SHORTEST PATH WITH DYNAMIC WEIGHT IMPLEMENTATION USING DIJKSTRA'S ALGORITHM

Elizabeth Nurmiyati Tamatjita[1]; Aditya Wikan Mahastama[2]

[1]Department of Informatics, Sekolah Tinggi Teknologi Adisutjipto
Jl. Janti Blok R Lanud Adisutjipto, Yogyakarta, 55198
[2]Depatment of Informatics, Faculty of Information Technology, Universitas Kristen Duta Wacana
Jl. Dr. Wahidin Sudirohusodo No. 5 – 25, Yogyakarta, 55224
[1]tamatjita@gmail.com; [2]mahas@staff.ukdw.ac.id

ABSTRACT

Shortest path algorithms have been long applied to solve daily problems by selecting the most feasible route with minimum cost or time. However, some of the problems are not simple. This study applied the case using Dijkstra's algorithm on a graph representing street routes with two possible digraphs: one-way and two-way. Each cost was able to be changed anytime, representing the change in traffic condition. Results show that the usage of one way digraph in mapping the route does make the goal possible to reach, while the usage of two-way digraph may cause confusion although it is probably the possible choice in the real world. Both experiments showed that there are no additional computation stresses in re-calculating the shortest path while going halfway to reach the goal.

Keywords: *shortest path, dynamic weight, Dijkstra's algorithm, graph, digraph*

INTRODUCTION

A graph is a symbolic representation of a network and its interconnections. A graph shows an implication of reality simplified as a set of connected nodes. Graph theory is a study of mathematics which codes and measures the features of a network. Graph theory has been enriched in these last decades with influences from social sciences (Rodrigue & Ducruet, 2015). An implementation of graph theory is to find the shortest path connecting a start and end nodes through several other nodes. This search takes into consideration a cost optimization regarding distance or other costs, so in the end, there will be only one optimal path which has the most minimum cost.

Therefore, the usual case to which shortest path algorithms were implemented to solve is transportation problems, where transport cost are the main compensation expendable to run a route between two locations. These were a static case since a few decades ago there weren't many dynamic factors affecting transport cost, and some established systems are still using the same measurement such as freight forwarders, packet services, etc.

However, when computer games come to a rise, the case are expanded to solve transportation problems in games, which in turn represents today's real condition on a micro scale. Computer games' transportation problems are more complex because it involves just-in-time decision to change the next route when additional problems came up, such as unexpected monsters, disasters, and so on. These are not too far away from today's traffic problems. Traffic congestion often came out of nowhere and is not expected before. Gladly the automatic traffic control has the data which shows on what route the congestion occurs and how bad the traffic is, so it is feasible to model the graph and cost using these data in order to find the most possible new route.

Dijkstra (1959) created one of the earliest algorithms in finding the shortest path, based on selecting the most minimum transportation cost (called weight) of an edge connecting two nodes (basically a way connecting two locations), expand the weight measurement to the next possible nodes, calculating the total weight and updated the route if a new total minimum weight is found. The network is modeled as a directed graph consisting of nodes and directed edges, without any loop edges coming from and pointing to a same node (Harju, 2011).

Dijkstra's algorithm has been implemented mainly to solve static transportation problems such as the shortest path between two cities in Central Java (Sunaryo, Siang & Chrismanto, 2012) and South Sumatera (Fitria & Triansyah, 2013), to define the shortest path using multi-means of public transportation (Arifianto, 2012), and to locate the nearest public facility like hospitals, hotels and bus stations in a city (Sholichin, Yasindan, & Octoviana, 2012).

A nearly dynamic implementation was used to solve adaptive drinking water distribution problem for housing (Prasetyo, 2013). However, the simulation does not include a real-time dynamic change of water distribution capacity as its weights. A comparative study has also been conducted regarding computational loads of Dijkstra's algorithm against Floyd-Warshall algorithm for a same certain case (Djojo & Karyono, 2013).

Although the algorithm does expand way too wide in search for the optimum weight resulting in a rather inefficient time (Głabowski *et al.,* 2013), it is quite simple to be implemented, so for a limited or a selected number of alternative nodes, this algorithm should fit and gives a clear impression of whether a dynamic change in edge weights between the start and end node is still feasible to implement and how it will affect the computational load. The expected results may show whether dynamic changes in weight is still feasible to be implemented and used for everyday traffic problem solving, e.g. for a fire-fighter to reach the fire location, an ambulance en route to a hospital, etc.

This study will focus to resolve several problems. First, to find out whether Dijkstra's algorithm is feasible to solve and find the shortest path of a directed graph with dynamic weights. Second, to know whether or not the path offered as the final solution is the correct shortest path. Finally, to find whether or not a performance problem occurs when the weights are changed dynamically?

For this study, an experimental environment is limited to a maximum of 30 nodes, with one start and one end nodes. The graph used for modelling are directed graph which doesn't contain any direct loop. Weight changes are able to be done real-time through the on-screen interface, regardless the current route calculation has been commenced or not. The result (final overall route selected) will be displayed on the screen to aid manual study for the correct path.

There are some purposes to be achieved in this research. The first one is to create a model for shortest path analysis with dynamic weight using Dijkstra's algorithm. Next, to study whether certain cases of dynamic weight change may render a solving failure which leads to wrongly selected route or an unfinishable route. Third, observing whether a raise in performance load (measured in time needed to calculate) will occur for certain cases of dynamic weights.

METHODS

A graph is a symbolic representation of a network and its interconnections. A graph shows an implication of reality simplified as a set of connected nodes. Graph theory is a study of mathematics

which codes and measures the features of a network. According to Astuti (2015), graph G (V, E) is a collection of two sets: (1) set V which elements are the nodes or vertices and (2) set E which elements are the edges.

The amount of members in set V determines the order of graph G, while the amount of members in set E is the size of graph G. Examples of graphs are shown in Figure 1.

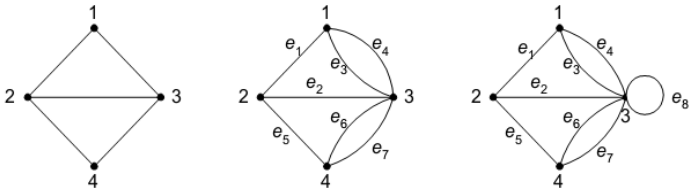

Figure 1 Graph Examples

The second graph from left in Figure 1 shows an occurrence of multiple edges or parallel edges e_3= (1, 3) and e_4= (1, 3) which connects a same pair of nodes. The third graph from left shows an occurrence of loop e_8 which connected to and for a same node.

If the edges are having weight (cost needed to pass the edge), then the graph is called a weighted graph. The weight is written near the edge as the name and placed in certain way to avoid confusion.

According to the orientation of edges, graph falls into two categories; undirected graph or just "graph" and directed graph or "digraph." Digraph has arrowheads on edges showing the direction in which the edge is leading to, such as displayed in Figure 2.

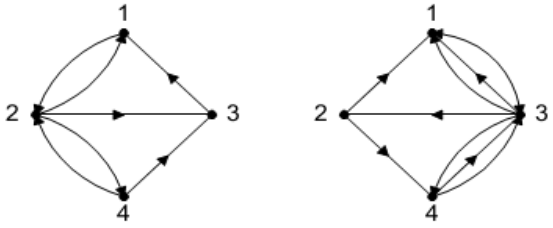

Figure 2 Directed Graph

Ruohonen (2013) defined that Dijkstra's algorithm is used to solve shortest path problem (finding a path with the minimum length) from a start node to an end node in a weighted graph, and the weight should be a positive number. Given G is a weighted digraph with nodes V (G) = {v_1, v_2,…,v_n} and shortest path in question is from $v_1$to v_n, Dijkstra's algorithm begins from v_1.

During its iteration, Dijkstra's algorithm will find a successor node which costs a smaller up-to weight than the current node. Selected successor nodes are kept aside and not involved in the next iteration. The whole pseudo code for Dijkstra's algorithm is declared in Figure 3.

```
Function Dijkstra (M: graph weight in an array of nodes, a :
integer)

Declare
D, S : array[1..n] of integer
i, j, k min : integer

Steps
{ Step 0 (initialisation): }
for i ← 1 to n do
  S[i] ← 0
  D[i] ← M[a, i]
Endfor
{ Step 1: }
S[a] ← 1 { input initial node into S }

{ Steps 2, 3, ..., n-1 : }
for k ← 2 to n − 1 do

{ look for node j as to S[j] = 0
  Dan D[j] = Minimum{D[1], D[2], ..., D[n] }
  min ← D[1]
  j ← 1
  for i ← 2 to n do
    if (S[i] = 0) and (D[i] < min) then
      min ← D[i]
      j ← i
    endif
  endfor

  S[j] ← 1 {Node j is already selected into the shortest path}
  {recalculate D[i] from node a to node i ∉ S}
  for i ← 1 to n do
    if S[i] = 0 then
      if D[i] > (D[j] + M[j,i]) then
        D[i] ← D[j] + M[j,i]
      endif
    endif
  endfor
endfor

return D
```

Figure 3 Pseudocode for Dijkstra's Algorithm

The study constructed these models to support the research. A system model used in this research, which follows the block diagram shown in Figure 4.

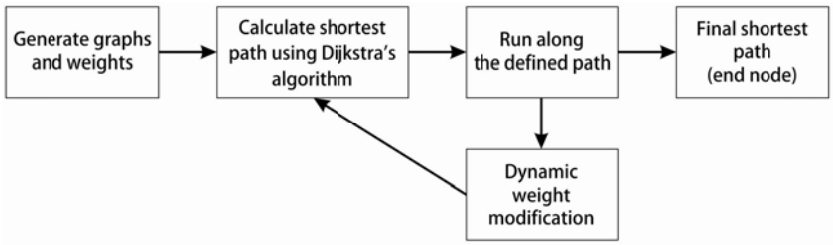

Figure 4 System Model

A data structure model used to store the digraph data. The data structure should be able to store nodes, edges relating two nodes and its weight. The weight should be able to be modified dynamically.

These models are implemented as a program which used to test the cases in this study. Observations and evaluations are conducted using the program. The resulting program has a graphical user interface which helps the ease of use in visualizing the map model of an area, modifying the original weight of an edge (stated as the distance between two nodes), or dynamically modify it by changing a node terrain or putting certain obstacles available into an edge.

The terrain and obstacle represent a change in traffic load or congestion in a successive road, which may lead the system to select another feasible road on an intersection. The selected path is visualized on-screen by an ambulating red sphere running along the "shortest" path from the start node to end node, and changes in path selection are easy to observe visually with the sphere deviating from its "original" shortest path. The animated sphere is also used to observe whether the offered path is finishable, correct or render to an unfinished loop. The program interface is presented in Figure 5.

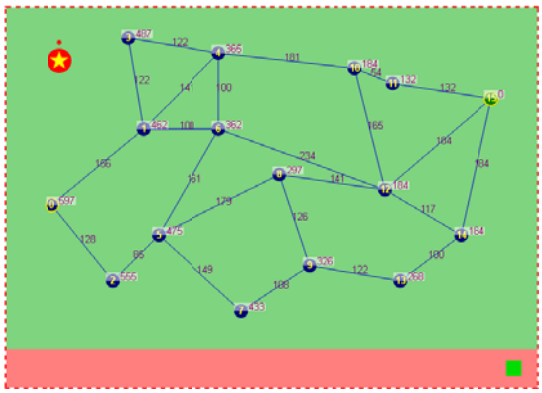

Figure 5 Testing Program Interface

The path is recalculated from a node to be reached by the red sphere. Recalculation occurs whenever there is a weight change in at least one of established shortest path edges. The recalculated path is then merged with the previous path which has trodden and become the final shortest path.

Two predefined “maps” or certain forms of graphs are used in forty testing sessions, with each map for twenty sessions conducted. Map A has a total weight of 2000 for its initial shortest path, which is also its total initial shortest path length, with standard completion time 200 seconds. Map B has a total weight of 2500 for its initial shortest path, with 2500 units long and standard completion

time of 250 seconds. Map A and Map B has their special characteristics in which Map A has most edges exiting from a node with slight differences in weight – which reflects small differences in edge length, while Map B has a very different weight for most existing edges, as shown in Figure 6.

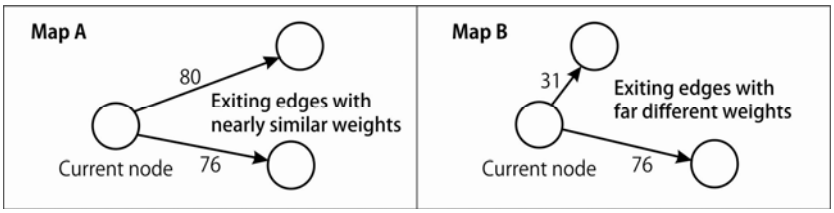

Figure 6 Characteristics of Map Types

These served as possible conditions of a road map, where sometimes the path choice are easily available because they are near identical; and sometimes there are only a few choice, including turning back if necessary, to obtain the shortest path. Every two sessions, the same dynamic modifications conducted using a single-way digraph and two-way digraph.

For the two-way digraph test, the maps are not entirely made as the two-way digraph, only selected edges are made to have parallel edges to simulate the real condition of city streets, which frequently are a two-way system, and vehicles are often able to turn back to select a better path from a node before. The two-way digraph also uses to test the possibility of Dijkstra's algorithm in solving it, since researches conducted before have not been discussing about a possibility of the two-way digraph. The types test sessions conducted are illustrated in Table 1.

Table 1 Test Types

Map	Dynamic Change #	Digraph	Test Session
Map A	Mod #1	One-way	Test A1-1
		Two-way	Test A1-2
	Mod #2	One-way	Test A2-1
		Two-way	Test A2-2
	Mod #3	One-way	Test A3-1
		Two-way	Test A3-2
	Mod #4	One-way	Test A4-1
		Two-way	Test A4-2
	Mod #5	One-way	Test A5-1
		Two-way	Test A5-2
	Mod #6	One-way	Test A6-1
		Two-way	Test A6-2
	Mod #7	One-way	Test A7-1
		Two-way	Test A7-2
	Mod #8	One-way	Test A8-1
		Two-way	Test A8-2
	Mod #9	One-way	Test A9-1
		Two-way	Test A9-2
	Mod #10	One-way	Test A10-1
		Two-way	Test A10-2

Table 1 Test Types (continued)

Map	Dynamic Change #	Digraph	Test Session
Map B	Mod #1	One-way	Test B1-1
		Two-way	Test B1-2
	Mod #2	One-way	Test B2-1
		Two-way	Test B2-2
	Mod #3	One-way	Test B3-1
		Two-way	Test B3-2
	Mod #4	One-way	Test B4-1
		Two-way	Test B4-2
	Mod #5	One-way	Test B5-1
		Two-way	Test B5-2
	Mod #6	One-way	Test B6-1
		Two-way	Test B6-2
	Mod #7	One-way	Test B7-1
		Two-way	Test B7-2
	Mod #8	One-way	Test B8-1
		Two-way	Test B8-2
	Mod #9	One-way	Test B9-1
		Two-way	Test B9-2
	Mod #10	One-way	Test B10-1
		Two-way	Test B10-2

Weight modification is done by altering a predetermined node for each test to increase the weight required for passing edges to reach this node. Alternation is done by selecting a node then slide the "terrain level" to water or hill (default is 0 or a flat land), to increase the weight by adding certain number to arriving edges' weight. This can also serve as traffic congestion weight. The predetermined node is altered so as the original incoming path's weight become not eligible to pass, and an alternative path has to be selected. The intensity of the alterations is indicated by the radius of the bluish mists.

Modification with a bigger number indicates previous modification added with another modification, so Mod #10 will have at least ten modifications as it resulted from all previous modifications. Figure 7 illustrated how modifications #1 to #3 of Map A are made. After each modification, the shortest path is recalculated and waited until the red sphere finished following the path up to the end node. Next test session commenced by performing all modifications and wait until the end node is reached.

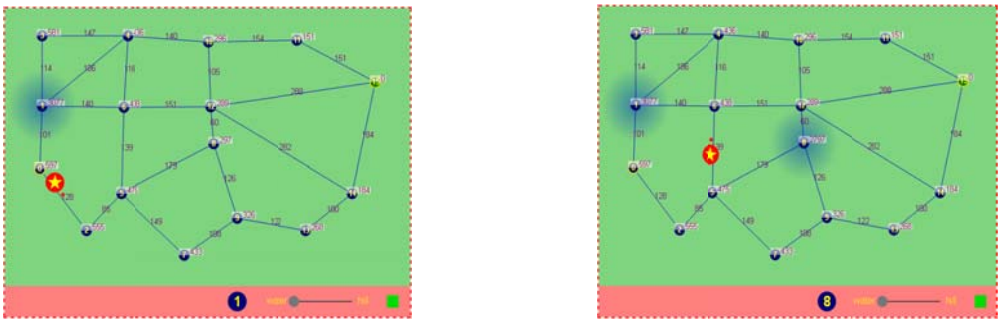

Figure 7 Mod #1-#3 for Map A

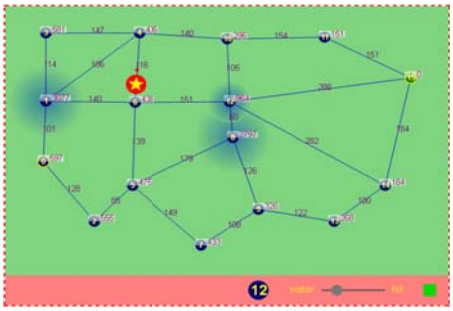

Figure 7 Mod #1-#3 for Map A (continued)

For every test session conducted, data is collected regarding shortest path solved, finish ability of the path, factors occurred regarding finish ability, and time of completion.

RESULTS AND DISCUSSIONS

Data collected from testing sessions sum up the results as presented in Table 2. Not all suggested path are subject to finishability, therefore, completion time regarded as not available.

Table 2 Test Session Results

Map	Dyn. Change #	Digraph	Test Session	Finishable	Complet. time (s)
Map A (Initial shortest path: (1) total weight: 2000, (2) length: 2000, (3) standard completion time 200seconds)	Mod #1	One-way	Test A1-1	Yes	201
		Two-way	Test A1-2	Yes	201
	Mod #2	One-way	Test A2-1	Yes	203
		Two-way	Test A2-2	Yes	203
	Mod #3	One-way	Test A3-1	Yes	210
		Two-way	Test A3-2	No – Loop	
	Mod #4	One-way	Test A4-1	Yes	206
		Two-way	Test A4-2	No – Loop	
	Mod #5	One-way	Test A5-1	Yes	201
		Two-way	Test A5-2	Yes	205
	Mod #6	One-way	Test A6-1	Yes	215
		Two-way	Test A6-2	No – Loop	
	Mod #7	One-way	Test A7-1	Yes	208
		Two-way	Test A7-2	No – Loop	
	Mod #8	One-way	Test A8-1	Yes	204
		Two-way	Test A8-2	Yes	204
	Mod #9	One-way	Test A9-1	Yes	202
		Two-way	Test A9-2	Yes	202
	Mod #10	One-way	Test A10-1	Yes	216
		Two-way	Test A10-2	Yes	211

Table 2 Test Session Results (continued)

Map	Dyn. Change #	Digraph	Test Session	Finishable	Complet. time (s)
Map B (Initial shortest path: (1) total weight: 2500, (2) length: 2500, (3) standard completion time 250 seconds)	Mod #1	One-way	Test B1-1	Yes	223
		Two-way	Test B1-2	Yes	222
	Mod #2	One-way	Test B2-1	Yes	235
		Two-way	Test B2-2	Yes	251
	Mod #3	One-way	Test B3-1	Yes	258
		Two-way	Test B3-2	Yes	245
	Mod #4	One-way	Test B4-1	Yes	258
		Two-way	Test B4-2	Yes	262
	Mod #5	One-way	Test B5-1	Yes	255
		Two-way	Test B5-2	No – Loop	
	Mod #6	One-way	Test B6-1	Yes	256
		Two-way	Test B6-2	No – Loop	
	Mod #7	One-way	Test B7-1	Yes	253
		Two-way	Test B7-2	No – Loop	
	Mod #8	One-way	Test B8-1	Yes	252
		Two-way	Test B8-2	Yes	252
	Mod #9	One-way	Test B9-1	Yes	252
		Two-way	Test B9-2	Yes	252
	Mod #10	One-way	Test B10-1	Yes	252
		Two-way	Test B10-2	Yes	252

From 40 test sessions conducted, seven of them (17,5%) are failed to be finishable. This means that the red sphere keeps looping between two nodes and not selecting another option as a way out of this. Three modifications (7,5%) even resulting better completion times than the original standard, and seven pairs of one-way and two-way tests or 14 sessions (35%) result in the same completion time.

The result is inevitably subjected to the features of Map A or Map B. Map A has the feature of which alternative edges have nearly similar weights, while Map B has the unique feature of which alternative path(s) may have a very different weight, speaking of individual edge weights, total alternative path weight or even node count. Thus, Map B has a wider variation of alternative path weights than Map A. Nevertheless, there are several discussion points regarding the results.

First, identical completion time between pairs of one-way and two-way tests is a result of same new path rendered and one or two-way edge are not heavily affecting the resulting new path. This especially occurred at the beginning or near the edge of the path where options are rare.

Second, loops occurred whenever an edge selected as the best beginning “way back” also has the smallest weight in the “next” intersection, which originally was the “previous” intersection. This leads to selecting the same edge as a path over and over and the red sphere are keep rotating over the edge. It may be a feature specific to the map.

Third, completion times which are lower than the standard completion time is a result of over adding weight on an edge, then the sphere turned back on its first edge and selected another edge which happens to have less weight and less node compared to the added modification.

Fourth, another possible explanation for completion time lower than standard is that the red sphere began to move right after the first edge is determined. This is a programming feature which

may or may not interfere with the final "correct" shortest path, as is it unknown whether the first edge determination is final or not for the corresponding path.

Fifth, each completion time has a small deviation compared to the real length of path. Every second is roughly equal to 10 units of route length. Although completion time cannot be rendered as detail as milliseconds, the deviation which manually calculated in seconds shows only 1%, thus the recalculation time is so fast and not effective when implemented to the real map of the same complexity.

CONCLUSIONS

The study made three conclusions from research conducted. First, Dijkstra's algorithm is feasible to solve shortest path problem with dynamic weights, using a one-way digraph (digraph without parallel edges). Second, Dijkstra's algorithm is not feasible to solve shortest path problem with dynamic weights, using a two-way digraph (digraph with opposite parallel edges), because a large "loop" may occur between two nodes after a recalculation. Therefore regular road map consisting of two-way traffic is not to be solved as is using Dijkstra's algorithm. A suggestion to eliminate this is by programming that parallel edges between two nodes are not to be selected as a sequence in the path. Third, there are no performance drawbacks in recalculating the shortest patch every time a change happened. For example, when situated as a problem-solving alternative of how to avoid city traffic congestion and road class from the fire station to fire site, this has no significant effect in computational time. Effects on a wider graph consisting of more than 30 nodes are unknown.

REFERENCES

Astuti, Y. D. (2015). *Dasar Teori Grafin "Logika dan Algoritma" lecture notes*. Retrieved February 18, 2015 from http://rifki_kosasih.staff.gunadarma.ac.id/Downloads/files/37568/Bab+1+-+Dasar+Teori+Graf.pdf

Arifianto, S. (2012). *Sistem Aplikasi Penentuan Rute Terpendek Pada Jaringan Multi Moda Transportasi Umum Menggunakan Algoritma Dijkstra* (Master's thesis). Semarang: Program Studi Sistem Informasi, Program Pascasarjana Universitas Diponegoro

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271

Djojo, M. A., & Karyono. (2013). Pengukuran Beban Komputasi Algoritma Dijkstra, A*, dan Floyd-Warshall pada Perangkat Android. *ULTIMA Computing*, *5*(1), 13-17.

Głabowski, M., Musznicki, B., Nowak, P., & Zwierzykowski, P. (2013). Efficiency Evaluation of Shortest Path Algorithms. In *The Ninth Advanced International Conference on Telecommunications (AICT) 2013 Proceedings* (pp. 154-160). Rome, Italy.

Harju, T. (2011). *Lecture Notes on Graph Theory*, Department of Mathematics, University of Turku, Finland. Retrieved February 18, 2015 from http://cs.bme.hu/fcs/graphtheory.pdf

Prasetyo, V. Z. (2013). *Penerapan Algoritma Dijkstra Untuk Perutean Adaptif Pada Jaringan Pendistribusian Air PDAM di Kabupaten Demak* (Bachelor's thesis). Semarang: Jurusan Matematika, Fakultas MIPA, Universitas Negeri Semarang

Rodrigue, J-P., & Ducruet, C. (2015). *Graph Theory: Measures and Indices*. Retrieved February 18, 2015 from https://people.hofstra.edu/geotrans/eng/methods/ch1m3en.html

Ruohonen, K. (2013). *Graph Theory*. Retrieved February 18, 2015 from http://math.tut.fi/~ruohonen/GT_English.pdf

Sholichin, R., Yasindan, M., & Oktoviana, L. T. (2012). Implementasi Algoritma Dijkstra Dalam Pencarian Lintasan Terpendek Lokasi Rumah Sakit, Hotel Dan Terminal Kota Malang Berbasis Web. *Jurnal Online Universitas Negeri Malang, (Online).*

Sunaryo, Siang, J. J., & Chrismanto, A. R. (2012). *Pencarian Jalur Terpendek Antar Kota Di Jawa Tengah Dan D.I. Yogyakarta Dengan Algoritma Dijkstra Via SMS Gateway* (Bachelor's thesis). Yogyakarta: Program Studi Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana

Triansyah, F. A. (2013). Implementasi Algoritma Dijkstra Dalam Aplikasi Untuk Menentukan Lintasan Terpendek Jalan Darat Antar Kota Di Sumatera Bagian Selatan. *Jurnal Sistem Informasi (JSI)*, *5*(2), 611-621.