MULTI-CRITERIA PATH FINDING

Ehsan Mohammadi, Andrew Hunter

Geomatics Department, Schulich School of Engineering, University of Calgary
(emohamma, ahunter, @ucalgary.ca)

Commission II, WG II/7

KEY WORDS: GIS, Analysis, Network, Databases, Query, Parameters

ABSTRACT:

Path finding solutions are becoming a major part of many GIS applications including location based services and web-based GIS services. Most traditional path finding solutions are based on shortest path algorithms that tend to minimize the cost of travel from one point to another. These algorithms make use of some cost criteria that is usually an attribute of the edges in the graph network. Providing one shortest path limits user's flexibility when choosing a possible route, especially when more than one parameter is utilized to calculate cost (e.g., when length, number of traffic lights, and number of turns are used to calculate network cost.) K shortest path solutions tend to overcome this problem by providing second, third, and K^{th} shortest paths. These algorithms are efficient as long as the graphs edge weight does not change dynamically and no other parameters affect edge weights. In this paper we try to go beyond finding shortest paths based on some cost value, and provide all possible paths disregarding any parameter that may affect total cost. After finding all possible paths, we can rank the results by any parameter or combination of parameters, without a substantial increase in time complexity.

1. BACKGROUND AND RELEVANCE

1.1 Shortest Path Algorithms

A best path through a network from an origin to a destination is a path having the least value (Hoffman and Pavley, 1959). Value (or weight) refers to any metric that describes the effort to traverse the path; such as length, time, cost, or roughness of the path, etc. These metrics can be assigned to the graph which describes the path network in an environment. However the best path is the one that minimizes the total value of the path. For some applications distance is the most important characteristic of a path, and can be extracted directly from geometry of the network; while for some other applications time may be the most important characteristic.

Dijkstra's (1959) algorithm is one of the best known algorithms to find the shortest path. This algorithm works on a weighted graph in which the edges weights are non-negative (Dreyfus, 1969). Most of the applications that try to find the shortest path rely on this algorithm. Dijkstra's (1959) algorithm is able to calculate all shortest paths from a single point to all other points in a network, which makes it a single source shortest path algorithm.

1.2 K Shortest Path Algorithms

However, sometimes it is desirable to know the second (third, etc.) shortest path in a network in addition to the first shortest path (Dreyfus, 1969). There may be different reasons behind finding the second (third, etc.) shortest path. Some applications require alternative routes to the shortest path for emergency reasons. Other applications may provide alternative paths that are not that different from the shortest path, but offer different characteristics. Also understanding the alternative paths from one point to another provides flexibility for applications and gives users the choice to select from a set of available paths.

The problem of finding more than one shortest path is defined as the K shortest path problem (Hoffman and Pavley, 1959). By definition, a solution to the K shortest paths problem returns a set of K shortest paths between two locations given a particular weighting function (Hoffman and Pavley, 1959).

Shortest path and K shortest path algorithms can be considered as one-criteria algorithms (Dreyfus, 1969); since all affecting parameters should be modelled as weight of segments in the network. Modelling all parameters as a weight function restricts the flexibility of path finding solutions and limits combination of these solutions with other spatial analysis solutions.

2. METHOD

2.1 All Possible Paths

The all possible paths solution, as a multi-criteria solution, is an alternative to the K shortest path solution in which the

value of K may range from a minimum (i.e. 1) to a maximum number of paths dynamically. The all possible paths method finds all paths that connect two nodes together in a network regardless of any parameters. In other words, any restrictions can be applied as a separate parameter on this solution, making it a multi-criteria solution. So the value of edges in a network is not the only parameter for path finding in this method, but a range of different parameters (criteria) can also be applied.

2.2 Algorithm

The all possible paths solution utilizes database operations to find and store all paths from one point (origin) to another (destination). For this purpose, we have to provide the database with start and end point of each edge in the network along with all characteristics of the edges that can be considered later as parameters for detecting the best path. The process starts with an iteration in which the database tries to find all edges that start with the origin point. Then the algorithm joins those edges with all other edges for which the start point is equal to the end point of the current edge. This process iterates until the end point (destination) of the path is reached.

To illustrate the algorithm we consider the graph in Figure 1. In this graph each point (node) has a unique ID and value of each segment is shown. However, this algorithm is independent from the value of segments.

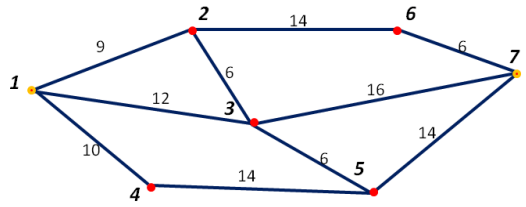

Figure 1. A sample graph

The goal is to find all possible paths from point 1 to point 7 in the network, assuming the network is a directed graph. In the database this directed graph can be stored as a table with at least 3 columns (ID, Start-Node-ID, End-Node-ID) plus additional cost related variables such as length, time, etc. Figure 2 shows the corresponding table for figure above.

ID	Start –Node-ID	End –Node-ID	Cost
1	1	2	9
2	1	3	12
3	1	4	10
4	2	6	14
5	2	3	6
6	3	5	6
7	3	7	16
8	4	5	14
9	5	7	14
10	6	7	6

Figure 2. Table storing graph

In order to find all possible paths from point 1 to point 7, the table above is joined to itself repeatedly until the segment with destination as end node is reached. Through the join operation all records that have point 1 as the start node are selected and joined with records in the same table where end nodes of first set of records are equal to start nodes of second table. This process is repeated until records with point 7 as the end nodes are reached. Figure 3 illustrates this process.

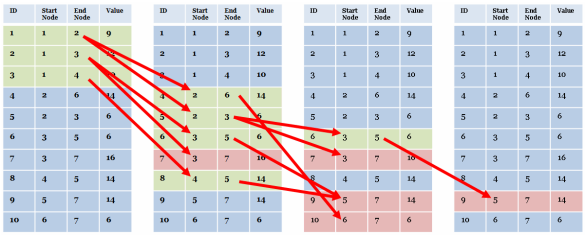

Figure 3. Process of finding all possible paths from 1 to 7

This process provides us with following results. There are 6 possible paths from point 1 to point 7 and the paths are as follow:

{1,2,6,7} Cost=29
{1,2,3,5,7} Cost=35
{1,2,3,7} Cost=31
{1,3,5,7} Cost=32
{1,3,7} Cost=28
{1,4,5,7} Cost=38

These paths can be ranked and stored in a database for future uses. Storing the results in a database is more useful for applications that require frequent access to the results.

3. DISCUSSION

3.1 Results

The complexity function is one of the best methods for comparing the results of different path finding algorithms, and is described as a function of time and memory use. Here, we consider three different algorithms and compare them using time complexity: Dijkstra's shortest path, K shortest paths and all possible paths.

The worst-case running time for the Dijkstra algorithm on a graph with n nodes and m edges is $O(n^2)$ (Duckham and Worboys, 2004). However, time complexity for the K shortest simple path algorithm for a graph with n nodes and m edges is $O(k(m + nlogn))$ in undirected graphs and $O(kn(m + nlogn))$ in directed graphs (Eppstein, 1999).

The process of finding K shortest path when we have all possible paths saved in a database for a graph of n nodes and m edges includes search for the start node with $O(logn)$ complexity (Duckham and Worboys, 2004), search for end node with $O(logn)$ complexity (Duckham and Worboys, 2004) and sorting the results with $O(klogk)$ complexity in the worst case (Astrachan, 2003); which results in a worst case complexity of $O(2logn+klogk)$, where k is the number of paths between two nodes. However in order to prepare a database of all possible paths we need to perform a pre-processing phase. The pre-processing phase for a complete graph of n nodes and $n(n-1)/2$ edges has a running time of $O(n(n-1)^2)$. This value is the result of finding all possible paths from n sources to $n-1$ destination performing $n-1$ (maximum number of edges in a path) joins in the database.

3.2 Conclusion

The all possible paths algorithm can be considered an umbrella solution that covers other network analysis tasks, such as accessibility analysis and travelling salesperson problem. The algorithm introduced in this paper is compatible with database operations, and the results can be stored in a database for future use, which can improve performance. The results of the all possible paths algorithm can be sorted based on one parameter (e.g., edge weight) or a combination of parameters where more than one parameter can affect the best paths (multi-criteria solution). Since the resulting routes are independent of the values of the edges, this solution provides appropriate results for situations in which the value changes dynamically, as the shortest paths do not need to be re-estimated when one or more weights for and edge changes.

The all possible paths solution can also be utilized with other spatial analysis. For instance if users are looking for paths with some spatial restrictions, all possible paths can provide results that meet those spatial restrictions, by making use of various spatial operators that satisfy the desired restrictions. However, using other path finding algorithms (such as Dijkstra's algorithm), one has to model these restrictions as edge weights and then find the best path.

The complexity analysis also shows that all possible paths can provide the opportunity for fast and efficient computation, assuming that pre-processing phase has been completed. When applied to dynamic networks this approach is clearly more efficient, because there is no need to re-compute the shortest K paths.

4. REFERENCES

O. Astrachan, 2003, Bubble sort: an archaeological algorithmic analysis, SIGCSE Bull., vol. 35, pp. 1-5.

E. W. Dijkstra, 1959, A note on two problems in connexion with graphs. Numerische Mathematik, vol.1, pp. 269-271

S. E. Dreyfus, 1969, An appraisal of some shortest-path algorithms, Operations Research, vol. 17, no. 3, pp. 395-412.

M. Duckham and M. Worboys, 2004, *GIS: A Computing Perspective*, 2nd ed., M. D. Michael Worboys, Ed. CRC PRESS LLC, pp 210-225.

D. Eppstein, 1999, Finding the k shortest paths, SIAM J. Comput., vol. 28, pp. 652-673.

W. Hoffman and R. Pavley, 1959, A method for the solution of the nth best path problem, J. ACM, vol. 6, pp. 506-514.