2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)

Dijkstra's and A-Star in Finding the Shortest Path: a Tutorial

Ade Candra
Department of Computer Science
Faculty of Computer Science and Information Technology
Universitas Sumatra Utara
Medan, Indonesia
ade_candra@usu.ac.id

Mohammad Andri Budiman
Department of Computer Science
Faculty of Computer Science and Information Technology
Universitas Sumatra Utara
Medan, Indonesia
mandrib@usu.ac.id

Kevin Hartanto
Department of Computer Science
Faculty of Computer Science and Information Technology
Universitas Sumatra Utara
Medan, Indonesia
hartantokevin90@gmail.com

***Abstract*—As one form of the greedy algorithm, Dijkstra's can handle the shortest path search with optimum result in longer search time. Dijkstra's is contrary to A-Star, a best-first search algorithm, which can handle the shortest path search with a faster time but not always optimum. By looking at the advantages and disadvantages of Dijkstra's and A-Star, this tutorial discusses the implementation of the two algorithms in finding the shortest path in routes selection between 24 SPBU (gas stations). The routes are located in Medan City and represented in a directed graph. Moreover, the authors compare Dijkstra's and A-star based on the complexity of Big-Theta (Θ) and running time. The results show that the shortest path search between SPBU can be solved with Dijkstra's and A-Star, where in some cases, the routes produced by the two algorithms are different so that the total distance generated is also different. In this case, the running time of A-Star is proven to be faster than Dijkstra's, and it is following A-Star principle which selects the location point based on the best heuristic value while Dijkstra's does not. For the complexity, Dijkstra's is $\Theta(n^2)$ and A-Star is $\Theta(m*n)$, where $0 \leq m \leq n$.**

Keywords—A-Star, Dijkstra's, Big-Theta, running time, SPBU

I. INTRODUCTION

The shortest path problem is the problem of finding a path that will be passed from one point to another with minimum or optimal value. In graph theory, object of the problem is represented as a node (vertice) and the relationship between the objects is represented by a line (edge) that connecting the nodes [1]. The shortest path search has been applied in various fields to optimize the performance of a system, either to minimize the costs or to speed up the process. In this tutorial, the authors discuss the implementation of Dijkstra's and A* (A-Star) algorithms in generating the optimal solution in the route's selection between 24 SPBU (gas stations). Furthermore, the authors compare the performance of both algorithms based on Big-Theta (Θ) and running time. The routes are represented as a weighted graph that has a value or weight and all weights are assumed to be positive. In this case study, the edge-weighted graph represents the distance between the locations of SPBU in Medan City.

Dijkstra's is one of the most popular shortest path algorithms formulated by computer scientist Edsger W. Dijkstra's in 1956 and published in 1959. Dijkstra's algorithm solves the single-source shortest path problem for a graph with non-negative edge path costs [2]. If the graph contains negative edges, the path cannot be passed and may give incorrect results [3]. This algorithm chooses the closest point which is the smallest weight [4], and these weights are the value of each edge [5]. As a form of the greedy algorithm, Dijkstra's processes each node on the graph once, and the number of nodes determines the speed of finding a solution. Dijkstra's can handle the shortest path with optimum result, but the time spent searching for the shortest path is longer.

On the other hand, A* algorithm was first described by Peter Hart, Nils Nilsson, and Bertram Raphael in 1968 with a heuristic method [6]. A* algorithm is one of the many search algorithms that take input, evaluates some possible paths, and return the solution. A* is widely used in pathfinding and graph transversal, the process of plotting an efficiently traversable path between points [7]. A* is a best-first search algorithm by modifying the heuristic function. This algorithm minimizes the total cost of the path, and in the right conditions will provide the best solution in optimal time. A * can choose the shortest path in a faster time, but it is not always optimum.

By looking at the advantages and disadvantages of Dijkstra's and A*, the authors apply both algorithms and analyze the performance in finding the shortest path based on Big-Theta (Θ) and running time.

II. METHODS

This section will discuss how Dijkstra's and A* solve the shortest path problem and define their performance.

A. Dijkstra's

Dijkstra's works by assigning some initial distance values and trying to improve them step by step, as explained below.

1) Assign to every node a tentative distance value: set it to zero for the initial node and to infinity for all other nodes (undefined).
2) Mark all nodes "unvisited" and initial node as "start."
3) Consider all of its unvisited neighbors and calculate their tentative distances through the "start" node. Compare the newly calculated tentative distance to the "start" node value and assign the smaller one.
4) When all neighbors already visited from the "start" node, mark these nodes as "visited." A "visited" node will never be rechecked and last-saved tentative distance is the smallest value.

5) Set the "unvisited" node that is marked with the smallest tentative distance as the new "start" and go back to step 3.

For a better understanding, the authors explain how Dijkstra's finds the shortest path from the example. As shown in Fig. 1, initial node = A and goal node = E.

978-1-7281-9792-0/20/$31.00 ©2020 IEEE

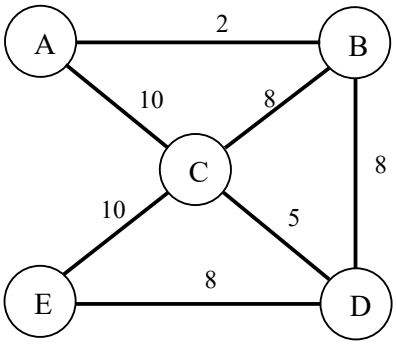

Fig. 1. Graph for Dijkstra's Algorithm

Step 1:

- Create a table for result distance, with i as iteration, u as unvisited node, v as visited node, c as current node, as shown in Table 1.
- i = 0 is initial condition.
- At i = 1; AB = 2, AC = 10. AB is the smallest tentative distance with cost = 2.
- Then, set B as "start" node for i = 2.

TABLE I. ITERATION 0 DAN 1

i	u	c	A	B	C	D	E
0	ABCDE	-	0,-	∞,-	∞,-	∞,-	∞,-
1	BCDE	A	0,-	2,A	10,A	∞,-	∞,-

Step 2:

- At i = 2; BC = 2+8 = 10, BD = 2+8 = 10, as shown in Table 2.
- The current node C [10,B] is the same as the previous node C [10,A], so result node C can be [10,B] or [10,A].
- Then, set C as "start" node for i = 3.

TABLE II. ITERATION 2

i	u	c	A	B	C	D	E
0	ABCDE	-	0,-	∞,-	∞,-	∞,-	∞,-
1	BCDE	A	0,-	2,A	10,A	∞,-	∞,-
2	CDE	B	0,-	2,A	10,B	10,B	∞,-

Step 3:

- At i = 3; CD = 10+5 = 15, CE = 10+10 = 20; as shown in Table 3.
- At current node D [15,C] is greater than the previous node D [10,B], so its value will be change to [10,B] from [15,C].
- Then, set D as "start" node for i = 4.

TABLE III. ITERATION 3

i	u	c	A	B	C	D	E
0	ABCDE	-	0,-	∞,-	∞,-	∞,-	∞,-
1	BCDE	A	0,-	2,A	10,A	∞,-	∞,-
2	CDE	B	0,-	2,A	10,B	10,B	∞,-
3	DE	C	0,-	2,A	10,B	10,B	20,C

Step 4:

- At i = 4; DE = 10+8 = 18; as shown in Table 4.
- The current node E [20,C] is greater than E [18,D], so the value is unchanged [18,D].
- Then, set E cannot be a "start" node because it is a "goal" node.
- Finish.

TABLE IV. ITERATION 4

i	u	c	A	B	C	D	E
0	ABCDE	-	0,-	∞,-	∞,-	∞,-	∞,-
1	BCDE	A	0,-	2,A	10,A	∞,-	∞,-
2	CDE	B	0,-	2,A	10,B	10,B	∞,-
3	DE	C	0,-	2,A	10,B	10,B	20,C
4	E	D	0,-	2,A	10,B	10,B	18,D

Therefore, the resulting path in Dijkstra's is A-B-D-E, as shown in Fig.2.

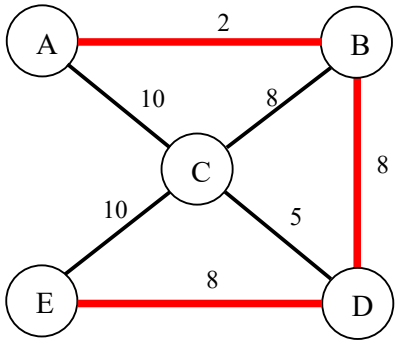

Fig. 2. The resulting path for Dijkstra's

*B. A**

A good heuristic function provides approximate costs that are close to actual costs. One of the heuristic functions that can be used in the shortest path problem is Euclidean Distance. A* use Euclidean Distance and evaluates nodes by combining g(n) and h(n) as:

$$f(n) = g(n) + h(n) \quad (1)$$

where:
f (n) = evaluation costs
g (n) = cost that already incurred from initial state to n state
h (n) = estimated cost to arrive at a destination from n state

A* work step by step as explained below.

1) Start
2) Set the "Start" node as a successor.
3) Calculate all evaluation cost which is connected from the successor.
4) Do the testing: if the prospective successor is the goal, it will stop, and if not, then proceed to the next step.
5) Determine the new successor from the best evaluation cost (minimum cost) and alphabet order (if there are two evaluation costs have the same value) from previous successors.
6) Moreover, go back to step 4.

For a better understanding, the authors explain how A* finds the shortest path from the example. As shown on Fig. 3, initial node = A and goal node = E. Each node has coordinates, namely: A(0,0), B(1.5,2.5), C(0,5), D(4,4), E(4,6).

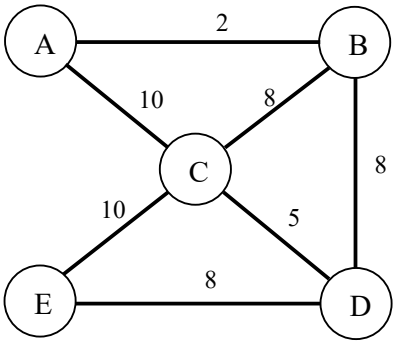

Fig. 3. Graph for A* Algorithm

Step 1:

$$h(BE) = \sqrt{(x_E - x_B)^2 + (y_E + y_B)^2}$$
$$h(BE) = \sqrt{(4 - 1.5)^2 + (6 - 2.5)^2}$$
$$h(BE) = \sqrt{(2.5)^2 + (3.5)^2}$$
$$h(BE) = \sqrt{6.25 + 12.25}$$
$$h(BE) = \sqrt{18.5}$$
$$h(BE) = 4.3$$

$$h(CE) = \sqrt{(x_E - x_C)^2 + (y_E + y_C)^2}$$
$$h(CE) = \sqrt{(4 - 0)^2 + (6 - 5)^2}$$
$$h(CE) = \sqrt{4^2 + 1^2}$$
$$h(CE) = \sqrt{16 + 1}$$
$$h(CE) = \sqrt{17}$$
$$h(CE) = 4.12$$

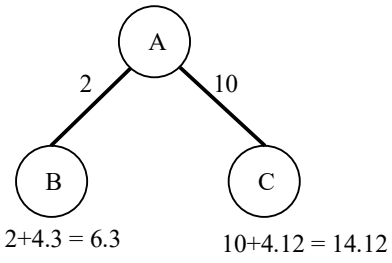

Fig. 4. Graph for step 1 in A*

Step 2:

$$h(CE) = 4.12$$
$$h(DE) = \sqrt{(x_E - x_D)^2 + (y_E + y_D)^2}$$
$$h(DE) = \sqrt{(4 - 4)^2 + (6 - 4)^2}$$
$$h(DE) = \sqrt{0^2 + 2^2}$$
$$h(DE) = \sqrt{4}$$
$$h(DE) = 2$$

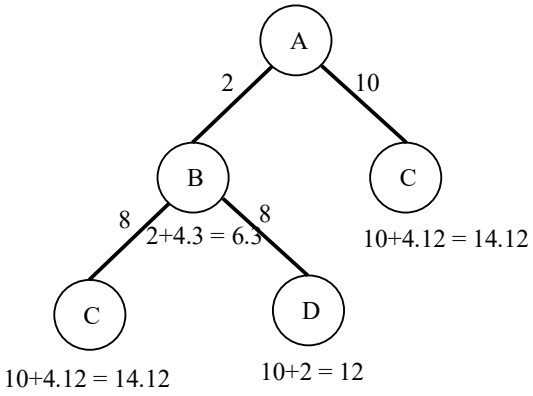

Fig. 5. Graph for step 2 in A*

Step 3:

$$h(CE) = 4.12$$

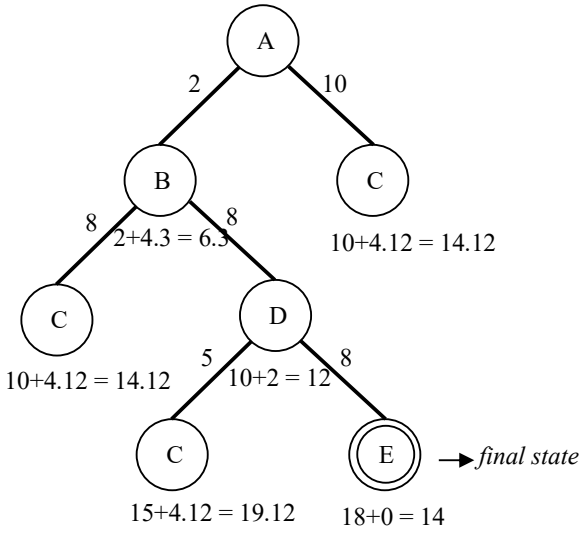

Fig. 6. Graph for step 3 in A*

Therefore, the resulting path in A* is A-B-D-E.

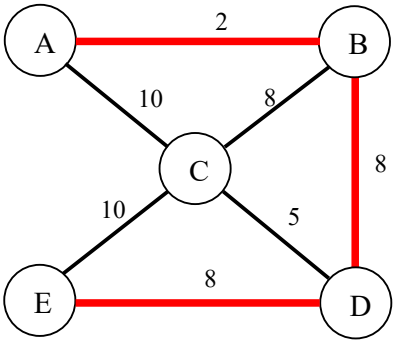

Fig. 7. The resulting path for A*

C. *Asymptotic time complexity*

Asymptotic notation is a notation that concern with the running time of the algorithm to explain the asymptotic time complexity. Asymptotic time complexity is an analysis of the asymptotic efficiency of an algorithm to determine the appropriate time complexity.

This research uses the Θ (Big-Theta) since it is more accurate than the O (Big-Oh) and the Ω (Big-Omega) notation. The Θ (Big-Theta) is used to categorize an algorithm into functions that describe the tight bound, which can be given the upper bound and lower bound of the function growth when the input of the function increases.

The function of t(n) is defined in Θ(g(n)) and denoted by t(n) ∈ Θ(g(n)). The t(n) function is given the upper and the lower limits by several multiples of positive constants of g(n) for all large values of n if there are some positive constants (c_1 and c_2) and some non-negative integers n_0 [8], so that

$$c_2 g(n) \leq t(n) \leq c_1 g(n) \quad (2)$$

for all $n \geq n_0$.

III. Result and Discussions

The computation was performed on PC Windows 8.1 Enterprise 64-bit (6.3, Build 9600), Intel(R) Core(TM) i5-5200 CPU @2.20 GHz (4 CPUs), and RAM 10240 MB. Users can choose the source SPBU, the goal SPBU, and the desired algorithm from the interface, as shown in Fig. 8 and Fig.9.

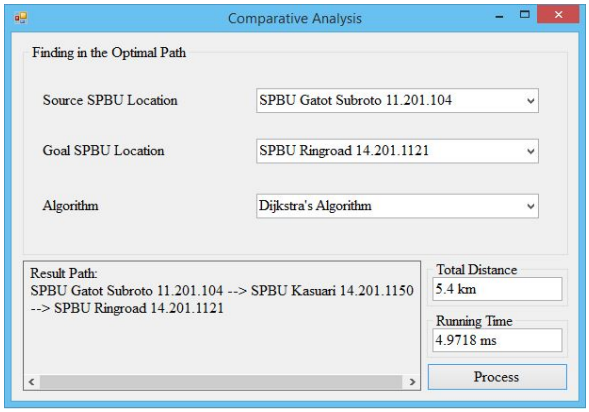

Fig. 8. The path, distance, and running time with Dijkstra's

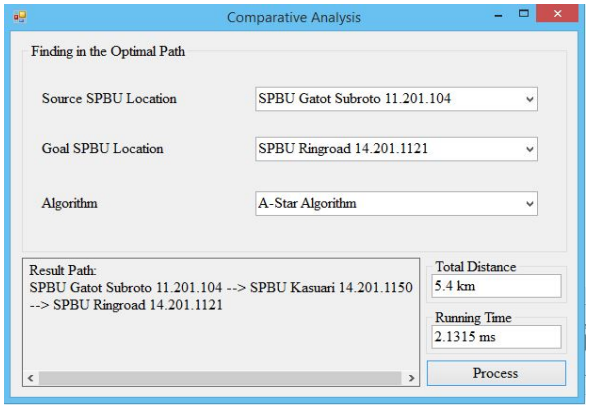

Fig. 9. The path, distance, and running time with A*

Moreover, the distance and running time of Dijkstra's and A* for different paths can be seen in Table 5 and Table 6.

TABLE V. THE RUNNING TIME OF DIJKSTRA'S

No.	Path	Total Distance (km)	Running Time (ms)
1.	C → X	13.9	7.5111
2.	V → E	23.3	7.9772
3.	D → A	21.9	7.777
4.	G → W	20.8	7.0161
5.	A → K	18.8	8.3112
Average			7.71852

TABLE VI. THE RUNNING TIME OF A*

No.	Path	Total Distance (km)	Running Time (ms)
1.	C → X	13.9	4.1866
2.	V → E	23.3	5.0898
3.	D → A	21.9	4.2552
4.	G → W	20.8	4.1213
5.	A → K	18.8	4.3774
Average			4.40606

Besides, the path and running time for Dijkstra's and A* can been seen in Fig. 10 and Fig.11. Meanwhile, the map for the graph can be seen in Fig.12.

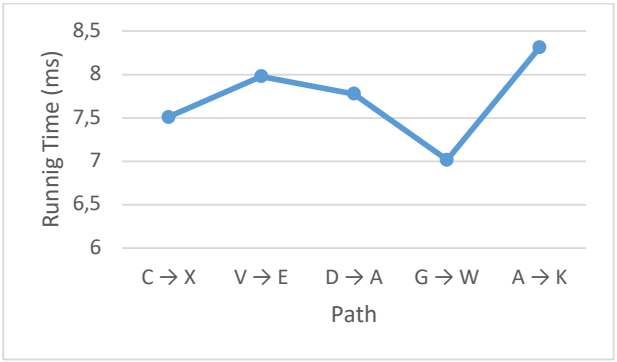

Fig. 10. Path and running time of Dijkstra's

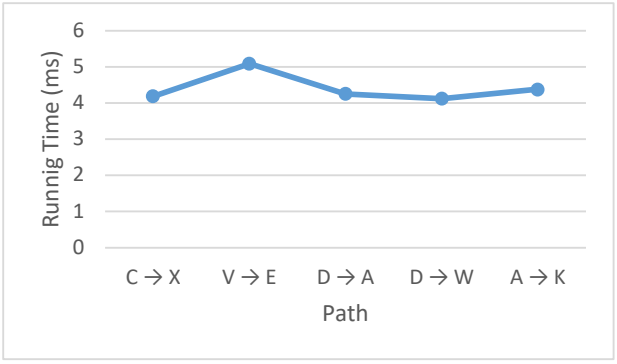

Fig. 11. Path and running time of A*

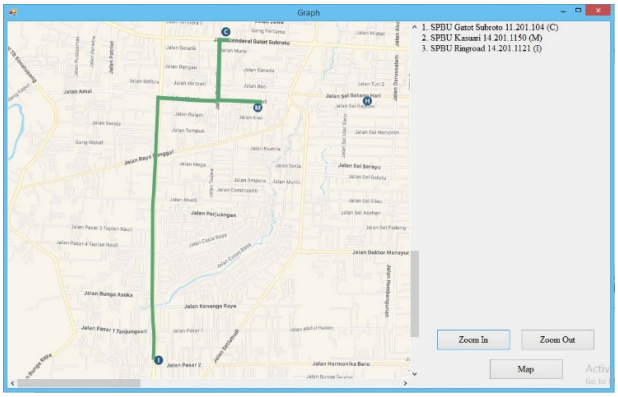

Fig. 12. Map and the graph

The computation result indicates that the shortest path search between SPBU can be completed with Dijkstra's and A*. For performance, the running time of A-Star is faster than Dijkstra's and it is related with A* principle which selects the location point based on the best heuristic value while Dijkstra's does not. For the complexity, Dijkstra's is $\Theta(n^2)$ and A-Star is $\Theta(m*n)$, where $0 \leq m \leq n$. $\Theta(n^2)$ means there is a nested loop with n as the variable involved. $\Theta(m*n)$ means there is a nested loop with m as the variable involved in the first loop and n as the variable involved in the second loop. Since m and n are free variables depend on the user, the complexity is considered equivalent.

IV. Conclusions

The experiments show that Dijkstra's and A* can solve the shortest path problem. However, in some cases, the routes produced by the two algorithms are different so that the total distance generated is also different. Furthermore, the running time of A* is faster than Dijkstra's where the average running time of Dijkstra's is 7.719 milliseconds and A* is 4.406 milliseconds. It is in line with A* principle that selects the location point based on the best heuristic value. Finally, the complexity of algorithm for Dijkstra's and A* is considered equivalent.

References

[1] T. Sutojo, E. Mulyanto, and V. Suhartono, Kecerdasan Buatan. Andi: Yogyakarta, 2011.

[2] B. Popa and D. Popescu, "Analysis of Algorithmic for Shortest Path Problem in Parallel," 17th International Carpathian Control Conference (ICCC), pp. 613-617, 2016.

[3] A. Laaksonen, "Competitive Programmer's Handbook. Draft," 2017.

[4] V. Pandey, S.C. Yadav, and P. Arora, "Retiming Technique for Clock Period Minimization using Shortest Path Algorithm," International Conference on Computing, Communication and Automation (ICCCA), pp. 1418-1423, 2016.

[5] J. C. D. Cruz, G. V. Magwili, J. P. E. Mundo, G. P. B. Gregorio, M. L. L. Lamoca, and J.A. Villasenor, "Items-mapping and Route Optimization in a Grocery Store using Dijkstra's, Bellman-Ford and Floyd-Warshall Algorithms," Proceedings of the International Conference IEEE Region 10 Conference (TENCON), pp. 243-246, 2016.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no.2, pp. 100-107, 1968.

[7] H. Reddy, "Path Finding-Dijkstra's and A* Algorithm's," available at http://cs.indstate.edu/hgopireddy/newalg.html, 2013.

[8] L. Anany, Introduction to The Design and Analysis of Algorithms. India: Person Education, 2009.