Performance Analysis of Dijkstra, A* and Ant Algorithm for Finding Optimal Path Case Study: Surabaya City Map

Leo Willyanto Santoso, Alexander Setiawan, Andre K. Prajogo
Informatics Department, Faculty of Industrial Engineering
Petra Christian University
Jl. Siwalankerto 121-131 Surabaya, 60236
leow@petra.ac.id

1. Introduction

In the programming world there are so many algorithms that can be used to do an optimal path finding, for example Dijkstra, Ant and A* algorithm. However, there has been little work on the benchmarking in terms of the performance analysis of these algorithms [3, 7]. In this paper, we compare the performance of Dijkstra, Ant and A* algorithm to better know the characteristic of each algorithm when finding optimal path of certain route.

It would need the appropriate algorithm to search the optimal route, therefore, the purpose of this research is to explore what a good routing algorithm by comparing the 3 types of algorithms that can be used to solve the problem route search is: Ant algorithm, Dijkstra and A * in hopes of finding the best algorithm for searching a route.

The problems to be solved in this research are as follows:

1. How to implement an optimal routing algorithm in this application.
2. How to create a user friendly and easily understandable application.
3. How to set the constraints in this application.
4. How to implement an optimal routing algorithm on several goals at once.

The purpose of this research is to make an application to compare search algorithm routes between the three algorithms used in the search for optimal route so that the results of the third comparison of these algorithms can be determined which algorithms are suitable for searching the optimal route.

The remaining part of this paper is organized as follows. Section 2 presents an overview of current proposal for dealing with routing algorithm. Section 3 depicts the approach that we have delineated to solve the proposed problems. Moreover, the performance of proposed methods were discussed. Finally, section 4 concludes the paper.

2. Background

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) such that the sum of the weights of its constituent edges is minimized. An example is finding the quickest way to get from one location to another on a road map; in this case, the vertices represent locations and the edges represent segments of road and are weighted by the time needed to travel that segment [2, 6].

Multiple destination path finding problem is problem to find a solution path which must pass through several places at once. This problem can be solved by two approaches, brute force and heuristic [1].

2.1 Ant Colony Algorithm

This algorithm is aiming to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food.

The original idea comes from observing the exploitation of food resources among ants, in which ants' individually limited cognitive abilities have collectively been able to find the shortest path between a food source and the nest. The first ant finds the food source (F), via any way (a), then returns to the nest (N), leaving behind a trail pheromone (b). Ants indiscriminately follow four possible ways, but the strengthening of the runway makes it more attractive as the shortest route. Ants take the shortest route; long portions of other ways lose their trail pheromones [5].

- An ant will move from node i to node j with probability:

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \quad (1)$$

where,
$\tau_{i,j}$ is the amount of pheromone on edge i,j
α is a parameter to control the influence of $\tau_{i,j}$
$\eta_{i,j}$ is the desirability of edge i,j (a priori knowledge, typically $1/d_{i,j}$, where d is the distance).
β is a parameter to control the influence of $\eta_{i,j}$

- It must visit each city exactly once.
- Pheromone Update:

$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j}$
where
$\tau_{i,j}$ is the amount of pheromone on a given edge i,j
ρ is the rate of pheromone evaporation
$\Delta\tau_{i,j}$ is the amount of pheromone deposited, typically given by

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k & \text{if ant } k \text{ travels on edge } i,j \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where L_k is the cost of the kth ant's tour (typically length).

2.2 Dijkstra Algorithm

Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree.

In the following algorithm [4], the code u := vertex in Q with smallest dist[], searches for the vertex u in the vertex set Q that has the least dist[u] value. That vertex is removed from the set Q and returned to the user. dist_between(u, v) calculates the length between the two neighbor-nodes u and v. The variable *alt* on line 13 is the length of the path from the root node to the neighbor node v if it were to go through u. If this path is shorter than the current shortest path recorded for v, that current path is replaced with this *alt* path. The previous array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source.

```
 1  function Dijkstra(Graph, source):
 2      for each vertex v in Graph:           // Initializations
 3          dist[v] := infinity               // Unknown distance function from source to v
 4          previous[v] := undefined          // Previous node in optimal path from source
 5      dist[source] := 0                     // Distance from source to source
 6      Q := the set of all nodes in Graph
        // All nodes in the graph are unoptimized - thus are in Q
 7      while Q is not empty:                 // The main loop
 8          u := vertex in Q with smallest dist[]
 9          if dist[u] = infinity:
10              break
             // all remaining vertices are inaccessible from source
11          remove u from Q
12          for each neighbor v of u:         // where v has not yet been removed from Q.
13              alt := dist[u] + dist_between(u, v)
14              if alt < dist[v]:             // Relax (u,v,a)
15                  dist[v] := alt
16                  previous[v] := u
17      return dist[]
```

An upper bound of the running time of Dijkstra's algorithm on a graph with edges E and vertices V can be expressed as a function of $|E|$ and $|V|$ using the Big-O notation.

For any implementation of set Q the running time is $O(|E| \cdot dk_Q + |V| \cdot em_Q)$, where dk_Q and em_Q are times needed to perform decrease key and extract minimum operations in set Q, respectively.

The simplest implementation of the Dijkstra's algorithm stores vertices of set Q in an ordinary linked list or array, and extract minimum from Q is simply a linear search through all vertices in Q. In this case, the running time is $O(|V|^2 + |E|) = O(|V|^2)$. For sparse graphs, that is, graphs with far fewer than $O(|V|^2)$ edges, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of adjacency lists and using a binary heap, pairing heap, or Fibonacci heap as a priority queue to implement extracting minimum efficiently. With a binary heap, the algorithm requires $O((|E| + |V|)\log |V|)$ time (which is dominated by $O(|E| \log |V|)$, assuming the graph is connected), and the Fibonacci heap improves this to $O(|E| + |V| \log |V|)$.

2.3 Algoritma A* (A Star)

A* (pronounced "A star") is a computer algorithm that is widely used in path finding and graph traversal, the process of plotting an efficiently traversable path between points, called nodes. As A* traverses the graph, it follows a path of the lowest *known* path, keeping a sorted priority queue of alternate path segments along the way. If, at any point, a segment of the path being traversed has a higher cost than another encountered path segment, it abandons the higher-cost path segment and traverses the lower-cost path segment instead. This process continues until the goal is reached [8, 9].

The time complexity of A* depends on the heuristic. In the worst case, the number of nodes expanded is exponential in the length of the solution (the shortest path), but it is polynomial when the search space is a tree, there is a single goal state, and the heuristic function h meets the following condition:

$$|h(x) - h^*(x)| = O(\log h^*(x))$$

where h^* is the optimal heuristic, the exact cost to get from x to the goal. In other words, the error of h will not grow faster than the logarithm of the “perfect heuristic” h^* that returns the true distance from x to the goal

3. Implementation and Testing

In this section, discussed the use and testing of application. Testing process conducted by performing the test route search using three existing path finding algorithms with multiple destinations at once and by providing constraints. Applications have been tested on computers with Intel ® Core2Duo processor specifications ™ T5550@1.83 GHz with 2 GB of memory.

Using this application, users can directly search the route because the application has been entered the point - the crossing point and the points are already connected, however, the user must enter a first starting point and desired point to do a search and choosing one among the three algorithms that have been provided. Figure 1 shows entering the destination and choosing the algorithm.

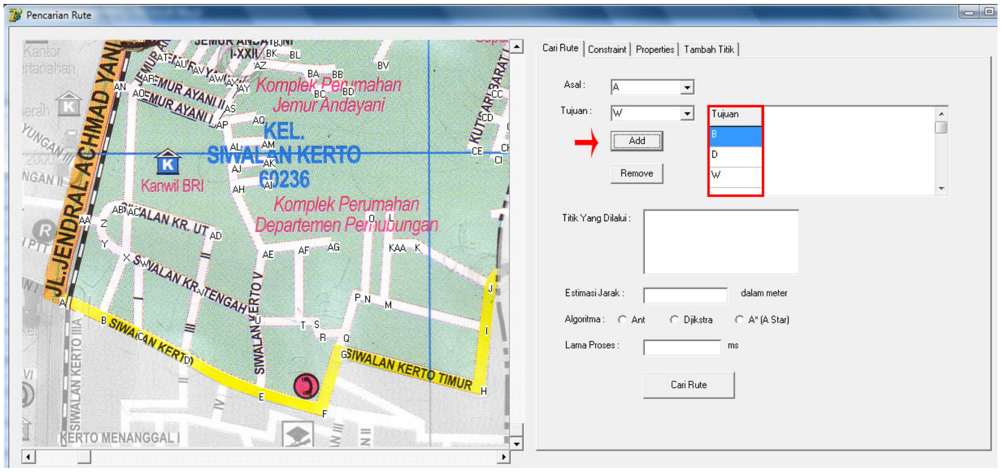

Figure 1: Entering Destination and Choosing Algorithm on Application

After entering the destination and choose the algorithm that is used then the user can press the search button after that will show the route through which the route, mileage and duration of the search process route. This can be seen in Figure 2.

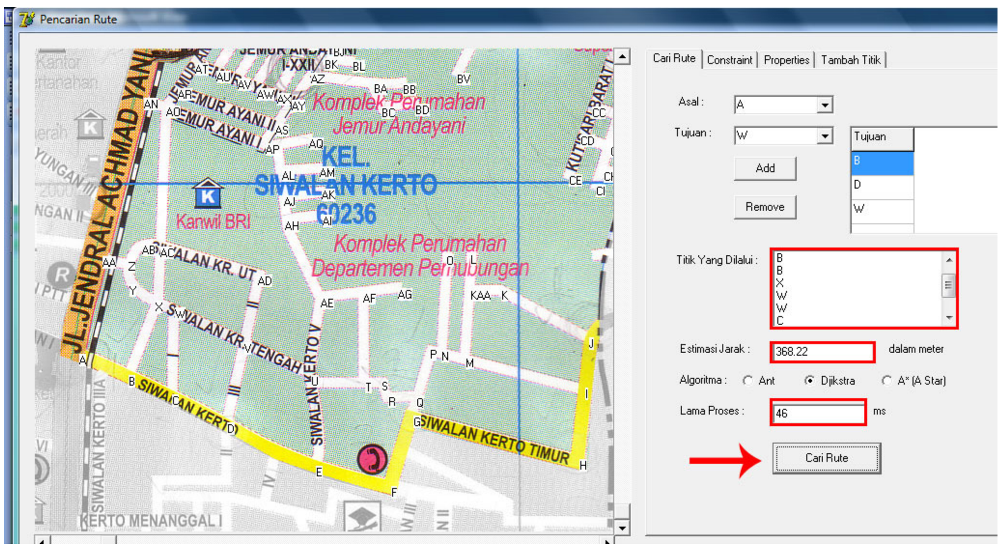

Figure 2: Route Search Result

Testing the application carried out by comparing the results of the 3 existing algorithms, the testing conducted are as follows:

- Testing the running time of algorithm
- Testing the correctness of the calculation in the program
- Testing mileage generated by the algorithm

There are two kinds of testing method of multiple destination on this application, heuristic methods and the Traveling Salesman Problem (TSP) where the heuristic method has better speed but with less accuracy. The test is done with destination " AE - U - E - G - BJ - AP ". Here are the test results and TSP heuristic method:

Table 1: Testing Dijkstra's algorithm with heuristic methods

No.	Point passed (with destination AE – U – E – G – BJ – AP)	Distance traveled (meter)	Time (ms)
1.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-	3007.61	140

	BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**		
2.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	141
3.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	141
4.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	140
5.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	140

Table 2: Testing A* Algorithm with Heuristic methods

No.	Point passed (with destination AE – U – E – G – BJ – AP)	Distance traveled (meter)	Time (ms)
1.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	109
2.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	109
3.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	109
4.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	124
5.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	94

Table 3: Testing Ant Algoritma Ant *with* Heuristic *methods*

No.	Point passed (with destination AE – U – E – G – BJ – AP)	Distance traveled (meter)	Time (ms)
1.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	14133
2.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	13946
3.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	13884
4.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	13604
5.	A-B-C-D–**E**-**U**-**AE**-AH-AJ-AL–**AP**-AS-AY-AZ-BK–**BJ**-BK-AZ-AY-AS-AP-AL-AJ-AH-AE-AF-T-S-R-Q-**G**	3007.61	13993

Figure 3: The output of *Heuristic* Method Testing

Table 4: The testing of *Dijkstra* Algorithm with TSP Method

No.	Point passed (with destination AE – U – E – G – BJ – AP)	Distance traveled (meter)	Time (ms)
1.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9454
2.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9485
3.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9453
4.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9484
5.	A A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9438

Table 5: Testing of A* Algorithm with TSP Method

No.	Point passed (with destination AE – U – E – G – BJ – AP)	Distance traveled (meter)	Time (ms)
1.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9313
2.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9328
3.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9298
4.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9327
5.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	9422

Table 6: Testing of Ant Algorithm with TSP Method

No.	Point passed (with destination AE – U – E – G – BJ – AP)	Distance traveled (meter)	Time (ms)
1.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	25155
2.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	26005
3.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	25990
4.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	25600
5.	A-B-C-D–**E**-F–**G**-Q-R-S-T–**U**-AH-AJ-AL–**AP**-AS-AY-AZ-BK-**BJ**	2260.96	26083

Figure 4: The output of TSP Method

From the test results can be concluded that the route search with heuristic methods can provide results faster but produces a much longer route, whereas when using the TSP method the time required to perform the search requires more time than the heuristic method, but produces a shorter route. However, there is also a condition in which the two methods produce similar results.

Our simulation show that the ant algorithm is not good enough to be used for path finding if compared to the Dijkstra and A* algorithm because lack of accuracy and stability and the duration for the process is far slower. However, under varying traffic conditions, Ant algorithm could adapts to the changing traffic and performs better than other shortest path algorithm. Moreover, the Dijkstra and A* algorithm could be developed more because the duration is still fast enough.

4. Conclusions

Based on the results of the implementation and testing program that has been done, it can be concluded that:

- Ant algorithm is not suitable for path finding algorithm because it is less stable and requires a long time to do a search.
- Dijkstra's algorithm and the algorithm A* provide optimal results in a fairly quick time.
- The more destinations you are looking for the longer process is needed.
- The constraint on the road does not affect the long process required. The constraint could affect the result of the search route depending on some conditions.
- Heuristic methods have a faster running time but sometimes gives a longer route, while the TSP method has a little longer running time but produces a much shorter routes than heuristic methods.
- Heuristic methods and TSP could provide the same results under certain conditions.

The emphasis of this paper was on feasibility – identification of possible approaches and development of methods to put them into practices. The evaluation of performance and the reliability of methods was proposed in this paper. Firstly, benchmarking for performance evaluation indicates for which method is the most efficient and effective from response time point of view. The next concern is the quality of the result.

References

[1] Champandard, Alex J. *How to Calculate Paths to Multiple Destinations*. Retrieved September 10, 2009 from http://www.aigamedev.com, 2003.

[2] Cormen, T.H., Leiserson, C.E., Rivest, R.L., & Stein, C. *Introduction to algorithms* (2nd ed.). Massachusetts: The MIT Press. 2001.

[3] Dariusz Król , Łukasz Popiela, Modelling Shortest Path Search Techniques by Colonies of Cooperating Agents, Proceedings of the 1st International Conference on Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems, October 05-07, 2009, WrocBaw, Poland, 2009.

[4] Dijkstra, E. W. "A note on two problems in connexion with graphs". *Numerische Mathematik* **1**: 269–271, 1959.

[5] Dorigo, M., Maniezzo,V. & Colorni, A., Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics–Part B, 26, 1-13*, 1996.

[6] Khan, Md. Mujibur Rahman. *Ant System to find the shortest path*. Paper presented at the 3rd International Conference on Electrical & Computer Engineering. Dhaka, Bangladesh, 2004.

[7] Lin Zhang , Li Xiaoping, The research and improvement of AntNet algorithm, Proceedings of the 2nd international Asia conference on Informatics in control, automation and robotics, p.505-508, March 06-07, 2010, Wuhan, China, 2010.

[8] Nillson, N.J. *Artificial intelligence: A new synthesis*. San Fransisco: Morgan Kauffman Publishers, 1998.

[9] Russel, S. & Norvig, P. (2003). *Artificial Intelligence: A modern approach* (2nd ed.). New Jersey: Prentice Hall, 2003.