

Application of Greedy and Djikstra Algorithm for Finding the Fastest Time Needed in Travelling

Riva Syafri Rachmatullah - 13512036
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512036@std.stei.itb.ac.id

Abstract — every person in this new world need to travel as fast as possible to reach his/her destination. The shortest path doesn't always give the fastest time needed because of traffic and the road. Narrow road and damaged road reducing the speed of travelling makes the travelling slower. Slower the speed, longer the time it takes to get to the destination. With knowing speed and distance to destination we can get time needed to travel so that we can predict which path should we take.

Index Terms—Greedy, Djikstra, Algorithm, Shortest path, Time.

I. INTRODUCTION

Shortest path problems is a common problem and often becoming an issue for drivers. An example of this problem is the TSP which stands for Travelling Salesman Person. Many algorithm can solve this problem with its advantages and disadvantages. Some algorithm which can solve this problem is Greedy Algorithm, Branch and Bound Algorithm, and Dynamic Programming.

Many people often search the shortest path from place to place with hope that they get the fastest time to get there. Physics tells us that average speed can be calculated from distance traveled divided by time needed to travel there. From there, people start to think that they can get the fastest time if they can get the shortest distance with the same speed.

However, this world started to change and have so many complex problem to solve. The shortest path often gives the fastest time but it can give the slowest time with so many possibilities. Traffic jam, for example, is a new problem in this world. Traffic jam occurs when there are so many vehicle which crowding the traffic so that they stopped for a long time in just a single traffic. There is also another example of problem such as narrow roads and damaged roads which often occurs in Indonesia. Car can't get through narrow roads, even motorcycle's drivers have to slow down to make sure their motorcycle fit that roads. Damaged roads make travelling becomes difficult but more

interesting in other way. Damaged roads and narrow roads are slowing down the traveler so that it makes the time travelling is slower than the normal one.

This combined problems make us to think more about how to get the fastest time by not only just getting the shortest path but also considering side problem that may impact travelling speed such as traffic jam, damaged roads, narrow roads, damaged bridge, etc. What people really need on travelling in this present world isn't just the shortest path to the place they want. They want the fastest time more than the shortest path.

II. FUNDAMENTAL THEORY

A. Distance and Displacement

Distance and displacement are two quantities that many people think they have the same meaning but they are really different from definition and what they are used to. Distance is a *scalar quantity* that refers to amount of ground that an object has covered during its motion whereas displacement is a *vector quantity* that refers to changing in position from its initial place.

The formula of distance and displacement is

$$\text{displacement} = \Delta \text{position} \quad (1)$$

$$\text{distance} = \text{length from a place to another place} \quad (2)$$

To make it easier, we can look at Fig. 1.

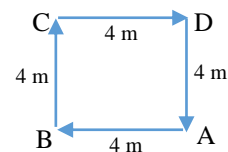


Fig. 1

The movement of that picture is from node A back to A again. From that picture, distance is applied from node A back to A again; that is 4+4+4+4 equals 16 meters whereas

displacement is a movement from node A back to A which is zero.

B. Average Speed and Velocity

Distance and displacement is the element for speed and velocity. Distance and displacement have slightly different meanings, so do speed and velocity.

Speed is a *scalar quantity* that we always see in speedometer on every vehicle. Speed refers to how fast an object is moving. Speed can be thought as the rate at which an object covers distance [1]. When object moves, it always has a change in speed. We can calculate the average speed with formula below:

$$\text{average speed } (s_{avg}) = \frac{\text{distance traveled}}{\text{time traveled}} = \frac{d}{t} \quad (3)$$

so that from equation above, time traveled can be calculated with:

$$\text{time traveled} = \frac{\text{distance traveled}}{\text{average speed}} = \frac{d}{s_{avg}} \quad (4)$$

Velocity is a *vector quantity* that can be thought as the rate at which an object changes its position [3]. If someone is stepping forth and back from its position, from definition of velocity we can get the result of velocity as zero. We can calculate the average velocity with formula below:

$$v_{avg} = \frac{\Delta \text{position}}{\text{time}} = \frac{\text{displacement}}{\text{time}} \quad (5)$$

C. Weighted Graph

Graph used to represent discrete objects and its relationship between that objects. Graph is defined by set of (V, E) that was written in notation:

$$G = (V, E)$$

with V is a set of *vertices* or *node* that mustn't empty and E is set of edges or arcs connecting couple of nodes that can be empty [4].

Weighted graph is a graph with each of edge in there has a weight or value. Weight on the edge of graph represent distance from city to the other city, time to send a messages, etc. In this shortest path problem, we know that weight of edge can't be negative so that the value is always positive.

Path on graph is a sequence of consecutive edges in a graph with the length of the path is total amount of weight of edges that used on path. This definition is used to

determine the shortest path from a node to another node.

D. Greedy Algorithm

"Take what you can get now!" is greedy algorithm's principle [2]. Greedy algorithm is step by step solving algorithm. There are so many choices to explore on each of step. Approach used on greedy algorithm is making every step looks the best by choosing the best option in that step. Greedy algorithm take the best on that step without looking at consequences on the future. Greedy algorithm is used to hope that choosing local optimum will make every step to end in global optimum.

Greedy algorithm has a common schema. Optimization problem on greedy algorithm's context is consist of [2]:

1. Set of candidate
This set contains elements that forming the solution. On every step, a candidate is taken from its set. An example of this set is set of coin, set of job to be done, and set of node in graph.
2. Set of solution
Set of solution is a subset of candidate. This set contains candidates that every candidate in it was chosen as a part of solution of its problem.
3. Selection function
Selection function or sometimes called as selection predicate is a function that selecting the highest possible candidate to make the most optimal solution in every steps. Candidate that has been taken to be part of the set of solution won't be considered in and affect the next step.
Prototype of selection function is [2]:

```
function SELECTION(C : set of candidate) →  
candidate  
{ return a candidate that was chosen from C  
with certain criteria. }
```

4. Feasible function
Feasible function or sometimes called as feasible predicate is a function to check if a chosen candidate is worth enough to become a part of set of solution. In this case, worth enough means that a chosen candidate with the set of solution that was constructed before doesn't break the defined constraints. Candidate which is worth enough is inserted to set of solution whereas the unworthy candidate is thrown away and never be considered again.
Prototype of feasible solution is [2]:

```
function FEASIBLE(S : set of candidate) →  
boolean  
{ this function returns true if S is a
```

```
solution which doesn't break the defined
constraint; false elsewhere. }
```

5. Objective function

Objective function is a function to maximize or minimize the value of solution.

Combining with all five schemas, in every step, this algorithm use solution function to check whether a set of solution is a wanted solution. Prototype of solution function is [2]:

```
function SOLUTION(S : set of candidate) →
boolean
{ this function returns true if S is a
solution of the defined problem. }
```

With this greedy algorithm, we hope that we can get the global optimum solution with taking the best choice on every step. However, the solution generated by greedy algorithm doesn't always become the optimum solution but may become a suboptimum solution because of two factor [1]:

1. Greedy algorithm doesn't check all alternative solution that can be made.
2. Choosing a selection function which isn't effective.

Many problems can be solved by greedy algorithm such as exchange coin problem, knapsack problem, job scheduling with deadlines problem, minimum spanning tree problem, shortest path problem, etc.

E. Dijkstra Algorithm

Dijkstra algorithm is greedy based algorithm for directed weighted graph. This algorithm is written by Edsger Wybe Dijkstra. Dijkstra algorithm solve this shortest path problem into several steps.

On every steps, take edge with minimum weight that connect a chosen node with the other unchosen node. Path from initial node to the new node must be a shortest path in all other path to node that hasn't been chosen [2].

From [2], djikstra algorithm pseudo code can be written into:

```
function Dijkstra(input M : matrix, a :
initial_node) → table
{Search the shortest path from initial node
a to every other node. M is an adjacency
matrix with G as a weighted graph and a as
a initial node. This function returns a
table that contains length of the shortest
path from a to the other node. }
```

Declaration

```
D, S : table
i : integer
```

Algorithm

```
{ Initialization }
for i ← 1 to n do
  S[i] ← 0
  D[i] ← m[a,i]
endfor

{ Step 1 }
S[a] ← 1
{ because a is initial node so that a
must be chosen in the shortest path }
D[a] ← INFINITE
{ there is no shortest path from a to a }

{ Step 2 and forth }
for i ← 2 to n - 1 do
  Search j so that S[j] = 0 and D[j] =
  Minimum{D[1], D[2], ..., D[n]}
  S[j] ← 1
  { node j is chosen in the shortest
path }
  Count the new D[i] from node a to i
  that wasn't in table of S with
  D[i] ← Minimum{D[i], D[j] + M[j,
i]}
endfor

return D
```

III. FINDING THE FASTEST TIME USING GREEDY ALGORITHM AND DYNAMIC PROGRAMMING

Given a distance map of big cities in Jawa Island in Indonesia:

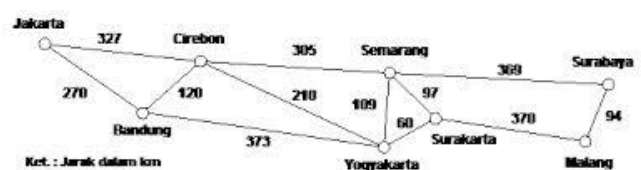


Fig. 2

That picture is taken from a worpress page. The link is <http://lialaesa.wordpress.com/2012/03/20/solving-tsp-8-kota-di-pulau-jawa/>. From fundamental theory we have, we can get time travelled by distance travelled divided by average speed. Given assumption that we can get average speed from satellite or GPS feature. Average speed from a node to another node is taken from the fastest car on that road. We can give an assumption like that so that the figure becomes:

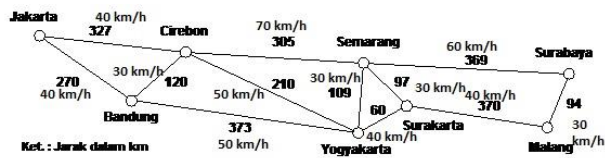


Fig. 3

Suppose that we want to travel from Bandung to Semarang. This problem is increased not just selecting the shortest path but also calculate the time need to pass from a node of city to another node of big city. Every data of average speed in the figure is random so the data is just a sample. This problem can be solved by two approach algorithm.

A. Using a Simple Greedy Algorithm

Simple greedy algorithm to solve this problem can be concluded in several main idea [2] such as greedy by distance, greedy by speed, and greedy by density (distance/time). Greedy by distance is as same as simple greedy for solving the simple shortest path or *Travelling Salesman Person* problem.

From figure 3, greedy by distance will give result from Bandung to Semarang as this table below:

Table 1

No	Current City	Chosen City	Next City	Shortest distance
1.	Bandung	Cirebon		120 m
2.	Cirebon	Yogyakarta		210 m
3.	Yogyakarta	Surakarta		60 m
4.	Surakarta	Semarang		97 m

this table gives a path from Bandung to Semarang through Cirebon, Yogyakarta, then Surakarta. This is the suboptimum solution with 487 meters. The optimum solution or the shortest distance to get from Bandung to Semarang is 425 meters.

Using simple greedy by speed doesn't give us the optimum solution. This greedy algorithm takes the fastest speed the route can take. We can look at the table below:

Table 2

No	Current City	Chosen Next City	Fastest speed
1.	Bandung	Yogyakarta	50 km/h
2.	Yogyakarta	Surakarta	40 km/h
3.	Surakarta	Malang	40 km/h
4.	Malang	Surabaya	30 km/h
5.	Surabaya	Semarang	60 km/h

Table above gives us a route from Bandung to Semarang through Yogyakarta, Surakarta, Malang, Surabaya, then

Semarang. This is the worst greedy we take because by taking speed doesn't ensure taking the fastest time needed to travel, even we can't get the shortest path from it. Looking at table 2, we have to get through 1266 meters and 27.4933 hours. This is really bad for solution.

There is greedy algorithm that much better than the other simple greedy algorithm. This algorithm is as same as the greedy by density algorithm [2]. On every step when solving the problem, time is calculated like density, by choosing the least result from d/s_{avg} . This strategy try to minimize the minimum time elapsed needed to go from a city to city destination.

Greedy algorithm for this problem is for choosing the least time needed from a node to another node. The pseudo code of the algorithm is:

```
function FastestTimeElapsed(input M :
matrix of distance per average speed, I :
initial_node, T : finish_node) --> path
{ return a list of path that was chosen as
the fastest time needed to travel }
```

```
Declaration
S : array of integer
D : list of integer
i : integer
```

```
Algoritma
Initialization by set S[i] to 0.
while (T is not in D) do
    i <-- node in M that has the least
d/s
    S[i] <-- 1
    Insert i to D
endwhile
return D
```

With this algorithm we can look at the table of result that this algorithm choose:

Table 3

No	Current City	Chosen City	Next City	$\frac{d}{s_{avg}}$ minimum
1.	Bandung	Cirebon		4 hour
2.	Cirebon	Yogyakarta		4.2 hour
3.	Yogyakarta	Surakarta		1.5 hour
4.	Surakarta	Semarang		3.22 hour

From the table we can see that from Bandung, we have to go through Cirebon-Yogyakarta-Surakarta-Semarang. Total time to get from Bandung to Semarang with this algorithm is around 12.92 hour. This solution isn't the optimum solution because the optimum solution that found is Bandung through Cirebon than Semarang with time needed to get there is around 8.36 hour.

B. Using Dijkstra Algorithm

Pseudo code of this algorithm has already been added in chapter 2 on subchapter F. With the same problem of figure

3, we can get the table of result generated by djikstra algorithm. Matrix of time travelled in hour of figure 3 is:

Table 4

	1 (Jakarta)	2 (Bandung)	3 (Cirebon)	4 (Semarang)	5 (Yogyakarta)	6 (Surakarta)	7 (Surabaya)	8 (Malang)
1	0	6.75	8.175	∞	∞	∞	∞	∞
2	6.75	0	4.0	∞	7.46	∞	∞	∞
3	8.175	4.0	0	4.36	4.2	∞	∞	∞
4	∞	∞	4.36	0	3.63	3.23	6.15	∞
5	∞	7.46	4.2	3.63	0	1.5	∞	∞
6	∞	∞	∞	3.23	1.5	0	∞	9.25
7	∞	∞	∞	6.15	∞	∞	0	3.13
8	∞	∞	∞	∞	∞	9.25	3.13	0

The solution table generated by djikstra algorithm is

Node	Chosen Node	Path	S								D							
			1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Inisial	-	-	0	0	0	0	0	0	0	0	6.75	0	4.0	∞	7.46	∞	∞	∞
1	2	2	0	1	0	0	0	0	0	0	6.75	0	4.0	∞	7.46	∞	∞	∞
2	3	2, 3	0	1	1	0	0	0	0	0	6.75	0	4.0	8.36	7.46	∞	∞	∞
3	1	2, 1	1	1	1	0	0	0	0	0	6.75	0	4.0	8.36	7.46	∞	∞	∞
4	5	2, 5	1	1	1	0	1	0	0	0	6.75	0	4.0	8.36	7.46	8.96	∞	∞
5	4	2, 3, 4	1	1	1	1	1	0	0	0	6.75	0	4.0	8.36	7.46	8.96	14.51	∞
6	6	2, 5, 6	1	1	1	1	1	1	0	0	6.75	0	4.0	8.36	7.46	8.96	14.51	18.21
7	7	2, 3, 4, 7	1	1	1	1	1	1	1	0	6.75	0	4.0	8.36	7.46	8.96	14.51	17.64
8	8	2, 3, 4, 7, 8	1	1	1	1	1	1	1	0	6.75	0	4.0	8.36	7.46	8.96	14.51	17.64

From table above, we can get the optimum solution from Bandung to Semarang for 8.36 hour. Dijkstra algorithm can gives the optimum solution from the table and even the optimum solution from Bandung to another city.

V. CONCLUSION

Dijkstra algorithm can give the optimum solution better than the simple greedy such as greedy by distance/speed, greedy by distance, and greedy by speed. With knowing the optimum solution we can get path that takes the fastest time travels or the shortest time needed to travel from a city or initial city to another city.

VII. ACKNOWLEDGMENT

Author thanks to Allah SWT for the blessing so that author can finish this paper. Author also thanks to parents and friends who assisting author in the execution. Author also thanks to Mrs. Masayu Leylia Khodra and Mr. Rinaldi Munir. They have given me an example and literature of this subject.

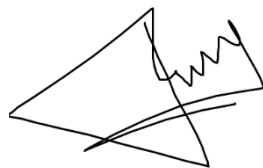
REFERENCES

- [1] Halliday, David and Robert Resnick, 2014, *Fundamental of Physics 10th Edition Extended*, New Jersey: John Wiley & Sons Inc., pp. 14 - 19.
- [2] Munir, Rinaldi, 2009, *Diktat Kuliah IF3051 Strategi Algoritma*, Bandung : Institut Teknologi Bandung, ch. 4 and 9.
- [3] Rachmatullah, Riva Syafri, 2013, *Penerapan Algoritma Koloni*

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Mei 2014

A handwritten signature in black ink, consisting of several overlapping loops and a jagged, zig-zagging line at the end, enclosed within a roughly triangular shape.

Riva Syafri Rachmatullah
13512036