International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org, editorijaiem@gmail.com
Volume 2, Issue 7, July 2013 ISSN 2319 - 4847

Shortest Path Algorithm

Shivani Sanan*[1], Leena jain[2], Bharti Kappor[3]

[*1]Assistant Professor, Faculty of Mathematics, Department of Applied Sciences
Global Institute of Management and Emerging Technologies, Amritsar

[2]Associate Professor & Head- MCA
Global Institute of Management and Emerging Technologies, Amritsar

[3]Assistant Professor, Faculty of Mathematics, Department of Applied Sciences
Global Institute of Management and Emerging Technologies, Amritsar

Abstract

The Shortest Path Problem (SPA) is one of the most fundamental and important in combinatorial Problem.SPA is an important problem in graph theory and has applications in communications, transportation, and electronics problems. In this paper different algorithm for solving SPA with their advantage, disadvantage and application has been discussed.

Keywords: Shortest Path Algoritm, Dijkstra's Algorithm, Bell Bellman-Ford's Algorithm, A* search algorithm, Floyd–Warshall algorithm

1. Introduction

The Shortest Path problem is defined on a directed, weighted graph, where the weights may be thought of as distances. The objective is to find a path from a source node, s, to node a sink node, t that minimizes the sum of weights along the path. To formulate as a network flow problem, let xi,j be 1 if the arc (i, j) is in the path and 0 otherwise. Place a supply of one unit at s and a demand of one unit at t. The formulation for the problem is:

$$\begin{array}{lll} \text{Min} & \sum_{((i,j)\in A} d_{i,j}x_{i,j} & \\ \text{subject to} & \sum_{(i,k)\in A} x_{i,k} - \sum_{(j,i)\in A} x_{j,i} = 0 & \forall i \neq s,t \\ & \sum_{(s,i)\in A} x_{s,i} = 1 & \\ & \sum_{(j,t)\in A} x_{j,t} = 1 & \\ & 0 \leq x_{i,j} \leq 1 & \end{array}$$

This problem is often solved using a variety of search and labeling algorithms depending on the structure of the graph. The most important algorithms for solving this problem with their advantage, disadvantages are describied in section 2. Analysis of different algorithm is given in table 1.

2. DESCRIPTION OF ALGORITHMS

2.1 Dijkstra's Algorithm

1) Dijkstra is a Greedy based algorithm and solves the single-source shortest path problems
2) Dijkstra doesn't work for negative weight edges.
3) Require global information of the network
4) Algorithm

```
Shortest path(v,cost,dist,n)
// v: Initial vertex; Cost: Weight; n: Number of vertex;
dist[j], 1≤j≤n, is the set to the length of the shortest path from vertex v to vertex j in a digraph G with n
vertices. dist[v] is set to zero. G is represented by its cost adjacency matrix cost [1:n, 1:n]
{
for i : =  1 to n do
  {  //  Initialize S.
 S[ i ] : =false. Dist [i]: = cost {v,i];
}
  S [ v ] : = true; dist[ v ] :  = 0.0;// put v in S.
   For num : = 2 to n do
   {
 // Determine n-1 paths from v  choose u from among those vertices not in s such that dist [u] is minimum;
      S [ u] : = true; // put u in S
      For   (each w adjacent to u with S[w] = false) do
```

```
    // update distances
If (dist [ w ] > dist [ u ] + cost [ u, w ] ) )  then
dist [ w ] : = dist [ u ] + cost [ u, w ] ;
            }
}
```

5) Time complexity of Dijkstra is O(|E| + |V|Log|V|)

2.1.1 Application of Dijkstra's Algorithm

- ➢ Robot path planning [1]
- ➢ Logistics Distribution Lines [2]
- ➢ Link-state routing protocols [3]
- ➢ OSPF(Open Shortest Path First)
- ➢ IS-IS (Intermediate System to Intermediate System**)**

2.1.2 Disadvantage of Dijkstra's Algorithm

- ➢ The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources.
- ➢ It cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path.

2.2 Bellman-Ford's Algorithm

1) Bellman-Ford is a Dynamic Programming based algorithm.
2) Bellman-Ford works for negative weight edges.
3) Uses only local knowledge of neighboring nodes
4) Algorithm

```
Algorithm Bellman Ford (v, cost , dist , n )
     // Single-source / all-destinations shortest  paths with negative edge costs
     {
       for i := 1 to n do // Initialize dist.
            dist [i] : = cost [ v , i ] ;
       for k : = 2 to n-1 do
            for each u such that u ≠ v and u has
                          at least one incoming edge do
                  for each < i, u > in the graph do
                        if  dist [u] > dist [i] + cost [i,u] then
                            dist [u] : = dist [i] + cost [i,u] ;
     }
```

5) Time complexity of Bellman Ford takes O(|V||E|) time where |V| and |E| are number of vertices and edges. This time complexity is the best time bound for the single source shortest path problem [4].

2.2.1 Application of Bellman-Ford's Algorithm

- ➢ A distributed variant of the Bellman–Ford algorithm is used in distance-vector routing protocols.
- ➢ Saving network resource
- ➢ Routing Information Protocol (RIP)
- ➢ Two metric routing problem [5]

2.2.2 Advantages of Bellman-Ford's Algorithm

- ➢ We can minimize our cost when we build a network. It is because the Bellman-ford algorithm will find the shortest path weight from a given source node subject to other node. Therefore, we need not build much of router to build path from a node to other.
- ➢ Bellman-Ford algorithm also can maximize the performance of your system. The algorithm will find the minimum path weight. Path weight is propagation delays for a system.

- Bellman-Ford also allows splitting traffic between several paths. This action will increase the system performance.

2.2.3 Disadvantage of Bellman-Ford's Algorithm

- The main disadvantage of the Bellman–Ford algorithm in RIP is that it doesn't take weightings into consideration.
- Another disadvantage of the Bellman-Ford algorithm is due to the slow updates passed from one RIP router to the next. This results in a slow response to changes in the network topology, which in turn results in more attempts to use routes that are down, which wastes time and network resources.

2.3 A* search algorithm

1) A* algorithm is a graph/tree search algorithm that finds a path from a given initial node to a given goal node
2) It employs a "heuristic estimate" h(x) that gives an estimate of the best route that goes through that node.
3) It visits the nodes in order of this heuristic estimate.
4) It follows the approach of best first search and and finds a least-cost path from a given initial node to one goal node.
5) As A* traverses the graph, it follows a path of the lowest expected total cost or distance, keeping a sorted priority queue of alternate path segments along the way.
6) It uses a knowledge-plus-heuristic cost function of node *x* (usually denoted *f(x)*) to determine the order in which the search visits nodes in the tree. The cost function is a sum of two functions:
    i. The past path-cost function, which is the known distance from the starting node to the current node *x* (usually denoted *g(x)*)

    ii. A future path-cost function, which is an admissib le "heuristic estimate" of the distance from *x* to the goal (usually denoted *h(x)*).
7) The *h(x)* part of the *f(x)* function must be an admissible heuristic; that is, it must not overestimate the distance to the goal.
8) A* is equivalent to running Dijkstra's algorithm with the reduced cost $d'(x, y) := d(x, y) - h(x) + h(y)$.
9) Analysis: 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 [6][7].

2.3.1 Application of A* Search algorithm

A* is commonly used for the common pathfinding problem in applications such as games, but was originally designed as a general graph traversal algorithm [8]. It finds applications to diverse problems, including the problem of parsing using stochastic grammars in NLP [9].

2.4 Floyd–Warshall algorithm

1) This algorithm solves all pair's shortest paths problem in an edge directed graph.
2) This algorithm work with positive or negative edge weights.
3) This algorithm does not work with any negative cycle.
4) This algorithm doesn't find the paths
5) This algorithm finds only their minimum path lengths.
6) Algorithm

```
Floyd-Warshall(W)
        $n=w.rows$
        $d^{(0)}=w$
        for $k = 1$ to $n$
                let $d^{(k)}=(d_{ij}^{(k)})$ be a new matrix
                for $i = 1$ to $n$
                        for $j= 1$ to $n$
                        $d_{ij}^{(k)}= \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} +d_{kj}^{(k-1)})$
return $d^{(n)}$
```

7) Time Complexity: $o(n^3)$
8) Space Complexity: $o(n^3)$

2.4.1 Applications Floyd–Warshall algorithm

- ➢ Shortest path in directed graph.
- ➢ Transitive closure of directed graphs.
- ➢ Finding a regular expression denoting the regular language by finite automaton.
- ➢ Inversion of real matrics.
- ➢ Optimal routing

2.5 Johnson's algorithm

1) Algorithm solves all pairs shortest paths, in a sparse weighted, directed graph.
2) It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist.
3) This algorithm may be faster than Floyd–Warshall on sparse graphs.
4) Johnson's algorithm uses as subroutines both Dijkstra's algorithm and the Bellman-Ford algorithm,
5) Johnson's algorithm uses the technique of reweighting which works as follows[10,11]:
   a. First, it adds a new node q with zero weight edges from it to all other nodes.
   b. Then runs the Bellman-Ford algorithm starting from the new vertex *q*, to find for each vertex *v* the minimum weight *h*(*v*) of a path from *q* to *v*. Bellman-Ford algorithm used to check for negative weight cycles. If this step detects a negative cycle, the algorithm is terminated.
   c. Next the edges of the original graph are reweighted using the values computed by the Bellman–Ford algorithm: an edge from *u* to *v*, having length $w(u,v)$, is given the new length $w(u,v) + h(u) - h(v)$.
   d. Finally, *q* is removed, and Dijkstra's algorithm is used to find the shortest paths from each node *s* to every other vertex in the reweighted graph.
6) Analysis: The time complexity of this algorithm, using Fibonacci heaps in the implementation of Dijkstra's algorithm, is $O(V^2\log V + VE)$: the algorithm uses $O(VE)$ time for the Bellman–Ford stage of the algorithm, and $O(V \log V + E)$ for each of V instantiations of Dijkstra's algorithm. Thus, when the graph is sparse, the total time can be faster than the Floyd–Warshall algorithm, which solves the same problem in time $O(V^3)$[10]

Table 1: Analysis of different algorithm

ALGORITHM	Negative edge	Single Source	All Sources	Time Complexity	Space Complexity
Dijkstra		√		O(|E| + |V|Log|V|)	$O(\mathbf{v}^2)$
Bellman-Ford	√	√		$O(VE)$	$O(\mathbf{v}^2)$
FloydWarshall	√		√	$O(\mathbf{n}^3)$	$O(\mathbf{n}^3)$
Johnson	√		√	$O(V^2\log V + VE)$	
A*-Search		√		Depends on the heuristic	Depends on the heuristic

3. CONCLUSION

SPA is an important problem in graph theory and has applications in communications, transportation, and electronics problems. In this paper different algorithm for solving SPA with their advantage, disadvantage and application has been discussed.

Reference

[1.] Huijuan Wang et. al , "Application of Dijkstra algorithm in robot path-planning", Second International Conference on Mechanic Automation and Control Engineering (MACE), pp. 1067 - 1069 ,2011
[2.] Liu Xiao-Yan, Chen Yan-Li, "Application of Dijkstra Algorithm in Logistics Distribution Lines", Proceedings of the Third International Symposium on Computer Science and Computational Technology(ISCSCT '10) *14-15,August 2010, pp. 048-050*
[3.] Radwan s. Abujassar and mohammed ghanbari, "Efficient algorithms to enhance recovery schema in link state protocols", international journal of ubicomp (iju), vol.2, no.3 , july 2011 doi:10.5121/iju.2011.2304 53
[4.] Cherkassky B. V., Goldberg, A. V., "NegativeCycle Detection Algorithms", Mathematical Programming, Springer-Verlag, vol. 85, pp. 277311, 1999.
[5.] D. Cavendish and M. Gerla (1998), "Internet QoS Routing using the Bellman-Ford Agorithm", Chapman & Hall.

[6.] Pearl, Judea (1984). “Heuristics: Intelligent Search Strategies for Computer Problem Solving”, Addison-Wesley. ISBN 0-201-05594-5
[7.] Russell, S. J.; Norvig, P. (2003). “Artificial Intelligence: A Modern Approach”. Upper Saddle River, N.J.: Prentice Hall. pp. 97–104. ISBN 0-13-790395-2.
[8.] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2): 100–107. doi:10.1109/TSSC.1968.300136.
[9.] Klein, Dan; Manning, Christopher D. (2003). "A* parsing: fast exact Viterbi parse selection". Proc. NAACL-HLT.
[10.]Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), “*Introduction to Algorithms”*, MIT Press and McGraw-Hill, ISBN 978-0-262-03293-3.
[11.]Black, Paul E. (2004), "Johnson's Algorithm", Dictionary of Algorithms and Data Structures, National Institute of Standards and Technology.