Yogyakarta: ANDI, 2009.
[4] Alam, Md. A., Faruq, Md. O., "Finding shortest path for road network using Dijkstra's algorithm," *Bangladesh Journal of Multidisciplinary Scientific Research*, vol. I, no. 2, 2019
[5] Kriswanto, dkk., "Penentuan jarak terpendek rute transmusi dengan algoritma Floyd-Warshall," *Proc. of Seminar Nasional Teknologi Informasi & Komunikasi Terapan (SEMANTIK)*, Yogyakarta, 2014.