# Introduction to Algorithm: Shortest path problem simulation and analysis

**Yung-Chi Liu,** A20364639,IIT, Chicago, IL  
[yliu287@hawk.iit.edu](mailto:yliu287@hawk.iit.edu) **Chih-Feng Yu** ,A20314374,IIT,Chicago,IL  
[cyu25@hawk.iit.edu](mailto:cyu25@hawk.iit.edu)

**ABSTRACT** Shortest path problem is a very popular and common problem in graph theory. The shortest path can be defined as various types of graph such as directed, undirected and mixed. In our simulation test, first, we will use three common algorithms of directed single path to compare: **Dijkstra’s, Bellman-Ford, Label Setting**. Second, we will create the graph in two different data structures (**Adjacent matrix** and **Dictionary)** via assigning different scale of nodes **{250, 500, 1000, 2500, 5000, 7500, 10000}**. Third, in the weight of distance scale, we use {1,2,3,4,5,6,7,8,9,10,INF,INF,INF,INF,INF,INF,INF,INF,INF,INF} to randomly distribute to the edges between nodes. In addition, the edge which is assigned an INF weight which means the two nodes do not connect to each other and get the infinity weight.

In our analysis, we plan to test and analyze 6 permutations of 3 algorithms with 2 data structures. The content will include tables with data of time counting results and figures of comparisons. Last, the conclusion and reference will be drew at the end.

**INDEX TERMS** Shortest path problem, Dijkstra’s, Bellman-Ford, Label Setting

# [[1]](#endnote-1) INTRODUCTION

**Motivation**

1. The reason we choose **dictionary** and **matrix** data structure is because they are similar but the operation might cause various results. Since matrix structure will travel all the indices to check and calculate the V-to-V path, dictionary only stores the adjacent nodes of edge weight. Therefore, we assume the *dictionary* is **faster** than the *matrix* to these three algorithm.
2. We choose **{250, 500, 1000, 2500, 5000, 7500, 10000}** nodes assigned to the scale of graph, because it will cause memory error when we try to run over 10000 nodes. Therefore, we reduce the testing input scale of nodes.
3. Weight values list {1,2,3,4,5,6,7,8,9,10,INF,INF,INF,INF,INF,INF,INF,INF,INF,INF}. It doesn’t matter if we choose from 1 to 10 or 1 to 100 for weighted edges. However, the amount of infinity weights will cause the complexity of graph which means a node connect few nodes if it has a lot of infinity edges and vice versa. Therefore, we fix 50% of infinity values that a node might connect to half nodes and it will keep the graph not so awkward.

**Scheme of Algorithm**

1. **Initializing Graph:**

Vertex v , W = set of weight

Dic {{}} , M [v][v]

While I = 0-> v

While j=0-> v

If I = j, { M[i][j] = 0 , Dic[i][j] = 0 }

Else,

M[i][j] = random(W)

If (M[i][j] != INF) //Dictionary don’t need to store Infinity edge

Dic[i][j] = M[i][j]

Return(M,Dic)

1. **Relax (A, B):**

If d (A) + weight (A, B) < d(B)

d(B) <- d (A) + weight (A, B)

1. **Dijkstra's Algorithm:**

================================================

Graph G

Set distance(start) = 0

While 0-> number of vertices //O(v)

Find vertex A //Find a minimum vertex which is not visited. O(v)

Mark vertex A as visited.

For each B connect to A //O(v)

Do Relax (A, B) //B is a node which hasn’t been visited

================================================

**Expected time complexity: O(V)\*O(V+V) = O(2V^2) = O(V^2)**

1. **Bellman-Ford Algorithm:**

================================================

Graph G

Set distance(start) = 0

While 0->(V-1) //(Shortest Path had V-1 edges). O(v)

Do for each edge (A, B) //O(V^2)

Do Relax (A, B)

================================================

**Expected time Complexity: O(V^2) \*O(V) = O(V^3)**

1. **Label Setting Algorithm:**

================================================

Graph G

Set distance(start) = 0.

Start -> visited

SPT (Shortest Path Tree) <- Start

While 0 -> V //O(V)

Travel all Vertex in SPT //O(V)

travel all vertex adjacent to SPT //O(V)

DO Find a Vertex B (minimum weight)

Set A, B and weight

Set distance(B)

Mark B visited

================================================

**Expected time Complexity: O(V)\* O(V)\* O(V) = O(V^3)**

# Results and Analysis

There are two main sections of comparing the results: **comparing two different data structure in each algorithm** and **comparing all algorithm’s time consumption in each data structure**. In the section, each algorithm will show the result table of time consumption related to the growth of vertices. The column of “Big-O” is the expected time growth rate increasing from the basic case (which is vertices = 250). Moreover, the figure of “time growth rate comparison” will show the trend of growth rate helping to figure out the assumption and analysis whether is correct or not.

In the second section is comparing all algorithm’s time consumption in each data structure. It will show the performances of three algorithms in each data structure. More figures will be shown to present the results of comparison.

1. **Comparison of different data structure in each algorithm**

* Dijkstra matrix vs dictionary, with Big-O



**Table 1. Dijkstra’s time consuming with different data structure**

According to **Table 1.** , at the same nodes of graph, the results of dictionary structure are all faster than using the matrix data structure. We also can notice the time consumption of Matrix will grow rapidly than dictionary as the number of vertices growing. Next, we calculate the expected big-O time growing rate to compare the operating time growth rate of two data structures, and we can see the expected time complexity is matched the result of running the dictionary (Shown on **Figure 1.**). Therefore, in general, we assume using the dictionary data structure is faster than matrix (Shown on **Figure 2.**).

**Figure 1. Dijkstra’s time growth rate comparison**

**Figure 2. Time Comparison of Dijkstra’s with two different data structure**

* Bellman-Ford matrix vs dictionary, with Big-O



**Table 2. Bellman-Ford time consuming with different data structure**

In **Table 2**, it shows a different result of Bellman-Ford. First, we expect both data structure could be O (V^3). However, the time complexity of Bellman-Ford by using dictionary is O (V^2) which is faster than using the matrix (Shown on **Figure 4**). It’s interesting that the expected big-O time growing rate is close to the result of using matrix data structure (Shown on **Figure 3**). Therefore, we can realize that if choose appropriate data structure to store the graph data, it will improve the performance of Bellman-Ford a lot.

**Figure 3. Bellman-Ford’s time growth rate comparison**

**Figure 4. Time Comparison of Bellman-Ford with two different data structures**

* Label Setting matrix vs dictionary, with Big-O



**Table 3. Label Setting time consuming with different data structure**

**Figure 5. Label setting’s time growth rate comparison**

**Figure 6. Time Comparison of Label setting with two different data structures**

1. **Comparison of each algorithm**

**Figure 7. Time Comparison of three algorithms in matrix data structure**

**(Note: Over 5000 nodes, Bellman-ford operation will cause memory error)**

According to **Figure 7**, we can notice that Dijkstra is fastest. Label setting grows gradually but Bellman-Ford grows very rapidly and even caused memory error. However, the time complexity of Bellman-Ford and Label Setting are O (V^3). Why Bellman-ford is the slowest? Because Bellman-Ford will update all the non-visited nodes’ distance. But Label Setting will choose the smallest distance of non-visited node to add to the shortest path tree, hence, it will update only one node.

**Figure 8. Time Comparison of three algorithms in dictionary data structure**

**(Note: Over 5000 nodes, Bellman-ford operation will cause memory error)**

As the **Figure 8** shown, Bellman-Ford is almost same as Dijkstra’s that’s because their time complexity is O (V^2). However, Label setting seems remain the same outcome as using matrix data structure. It’s obvious that Bellman-Ford improves the performance a lot.

# Conclusion

As the testing result, it is obviously that use dictionary in data structure is faster than use matrix in data structure. In our opinion, one reason is based on data structure’s characteristic. When search minimum weight edge in matrix, it travels all vertices even with distance is infinite. However, this not happened when search in dictionary, because dictionary only had distance with adjacent vertices.

By using matrix as data structure, the fastest one is Dijkstra, come with Label Setting and the slowest one is Bellman-Ford, all there three algorithms are ordered by speed. By using dictionary as data structure, the fastest one is Dijkstra, come with Bellman-Ford and the slowest one is Label Setting.

# Improvement and Future Work

First we could add more single path algorithms, like: Dial's Algorithm and Gabow's Algorithm to support our observation and analysis.

Make more test case with different graph complexity. In this project, we set 50% of probability that a node will not connect to half of nodes. Furthermore, we can try 25% or 75% to do test.

Last, as shortest path algorithm use matrix in data structure, in order to make program run faster, is doing the program by parallel. In a N\*N matrix, we find minimum weight in each row which has N elements. By Shared Memory Parallel method, each processor handles N/k elements with k processors, and merges the result. This could Speed Up the program.

# Reference

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001).

"Dijkstra's algorithm". Introduction to Algorithms (Second ed.)

Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E. (April 1990).

"Faster Algorithms for the Shortest Path Problem". Journal of Association for Computing Machinery

Bellman, Richard (1958).

"On a routing problem". Quarterly of Applied Mathematics

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.

"The Bellman–Ford algorithm". Introduction to Algorithms.

Heineman, George T.; Pollice, Gary; Selkow, Stanley (2008).

"Chapter 6: Graph Algorithms". Algorithms in a Nutshell. O'Reilly Media.

Kroger, Martin (2005).

"Shortest multiple disconnected path for the analysis of entanglements in two- and three-dimensional polymeric systems". Computer Physics Communications

Michel Gondran; Michel Minoux (2008).

"Graphs, Dioids and Semirings: New Models and Algorithms". Springer Science & Business Media.

Nathaniel Fan.

“Dijkstra's Algorithm”, https://www.youtube.com/watch?v=gdmfOwyQlcI

Prof. Erik Demaine

“Introduction to Algorithms: Lecture 15: Shortest Paths II”, http://courses.csail.mit.edu/6.006/spring11/lectures/lec15.pdf

Bjørn Petersen(2006)

“Label-Setting Algorithm for Shortest Path Problems”

http://www.diku.dk/OLD/undervisning/2006-2007/2006-2007\_b2\_426/slides3.pdf

1. [↑](#endnote-ref-1)