REDUCED SOLUTION SET SHORTEST PATH PROBLEM: CAPTON ALGORITM WITH SPECIAL REFERENCE TO DIJKSTRA'S ALGORITHM

Qaiser Abbas[1], Qasim Hussain[2], Tehseen Zia[3], and Arfan Mansoor[4]

[1,2,3]Department of Computer Science & Information Technology, University of Sargodha, 40100, Sargodha, PK
[4]Institut für Informationstechnik, Technische Universität Ilmenau, Ilmenau, Thuringia, DEU

Email: qaiser.abbas[1], tehseen.zia@uos.edu.pk[3], qasimfsc@gmail.com[2], arfan.mansoor@tu-ilmenau.de[4]

DOI: https://doi.org/10.22452/mjcs.vol31no3.1

ABSTRACT

To find the shortest path between the nodes of a graph, different algorithms like Bellman-Ford, Dijkstra, Floyd-Warshall and Johnson exist. However, in this paper, the issue of shortest path problem with special reference to Dijkstra's algorithm is presented. An idea of shortlisting the appropriate nodes in a graph is proposed and presented, which is then used to find the shortest path with the help of Dijkstra's algorithm. This complete work - named Capton algorithm- provides a solution to single source shortest path problem with minimized time complexity as compared to Dijkstra's algorithm.

Keywords: Shortest Path Problem, Reduction Factor, Reduce Solution Set, Capton Algorithm, Heuristic Distance, Dijkstra's Algorithm.

1.0 INTRODUCTION

Shortest path problem is the well-known problem in graph theory, in which a path between two nodes or vertices is calculated in such a way that the sum of the weights of its constituent edges is minimized. To find the shortest path between the nodes of a graph, different algorithms like Bellman-Ford [1], Dijkstra [2], Floyd-Warshall [3], and Johnson [4] are frequently used. The first two algorithms are best for single source shortest path problem, the third is mostly used for dense graph to find all-pairs shortest path, while the last is considered more efficient for sparse graphs. These algorithms explore the whole graph and calculate the shortest path from each node, which is a time consuming process. Their respective time complexities are O(VE), O(E+VlogV), O(V^3) and O(V^2log V + VE) where V is the set of vertices and E is the set of edges in their respective graphs. These time complexities can be minimized by implementing an idea, proposed in this paper. According to this idea, the problem area in a given graph can be reduced through the short-listing of appropriate nodes that can participate in problem solving.

Shortest path problem is the problem of directed, undirected and mixed graphs. In this paper, a new solution is proposed that is by putting the x and y positions of vertices and their weights. These values are normally available on World maps, which Google is also using to calculate the coordinates in the Google Maps Distance Matrix API[1]. Irrespective of these World maps, if these values are not available, then we have to devise the scale first on the problem space like a kid does in geometry class on the graph paper. After that, we will be able to get the x and y positions of vertices/nodes and then their weights accordingly. Anyhow, an ideal centerline is selected and heuristic distance with respect to centerline is calculated for each node. Only those vertices are visited and extracted from the whole graph, which have the probability to be near and lie on the centerline. In this way, there is no need to visit all the vertices of a graph. This centerline is assumed as the shortest path from the source to the destination vertex.

Section 2.0 presents the literature review, starting with the classic, non-English literature and then coming to the technique of depth first search. Section 2.1 details the invention of matrices and how they started to be used in calculation of shortest path in networks, and Section 2.2 briefs the major shortest path discussions with the introduction of Dijkstra's and Bell-man Ford's methods for calculating the shortest path in graphs. Afterwards, Section 3.0 discusses the proposed augmented algorithm in two steps. In Section 3.1, a theoretical and numerical analysis of vertices shortlisting procedure is presented, while Section 3.2 details the respective algorithms along

[1] https://developers.google.com/maps/documentation/distance-matrix/intro

with the explanation of solution achievement using short queue method. Similarly, Section 4.0 analyzes the complexity of the final augmented algorithm with a claim that it has the better running time and memory management as compared to the Dijkstra's version.

The running example of the proposed Capton algorithm is elaborated in Section 5.0 with shortlisting of graph vertices first (Section 5.1) and then finding solution from the shortlisted vertices (Section 5.2). Merits and demerits of this experimentation are listed in Section 6.0 and then a comparative study is presented in Section 7.0. Sparse and dense graphs have an effect on area reduction factor. The graphs with larger number of vertices are a better source to provide optimum solutions. The Capton algorithm has an ability to increase the problem area from smaller to larger ones. The details can be seen in Section 8.0, and then this work is finally concluded in Section 9.0.

2.0 EVOLUTION OF SHORTEST PATH PROBLEM

To opt for shortest routes is common in daily life and the evolution of this idea cannot be determined (i.e. when this idea was first practiced is hard to track). To follow the shortest path in some circumstances is not only true for humans but also true for other species. However, in contrast to other combinatorial optimization problems, for example prime numbers [5], shortest spanning tree, assignment and transportation, etc., the work on shortest path problem in mathematical perspective was started quite late. Searching a path in maze puzzle fits to the classical graph problems, which were discussed by Wiener in 1873 [6], Lucas in 1882 [7], and Tarry in 1895 [8]. This literature was not in English, for which, later on, Biggs, Lloyd, and Wilson in [9] came forward in 1976. They translated the mentioned literature into English. After which, it was known to us that this is the literature that placed the foundation for the depth-first search techniques for the first time.

2.1 Matrix for Evaluation of Shortest Path

In 1946 to 1953, matrix methods were devised to evaluate network relations. For example, matrix transitive closure relation is used in networks to identify the pairs of points (s, t) in a directed graph such that t is reachable through s. Graphs can be represented by matrices like adjacency matrix. Similarly, by taking matrix products iteratively, one can calculate the transitive closure. These kinds of matrix operations are detailed in [10-15], which were used to evaluate the shortest path for the first time.

2.2 Shortest Distance Paths

For a given directed graph $G = (V, E)$, distances and shortest paths among the vertices can be calculated. Bellman-Ford and Dijkstra methods are the two popular approaches used for this purpose. Dijkstra's method has faster running time with nonnegative weights, while the Bellman-Ford's method needs only a single prevention of not having a cycle with negative length. The weight or the distance is the value of the edge connecting two vertices.
Both the methods have the same mechanism as discussed in [16]. After initializing the source vertex s with zero in given graph, the other vertices v are initialized with infinity. After that, for any pair (u, v), a condition $d(v) > d(u) + l(u, v)$ is checked, where d is used for distance, v is the vertex other than the source s, and l is the length or weight on the edge between the two vertices (u, v). If the condition holds, then the values are updated as seen in the following steps.

(1). $d(s) := 0$ *and* $d(v) := \infty$ *for each* $v \neq s$.
(2). *If* $d(v) > d(u) + l(u, v)$ *then* $d(v) := d(u) + l(u, v)$

3.0 PROPOSED METHODS

Our proposed Capton algorithm presented in Section 3.2 solves the problem in two parts: the first part is about shortlisting of the graph nodes/vertices (line 2 to 8), and the second part (line 9 to end) is to find the solution using the short queue presented in the Capton algorithm. Each part is discussed as follows:

3.1 Shortlisting of Vertices

In this part, the problem area is reduced using V.x and V.y values of the vertices using the following steps:

1. A straight line (centerline) given in Equation 1 is plotted first among the source s and destination d vertices of a graph.

   LE = $Ax + By + C = 0$ (1)
   Where A, B and C are coefficients and each vertex in vertices set V contains x and y coordinates. Equation 1 can be solved using its detailed version as follows:

   $(d_y – s_y)x – (d_x – s_x)y + (d_x s_y – s_x d_y) = 0$

   Where,
   A = $(d_y – s_y)$,
   B = $(d_x – s_x)$ and
   C = $(d_x s_y – s_x d_y)$

   Here x and y are the two coordinates mentioned, while s and d are the source and destination vertices as shown in the Fig. 1. The values of H.A, H.B, and H.C presented in algorithm *Center-line (s, d)* in Section 3.2 are calculated using these discussed A, B, and C. The values of H.A, H.B, and H.C are used in other algorithm *SHORT-QUEUE (G, W, H)* which will be used to calculate the actual heuristics distance of vertices.

2. An ideal heuristic distance HD based on Pythagorean theorem [17] is calculated from the source s(x, y) to destination d(x, y) vertices using the following Equation 2.

$$\text{HD} = \sqrt{(d_x - s_x)^2 + (d_y - s_y)^2} \quad (2)$$

   This is used to calculate the distance between the two vertices s and d, and presented in algorithm *Center-line(s, d)*.

3. Set the distance D=HD/RF, where D, presented in Fig. 1, is the distance of the parallel selection lines from the centerline that are used to shortlist the nodes, and RF is the area reduction factor that reduces the problem area with respect to HD, also given in the main Capton algorithm (See Section 3.2) at line 9.

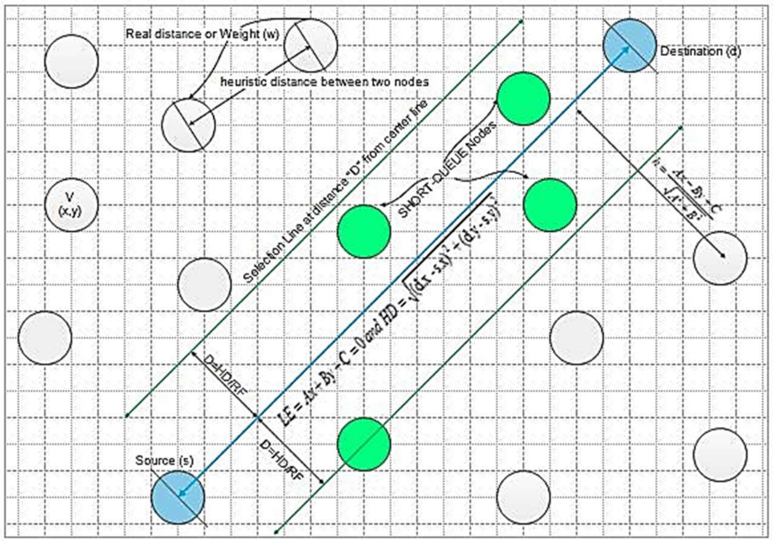
Real distance or Weight (w)
heuristic distance between two nodes
Destination (d)
V (x,y)
Selection Line at distance "D" from center line
SHORT-QUEUE Nodes
$h = \frac{-Ax - By - C}{\sqrt{A^2 + B^2}}$
$LE = Ax + By + C = 0$ and $HD = \sqrt{(dx - sx)^2 + (dy - sy)^2}$
D=HD/RF
D=HD/RF
Source (s)

Fig.1: Basic Concept of Capton Algorithm

4. Calculate another heuristic distance h from the centerline for each node presented in Fig.1. If h≤D then select that node and put it into a short queue SQ, otherwise search for other nodes to store in the short queue. For calculation of h, we can use the following Equation 3.

$$h = \frac{|\ Ax - By + C\ |}{\sqrt{A^2 + B^2}}$$
$$or \qquad h = \frac{|(d_y - s_y)x - (d_x - s_x)y + (d_x s_y - s_x d_y)|}{\sqrt{(d_y - s_y)^2 + (d_x - s_x)^2}} \tag{3}$$

   All the variables used are discussed already and presented in algorithm *SHORT-QUEUE (G, H)* of the next Section.

5. Repeat step 4 until all nodes are explored. Covered in the loop of the *SHORT-QUEUE (G, H)* algorithm.

6. Scan path from source to destination with in short queue SQ nodes. If no path exists from source to destination in the selected nodes, then set D=HD/RF and add more nodes in the short queue using step 4 and 5 as shown in Fig. 2.

7. Repeat step 4, 5 and 6 and select appropriate nodes that have a path from source to destination until all nodes of the graph are added in the short queue SQ.

3.2 Solution Using Short Queue

After shortlisting the appropriate nodes in short queue SQ, run the Dijkstra's algorithm to find the shortest path, as presented in line 10 of the Capton algorithm. Using the Dijkstra's algorithm, we can easily find the shortest path in the given graph. In general, Dijkstra's algorithm normally runs over the whole graph and scans each node to find the solution; however, in our proposed method, we minimized the number of nodes that could participate in the shortest path from source to destination. This approach reduces the time in the case of single source shortest path problem.

```
1.  CAPTON (G, s, d)
2.  {
3.  Center-line(s,d) ▷ s and d are source and destination nodes.
4.  Integer D;
5.  Integer RF=6;
6.  SQ= Φ  ▷ short queue used for short listed nodes.
7.  D=HD/RF      ▷fix solution set area.
8.  SHORT-QUEUE (G, s, d, H)
9.  Scan SQ ▷ If SQ has no destination vertex in it, then Set D=D+HD/RF and call
    the SHORT-QUEUE() method.
10. Run Dijkstra's algorithms on SQ to find shortest path
11. }
```

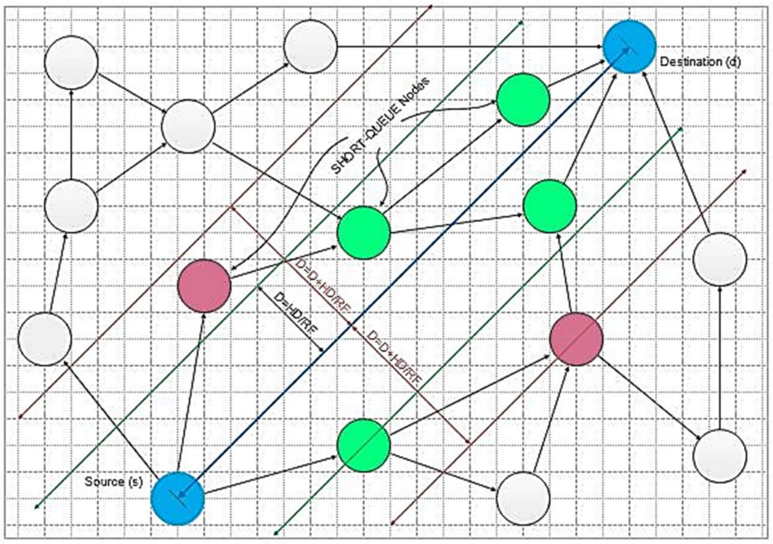

Fig. 2: Expansion in Problem Area by Increasing D

```
1. Center-line (s, d)
2. {
3. Plot centerline using straight line method
4. H.A = $d_y$- $s_y$;
5. H.B = $d_x$- $s_x$;
6. H.C= $d_x s_y$ - $s_x d_y$;
7. HD=$\sqrt{(d_x - s_x)^2 + (d_y - s_y)^2}$; ▷ heuristic distance from source to node,
   where x and y are coordinates.
8. Return H;
9. }
1. SHORT-QUEUE (G, s, d, H)
2. {
3. For each vertex v ∈ G.V
4. {
5. v.h= $\frac{|H.A*v.x-H.B*v.y+H.C|}{\sqrt{(H.A)^2+(H.B)^2}}$ ▷ heuristic distance of v from centerline.
6. If v.h ≤ D and path exists from preceding vertex to current vertex v,
   then SQ = SQ {v}
7. }
8. RETURN SQ
9. }
```

4.0 TIME COMPLEXTY

Time complexity of Capton algorithm is depicted in Fig. 3, and discussed as follows. The statements in line 3 to 7 are already detailed in Section 3.2. These lines of code, highlighted with square brackets, have complexity O(1) time. Similarly, the *SHORT-QUEUE (G, s, d, H)* method (called at line 8) takes O(V) time to short list the nodes in a given graph. This method produces a short queue SQ in which preceding vertex contains a path to following vertex. It is possible that proper production of SQ would fail, and no such path between any preceding and following vertex of SQ has been found, which will result in absence of destination vertex in SQ. To resolve this issue, line 9 of algorithm runs in O(v) times if proper production of SQ would have been achieved and would have O(V) time worst case complexity if SQ has no destination vertex in it after production. Here, v (short-listed nodes) is smaller than V number of vertices (total vertices in graph). Finally, the Dijkstra's algorithm runs on SQ with short listed nodes in O(vlogv + e) if SQ contains proper short-listed vertices and O(VlogV + E) in worst case as discussed. Latter is the complexity of the Dijkstra's algorithm on a given graph using Fibonacci heap as queue [18], [19]. In real time paths, the number of edges |e| is not necessary to be equal to the number of short-listed vertices |v|, or the number of edges |e|, it can be equal to the square of short listed vertices $|v^2|$. So, the O(vlogv + e) could not be evaluated to eloge, or $O(|v^2|)$ in case of |e|=|v|, or $|e| = |v^2|$, respectively. Hence, the running time of Dijkstra's algorithm is reduced to O(vlogv + e) from O(VlogV + E) using short listed method described in Section 3.0 and also coded in Capton algorithm.

```
                 1.  CAPTON (G, s, d)
               { 2.  {
               { 3.  Center-line(s,d) ▷ s and d are source and destination nodes.
O(1)           { 4.  Integer D;
               { 5.  Integer RF=6;
               { 6.  SQ= Φ     ▷ short queue used for short listed nodes.
               { 7.  D=HD/RF  ▷fix solution set area.
O(V)           { 8.  SHORT-QUEUE (G, s, d, H)
O(v)|O(V)      { 9.  Scan  SQ  ▷ If  SQ  has  no  destination  vertex  in  it,  then  Set
                     D=D+HD/RF and call the SHORT-QUEUE() method.
O(vlogv +e)|   { 10. Run Dijkstra's algorithms on SQ to find shortest path
O(VlogV+E)       11. }
```

Fig. 3: Time Complexity of the Capton Algorithm

Finally, with the help of analysis presented, the total time complexity can be summed up using Fig.3, which becomes as given in Equation 4.

$$O(\text{vlogv} + \text{e}) = O(1) + O(\text{V}) + O(\text{v}) + O(\text{vlogv} + \text{e}) \qquad (4)$$

Where V is the total number of vertices in a given graph G, v and e are the total short-listed vertices and edges in SQ, respectively. After applying the ignoring factor, the final complexity of the Capton algorithm becomes O (vlogv + e), which is always less than the Dijkstra's algorithm due to short listed vertices v with respect to the total number of vertices V in a given graph G. Moreover, the worst case time complexity O(VlogV + E) of Capton algorithm is not possible in real time graph paths; however, it can be achieved explicitly if SQ will not contain path from preceding vertex to destination vertex until all vertices of a given graph are explored.

5.0 EXAMPLE

The above algorithm can be understood with the following example. In this example, each step of algorithm is explained enough to give an idea about its performance. It performs better as compared to the Dijkstra's algorithm and, hence, becomes its variant. The effectiveness of Capton algorithm is evaluated after implementation on several graphs. Among them, a graph is depicted in Fig. 4 to explain the working of Capton algorithm. A network of 16 cities connecting each other is presented and scaled at 1mm = 1 KM.

As the algorithm has two steps, the following points are discussed in Section 3.0.

1. Shortlisting of graph vertices.
2. Find single source shortest path from vertices, lying in the short queue SQ.

Now, the running of the Capton algorithm is outlined next with given graph depicted in Fig. 4 along with source s and destination d vertices.

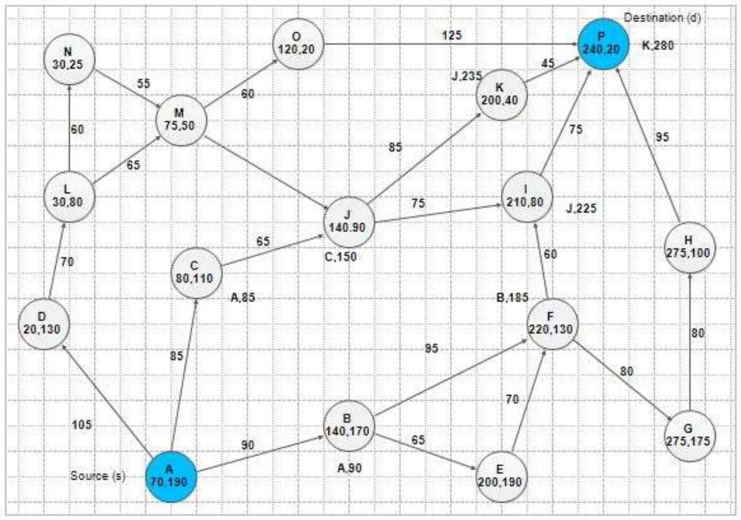

Fig. 4: Cities Network Graph

5.1 Shortlisting of Graph Nodes

First step is to shortlist the graph nodes to reduce the search space. The main factors in this shortlisting include: the straight line between source and destination nodes using their x and y coordinates, the calculation of a heuristic distance HD from source to destination node or vertex, a solution set reduction bound at distance D from the centerline that is passing through source and destination, and another heuristic distance h of each node from that center line.

According to Equation (1), we have the following version as

$$(d_y – s_y)x – (d_x – s_x)y + (d_x s_y – s_x d_y) = 0$$

Substituting values from the graph in Fig 4, we have the details of the above version as follows.

$$(20 - 190)x - (240 - 70)y + (240 * 190 - 70 * 20) = 0$$
$$(-170)x - 170y + 44200 = 0$$

With the help of above values, a straight line is plotted between source and destination vertices depicted in Fig. 5. Further, H.A= –170, H.B=170 and H.C=44200 are extracted to be used in second step. The heuristic distance between source and destination vertices is calculated as follows using a statement at line 7 of the *Center-line (s, d)*.

$$HD = \sqrt{(d.x - s.x)^2 + (d.y - s.y)^2}$$
$$HD = \sqrt{(240 - 70)^2 + (20 - 190)^2}$$
$$HD = 240.416$$

In the Capton algorithm, after having RF = 6, the value of D becomes D = HD/RF = 240.416/6 = 40.06 mm and method *SHORT-QUEUE (G, A, P, H)* is called, which produces the SQ with {A, B, I, J, K, P} vertices.

Table 1: Heuristic Distance of Graph Nodes

Node	X	Y	A	B	C	h
A	70	190	-170	170	44200	0.0
B	140	170	-170	170	44200	35.4
C	80	110	-170	170	44200	49.5
D	15	130	-170	170	44200	81.3
E	200	190	-170	170	44200	91.9
F	220	130	-170	170	44200	63.6
G	275	175	-170	170	44200	134.3
H	275	100	-170	170	44200	81.3
I	210	80	-170	170	44200	21.2
J	140	90	-170	170	44200	21.2
K	200	40	-170	170	44200	14.1
L	30	80	-170	170	44200	106.1
M	75	50	-170	170	44200	95.5
N	30	25	-170	170	44200	144.9
O	120	20	-170	170	44200	84.8
P	240	20	-170	170	44200	0.00

Initially, as the value of D is set to 40.06 mm and {A, B, I, J, K, P} nodes are inserted in SQ using $v.h \leq D$, but after scanning of SQ nodes at line 9 of Capton algorithm, no other path has been found to destination as shown in Fig. 5 and Table 1. So, setting D increases the problem area by D = D+HD/RF and *SHORT-QUEUE (G, A, P, H)* method at line 8 is called again. This call introduces two new nodes F and C into the SQ as shown in Fig. 6. The D and SQ have the values as D = D+H.D/RF = 40.06 + 240.416/6 = 80.12 mm and SQ = {A, B, C, F, I, J, K, P}, respectively where $v.h \leq D = 80.12$mm.

After completion of *SHORT-QUEUE (G, A, P, H)* and scanning of SQ at line 8 and 9, respectively, this time the SQ contains paths between preceding and following vertices. Hence, ready for final calculation of single source shortest path of a given graph briefed in Section 5.2. By applying the strategy discussed, the problem area is now minimized up to a small version (eight vertices - A, B, C, F, I, J, K, and P) of the input graph G (with 16 vertices) given in Fig. 6 and Table 1.

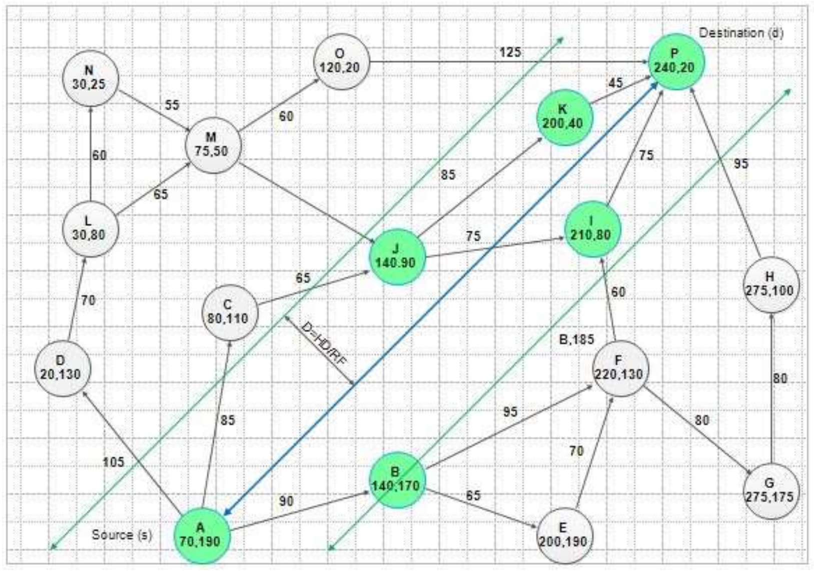

Fig. 5: First Step Short Listing of Graph Nodes

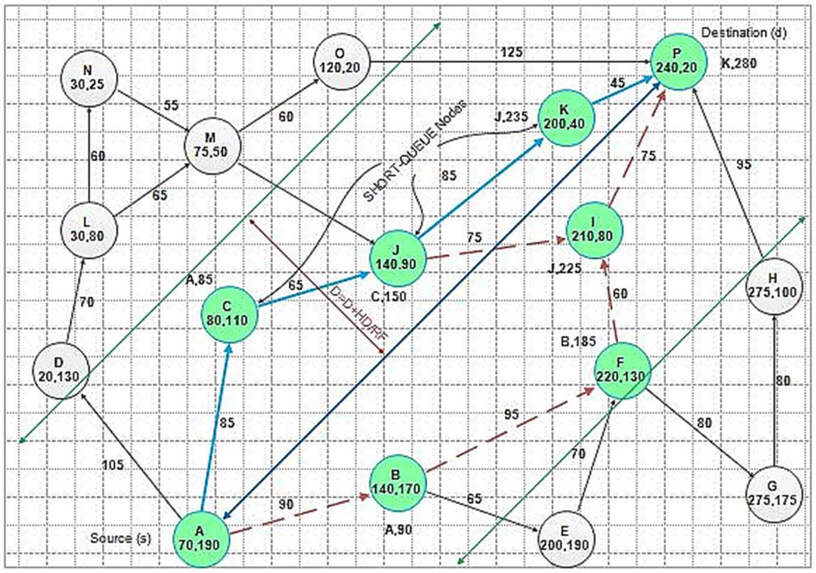

Fig. 6: Shortlisted Graph Nodes

5.2 Find Solution from the Shortlisted Nodes

Second step is to find the shortest path from the shortlisted nodes along with their respective paths. Weights (distances) are labeled on the paths between the city nodes. To find the shortest path, Dijkstra's algorithm on selected nodes in SQ is then permitted to run at line 10 of Capton algorithm. At this stage, the SQ consists of A, B, C, F, I, J, K, and P nodes and their respective weights. It is beyond the scope of this article to show the running of Dijkstra's algorithm; however, its computation is given in the following Table 2 without discussion. After traversing from source A to destination P, a shortest path of 280mm (280KM) passing through A, C, J, K, and P cities is obtained.

Table 2: Shortest Path of Selected Node from Source.

V	A	B	C	F	I	J	K	P
0	∞	∞	∞	∞	∞	∞	∞	∞
A	0A	90A	85A	∞	∞	∞	∞	∞
C	0A	90A	85A	∞	∞	150C	∞	∞
B	0A	90A	85A	185B	∞	150C	∞	∞
J	0A	90A	85A	185B	225J	150C	235J	∞
F	0A	90A	85A	185B	225J	150C	235J	∞
I	0A	90A	85A	185B	225J	150C	235J	300I
K	0A	90A	85A	185B	225J	150C	235J	280K
P	0A	90A	85A	185B	225J	150C	235J	280K

6.0 PROS AND COSNS OF CAPTON AND DIJKSTRA ALGORITHMS

For a graph G, Dijkstra's algorithm takes O(VlogV+E) time to compute the single source shortest path where V is the total number of vertices in the graph G [20]. In this method, we have to explore and calculate the shortest path by visiting all the vertices V in a graph, and therefore, the time complexity increases with respect to total number of vertices V or edges E in that graph G.

In Capton algorithm, the solution set is reduced as minimum as possible up to v vertices (sufficiently less than V). As vertices decrease from V to v, the number of edges also decreases from E to e and therefore, the Capton algorithm performs better than the Dijkstra's algorithm.

Dijkstra's algorithm uses the weights labeled on the edges between vertices. A record of source vertex and each vertex distance from its predecessor is maintained. The Capton algorithm requires the same, plus some extra information. It stores x and y coordinates of each vertex in the graph. This information needs a little extra memory and time resources to compute the solution set. Heuristic distance of each node from the centerline is also calculated. This evaluation of additional information reduces the performance of the Capton algorithm, but still outperforms the Dijkstra's algorithm in real time (see Section 4.0 for details).

Considering these pros and cons of the Capton algorithm, it is quite clear that this algorithm works efficiently and performs better than the Dijkstra's algorithm in real life problems such as finding shortest path among cities. Neighboring vertices on a shortest path or heuristic path can be identified using this technique. It can also help in building optimal shortest paths by visualizing and introducing non-existing edges in path-related tools.

7.0 COMPARISON WITH OTHER ALGORITHMS

An overview of time complexities of different shortest path algorithms for directed graphs with nonnegative weights was presented in [21], which is repeated here in Table 3, along with the time complexity of Capton algorithm. This algorithm outperform the Dijkstra's algorithm using the same data structure (Fibonacci heap) with time complexity O(vlogv + e), if used in real time situations (discussed in Section 4.0).

Table 3: Comparison of Algorithms Directed Graphs

Algorithm	**Time complexity**	**Authors**
Bellman-Ford-Moore	O(VE) using list	Bellman in [1], Ford in [16] and Moore in [22]
Dijkstra	O(V2) using a list	Dijkstra in [20] and Cormen in [23]
Dijkstra	O(ElogV) using binary heap	Fredman et al. in [18] and Cormen in [23]
Dijkstra	O(E+VlogV) using Fibonacci Heap	Fredman et al. in [18] and Cormen in [23]
Floyd Warshall	O(V3) using list	Floyd in [3]
Johnson's Algorithm	O(V2log V + VE) using Fibonacci heap	Johnson in [4]
Capton Algorithm	O(vlogv + e) with Fibonacci Heap	Abbas et al. in this work

8.0 AREA REDUCTION FACTOR TRADEOFF

Area reduction factor (RF) have an essential impact in the time complexity of the Capton algorithm. Greater value of RF minimizes the problem space. Narrow selected area can reduce the search space and time but the probability of path finding from source to destination also becomes low. There is a tradeoff that we may not find shortest path or any path in selected graph nodes. In such a case, we can increase the search space by increasing the value of D, but excessive scanning of path in SQ and insertion of new nodes from the graph in SQ increases the search time. The question arises, how much can we reduce the problem area?

RF tradeoff depends on the nature of different graphs. After running this algorithm on different graphs, it has been observed that higher value of the RF factor minimizes the search time in dense graphs, and lower RF value minimizes the search time in sparse graphs. For example, in our exercise, the algorithm performed best at value of 10 to 50 in case of dense graphs. Similarly, the best search time for sparse graphs was achieved at RF value of 4.0 to 8.0. In short, using Capton algorithm, optimum solutions can be obtained in case of graphs with large number of nodes.

9.0 CONCLUSION

Heuristic distance function makes the Capton algorithm able to have an improved computing time. It guarantees to find the shortest path by increasing the problem area from smaller to larger search space, and hence, can find the optimal path with in linear time. The tradeoff discussed between smaller and larger search space depends on the sparsity and density of a graph. This algorithm has the capability to repeat the search process by increasing the reduction factor, the only difference between this and Dijkstra's algorithm, hence, it is better to call it variant of Dijkstra's algorithm. The efficiency and performance of the Capton algorithm is better than the Dijkstra's algorithm on real world examples such as finding shortest path among cities. Neighboring vertices on a shortest path or heuristic path can be identified using this technique. It can also help in building new shortest routes through visualizing and manipulating absent paths in design-related tools and models.

REFERENCES

[1] R. Bellman, "On a routing problem", *Technical report, DTIC Document*, 1956.

[2] T. J. Misa and P. L. Frana, "An interview with Edsger W. Dijkstra", *Communications of the ACM* , Vol. 53, No. 8, 2010, pp. 41-47.

[3] R. W. Floyd, "Algorithm 97: shortest path", *Communications of the ACM*, Vol. 5, No. 6, 1962, pp. 345.

[4] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks", *Journal of the ACM*, Vol. 24, No. 1, 1977, pp. 1–13.

[5] M. Agrawal, N. Kayal and N. Saxena, "Primes is in p", *Annals of mathematics*, Vol. 160, No. 2, 2004, pp. 781–793.

[6] C. Wiener, "Ueber eine Aufgabe aus der Geometria situs", *Mathematis-che Annalen*, Vol. 6, 1873, pp. 29–30.

[7] E. Lucas, *Re ́cre ́ationsmathe ́matiques*, Vol. 1, Gauthier-Villars,1882.

[8] T. Gaston, "Le probleme des labyrinthes", *Nouvelles annales de mathématiques, journal des candidats aux écoles polytechnique et normale*, Vol. 14, 1895, pp. 187-190.

[9] N. L. Biggs, E. K. Lloyd and R. J. Wilson, *Graph Theory 1736–1936*, Clarendon Press Oxford, 1976, pp. 18–20.

[10] H.D. Landahl and R. Runge, "Outline of a matrix calculus for neural nets", *The bulletin of mathematical biophysics*, Vol. 8, No. 2, 1946, pp. 75–81.

[11] L. B. Huang, V. Balakrishnan, R.G. Raj, "Improving the relevancy of document search using the multi-term adjacency keyword-order model." *Malaysian Journal of Computer Science*, Vol. 25, No. 1, 2012, pp. 1-10.

[12] R. D. Luce and A. D. Perry. "A method of matrix analysis of group structure", *Psychometrika*, Vol. 14, No. 2, 1949, pp. 95–116.

[13] R. G. Raj, and V. Balakrishnan, "A Model For Determining The Degree Of Contradictions In Information" *Malaysian Journal of Computer Science*, Vol. 24, No. 3, 2011. pp 160-167.

[14] M. A. Shayegan, S. Aghabozorgi, R. G. Raj, "A Novel Two-Stage Spectrum-Based Approach for Dimensionality Reduction: A Case Study on the Recognition of Handwritten Numerals," *Journal of Applied Mathematics*, vol. 2014, Article ID 654787, 14 pages, 2014. doi:10.1155/2014/654787.

[15] A. Shimbel. "Applications of matrix algebra to communication nets", *Bulletin of Mathematical Biology*, Vol. 13, No. 3, 1951, pp. 165–178.

[16] L. R. Ford Jr, *Network Flow Theory*, The RAND Corporation, Santa Monica, California, August 14, 1956, p. 923.

[17] P. Alfred, *The Pythagorean Theorem: The Story of Its Power and Beauty*, Prometheus Books, 2010, pp. 23.

[18] M. L. Fredman, Michael L., and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms", *Journal of the ACM (JACM)*, Vol. 34, No. 3, 1987, pp. 596-615.

[19] K. Mehlhorn and P. Sanders, *Algorithms and data structures: The basic toolbox*, Springer Science & Business Media, 2008.

[20] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische mathematik*, Vol. 1, No. 1 1959, pp. 269-271.

[21] A. Schrijver, *Combinatorial Optimization Polyhedra and Efficiency, Algorithms and Combinatorics*, *Vol. A*, Springer, 2004, pp.103.

[22] E. F. Moore, *The shortest path through a maze*, Bell Telephone System, 1959.

[23] T. H. Cormen, *Introduction to algorithms*, MIT press, 2009.