IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 20, Issue 2, Ver. II (Mar. - Apr. 2018), PP 10-15
www.iosrjournals.org

Review and Analysis of Single-Source Shortest Path Problem Using Dijkstra's Algorithm

Jogamohan Medak[1] Partha Pratim Gogoi[2]
[1]*Assistant Professor, North Lakhimpur College (Autonomous), North lakhimpur, Assam, India.*
[2]*Master of Computer Applications, Tezpur university, Assam, India.*
Corresponding Author: Jogamohan Medak

Abstract: *To minimize the cost of routing in computer networks, the shortest path problem is greatly used in order to find the minimum distance path from source to destination. The shortest path problem is sometimes called the min-delay path problem. The main objective of this paper is to evaluate and discuss the general aspect of Dijkstra's Algorithm in solving the shortest path problem. Explanation and evaluation of the algorithm is illustrated in graphical forms in order to exhibit the functionality of the algorithm.*

Keywords: *Routing, Computer Networks, Min-Delay Path Problem, Dijkstra's Algorithm*

Date of Submission: 05-03-2018 Date of acceptance: 24-03-2018

I. Introduction

The shortest path problem consists in finding a path between two vertices (or nodes) in a graph such that the amount of the weights of its integral edges is minimized. Graphs are mathematical arrangements used to construct pairwise relations between objects. In this scenario, a graph is made up of vertices or points which are interlinked by edges. In shortest path problems, a directed or undirected graph is given with a weight for each edge and we have to find a shortest path i.e. a path with the smallest possible weight between one or more pair of vertices. We can experience the following three possible cases: Firstly, the second node is not accessible from the first; where we can consider the shortest path between the two has the length ∞. Secondly, the second point is accessible from the first point, i.e. there is a path, but may not be the shortest. Thirdly, there is a shortest path between the first and second point. The shortest path algorithm will evaluate the whole graph stepwise and identify all the above cases and eventually, will able to find the shortest path between the two points.

The Single-source shortest path problem states that in a weighted digraph, from a designated vertex s, we have to determine the shortest path from s to any other vertices. Here, an approximation is made from a designated vertex to any other vertices, which are steadily substituted by more precise values until the shortest distance is attained. The approximation is always a miscalculation of the true distance and is replaced by minimum of its old value with the length of a newly discovered route.

II. Dijkstra's Algorithm

Dijkstra's algorithm is a greedy algorithm for solving single source shortest path problem that provides us with the shortest path from one particular source node to all other nodes in a given graph. This algorithm can be applied for both directed and undirected graph, but graph must be connected and no negative weight must exist. In this case, the algorithm is quite efficient. Since there are no non-negative weight cycles, there will be a shortest path whenever there is a path.

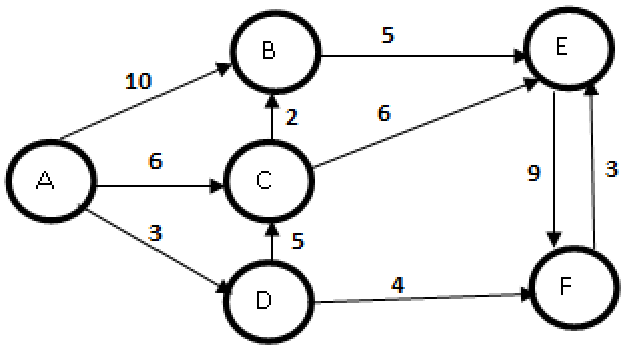
The steps to find shortest path using Dijkstra's algorithm are as follows:

1. Start at the source node by labelling it with a distance of 0 and all other nodes as ∞. The source node is treated as current node and considered as visited. All other nodes are considered as unvisited.
2. Identify all the unvisited nodes that are presently connected to the current node. Calculate the distance from the unvisited nodes to the current node by adding the weights of their edges.
3. Label each of the vertices with their corresponding weight to the current node, but only alter the label of a node, if it is less than the previous value of the label. Each time, the nodes are labelled with its weights; keep track of the path with the smallest weight.
4. Mark the current node as visited by coloring over it. Once a vertex is visited, we not need to look at it again.
5. From all the unvisited nodes, find out the node which has minimum weight to the current node, consider this node as visited and treat it as the current working node.
6. Repeat steps 2, 3 and 4 until all nodes are visited.

Algorithm:
/* S= set of visited node, Q= Queue, G=Graph, w=Weight */

```
Dijkstra (G, w, S)
{
        Initialize_single_soure(G,S)
        S ←Ø                                  // the set of visited nodes is initially empty
        Q← v[G]                               // The queue is initially contain all nodes
        While (Q≠ Ø)                          // Queue not empty
                do u← extract_min(Q)      // select the minimum distance of Q
                        S ←SU{u}              // the u is add in visited set S
                For each vertex v ϵ adj(u)
                        do relax(u, v, w)
}
Initialize_single_soure (G, S)
{
For each vertex v ϵ adj (u)
                d [v] ←∞                  //Unknown Distance from source node
π[v] ←nil                                 // Predecessor node initially nil.
d[s] ←0                           //Distance of source nodes is zero
}
Relax (u, v, w)
{
 If d[v]>d[u]+w(u,v)                      // comparing new distance with existing value
        {
                d[v] ← d[u]+w[u,v]
                π[u] ←u
        }
}
```

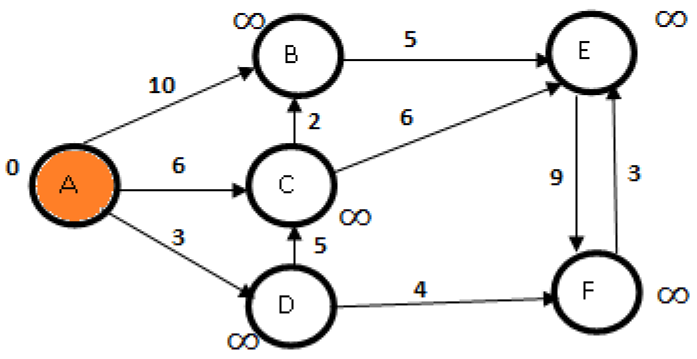
Using Dijkstra's algorithm, calculation of the shortest distance from the source node's to all other nodes is given in the following graph below.



$$\begin{tikzcd}
 & B \arrow[rr, "5"] & & E \arrow[d, shift right, "9"'] \\
A \arrow[ur, "10"] \arrow[r, "6"] \arrow[dr, "3"'] & C \arrow[u, "2"] \arrow[urr, "6"] & & \\
 & D \arrow[u, "5"] \arrow[rr, "4"] & & F \arrow[u, shift right, "3"']
\end{tikzcd}$$

Step1: Consider the source node as A. Therefore, A will be the current working node and can be treated as visited.

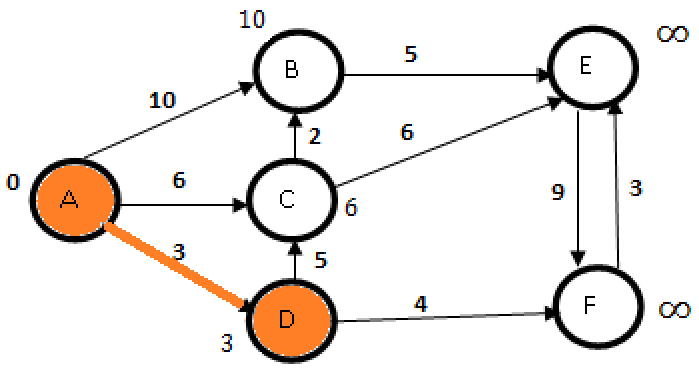
Nodes	A	B	C	D	E	F
Distance d[v]	0	∞	∞	∞	∞	∞
Distance from π[v]	A					
Status	visited	unvisited	unvisited	unvisited	unvisited	unvisited



Step2:
We can calculate the weights of unvisited adjacent nodes of current node A. The unvisited nodes of A are B, C, and D.
Now, Distance [B]> distance [A] + Weight (A, B) ⇨ ∞ > 0 +10 = 10
So, alternation of value is required in node B.
Distance [C]> distance [A] + Weight (A, C) ⇨ ∞ > 0 + 6 = 6
So, alternation of value is required in node C.
Distance [D]> distance [A] + Weight (A, D) ⇨∞ > 0 + 3 = 3
So, alternation of value is required in node D.

Nodes	A	B	C	D	E	F
Distance d[v]	0	10	6	3	∞	∞
Distance from π[v]	A	A	A	A		
Status	visited	unvisited	unvisited	unvisited	Unvisited	unvisited

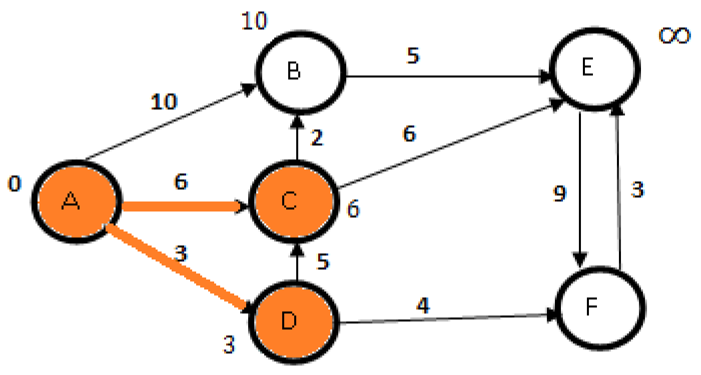
We can discover the smallest distance from A to the unvisited node which is D, traversing through A to D and now D can be considered as the current node.



Step3:
We can calculate the weights of unvisited adjacent nodes of current node D. The unvisited nodes of D are C and F.
Now,
Distance[C] < Distance [D] + Weight (D, C) ⇨ 6 < 3 + 5 = 11, so no change is required.
Distance [F] > Distance [D] + Weight (D, F) ⇨ ∞ > 3 + 4 = 7
So, alternation of value is required in node F.

Nodes	A	B	C	D	E	F
Distanced[v]	0	10	6	3	∞	7
Distance from π[v]	A	A	A	A		D
Status	visited	unvisited	unvisited	visited	unvisited	unvisited

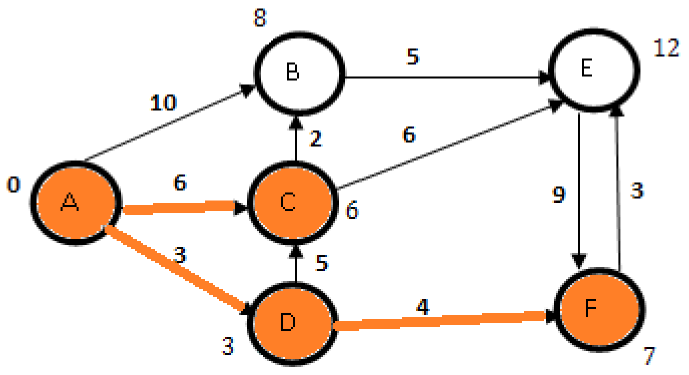
We can discover the smallest distance from A to the unvisited node which is C, traversing through A to C and now C can be considered as the current node.



Step 5:
Unvisited adjacent of current working node of C is B, E
Now,
Distance [B]> Distance[C] + Weight (C, B) ⇨ 10 > 6 + 2 = 8
So, alternation of value is required in node B.
Distance [E]> Distance[C] + weight (C, E) ⇨ ∞ > 6 + 6 = 12
So, alternation of value is required in node E.

Nodes	A	B	C	D	E	F
Distance d[v]	0	8	6	3	12	7
Distance from π[v]	A	C	A	A	C	D
Status	visited	unvisited	visited	visited	Unvisited	unvisited

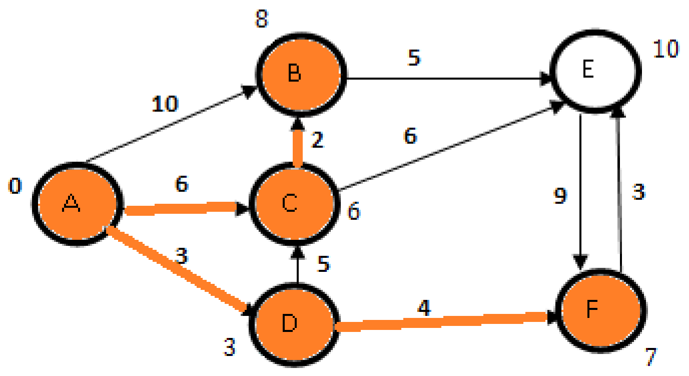
We can discover the smallest distance from A to the unvisited node which is F, traversing from A to F through D and now F can be considered as the current node.



Step 5:
Unvisited current node of F is E.
Now,
Distance [E]> Distance [F] + Weight (F, E) ⇨ 12 > 7 + 3 = 10
So, alternation of value is required in node E.

Nodes	A	B	C	D	E	F
Distance d[v]	0	8	6	3	10	7
Distance from π[u]	A	C	A	A	F	D
Status	visited	unvisited	visited	visited	Unvisited	visited

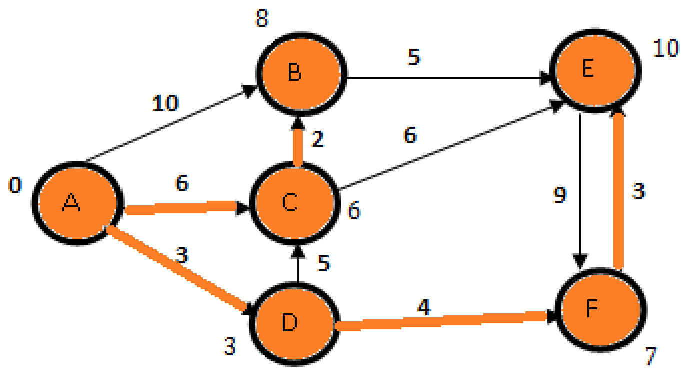
We can discover the smallest distance from A to the unvisited node which is B, traversing from A to B through C and now B can be considered as the current node.



Step6:
From current node B the unvisited node is E.
Distance [E] < Distance [B] + Weight (B, C), so no change is required.

Nodes	A	B	C	D	E	F
Distance d[v]	0	8	6	3	10	7
Distance from π[v]	A	C	A	A	F	D
Status	visited	Visited	visited	visited	unvisited	visited

Now, only unvisited node is E. So, Traverse from F to E and mark E as visited.



The final table is:

Nodes	A	B	C	D	E	F
Distance d[v]	0	8	6	3	10	7
Distance from π[v]	A	C	A	A	F	D
Status	visited	visited	visited	visited	visited	visited

Therefore, the shortest path from A to all other nodes is given below:

A to B	is	A→C→B
A to C	is	A→C
A to D	is	A→D
A to E	is	A→D→F→E
A to F	is	A→D→F

III. Complexity

Finding the minimum distance is O (n) where n is the number of nodes and overall complexity with adjacency list representation is O (n^2 + e) where e is the number if edges i.e. O (n^2). If queue is kept as a binary heap, relax will need a decrease-key operation which is O (log n) and the overall complexity is O (n log n + e log n) i.e. O (e log n). If queue is kept as a Fibonacci heap, decrease-key has an amortized complexity O (1) and the overall complexity is O (n log n + e). For dense graphs, ($e \approx n^2$) Fibonacci heap and arrays are better than binary heap. For sparse graphs $e << n^2$, binary heap is better than array but Fibonacci heap can be considered as the best.

IV. Conclusion

The time complexity for Dijkstra’s algorithm is acceptable in terms of its overall performance in solving the shortest path problem that produces only one solution. It is used in IP routing to find Open shortest Path First and link state routing protocol. However, it has some major drawbacks such as it cannot handle negative edges and it leads to acyclic graphs and most often cannot obtain the right shortest path. This algorithm is highly centralized, not very distributed which make it very irrelevant for modern internet routing. But its drawbacks are taken care by some other shortest path algorithm like Bellman–Ford algorithm and Floyd-Warshall Algorithm.

Reference

[1]. Srivastava, S.K, Srivastava, Deepali.(2011). Data Structures Through C in Depth. BPB publications.
[2]. Baluja, G.S.(2008). Data Structures Through C. Dhanpat Rai & Co.(Pvt) Ltd.
[3]. Kairanbay, M., Jani, H. M.(2013). A Review And Evaluations Of Shortest Path Algorithms. international journal of scientific & technology research volume 2, issue 6.
[4]. Ojekudo, N.A., Akpan, N. P.(2017). Anapplication of Dijkstra’s Algorithm to shortest route problem. Volume 13, Issue 3, PP 20-32.
[5]. Ahmat, K. A. (n.d.). Graph Theory and Optimization Problems for Very Large Networks. New York: City University of New York/Information Technology.
[6]. Brendan, H. a. (2005). Generalizing Dijkstra's Algorithm and Gaussian Elimination for solving MDPs. International Conference on Automated Planning and Scheduling/Artificial Intelligence Planning System, (pp. 151 - 160).
[7]. Chen, K. M. (2009). A real-time wireless route guidance system for urban traffic management and its performance evaluation. Papers of the 70th Vehicular Technology Conference Anchorage VTC, (pp. 1 -5).
[8]. Ebrahimnejad, S. A. (2011). Find the Shortest Path in Dynamic Network using Labelling Algorithm. Journal of Business and Social Science.
[9]. Goldberg, A. V. (n.d.). Point - to - Point Shortest Path Algorithms with Pre-processing. Silicon Valley:MicrosoftResearch.
[10]. Goyal, S. (n.d.). A Survey on Travelling Salesman Problem. North Dakota: Department of Computer, University of North Dakota.
[11]. Lee, D. C. (2006). Proof of a modified Dijkstra's algorithm for computing shortest bundle delay in networks with deterministically time-varying links. IEEE Communications Letters, 734 -736.
[12]. Dijkstra's algorithm.(2018). Available at https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.
[13]. Greedy Algorithms.(2018). Available at https://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/
[14]. Dijkstra's Shortest Path Algorithm.(2018). Available at https://www.youtube.com/watch?v=FlrysJdMcdc&t=1467s
[15]. Dijkstra's algorithm.[2018]. Available at https://www.youtube.com/watch?v=wt5cqvfdyxg

IOSR Journal of Computer Engineering (IOSR-JCE) is UGC approved Journal with Sl. No. 5019, Journal no. 49102.

Jogamohan Medak. " Review and Analysis of Single-Source Shortest Path Problem Using Dijkstra's Algorithm." IOSR Journal of Computer Engineering (IOSR-JCE) 20.2 (2018): 10-15.