A Fast and Simple Algorithm for Computing M Shortest Paths in Stage Graph

M. Sherwood
HRH College of Engineering

Laxmi P. Gewali
University of Nevada, Las Vegas, laxmi@cs.unlv.edu

Henry Selvaraj
University of Nevada, Las Vegas, henry.selvaraj@unlv.edu

Venkatesan Muthukumar
University of Nevada, Las Vegas, vm@unlv.nevada.edu

Follow this and additional works at: http://digitalscholarship.unlv.edu/ece_fac_articles

Part of the Applied Mathematics Commons, Controls and Control Theory Commons, Signal Processing Commons, and the Theory and Algorithms Commons

Citation Information
http://digitalscholarship.unlv.edu/ece_fac_articles/307

This Conference Proceeding is brought to you for free and open access by the Electrical & Computer Engineering at Digital Scholarship@UNLV. It has been accepted for inclusion in Electrical and Computer Engineering Faculty Publications by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
A FAST AND SIMPLE ALGORITHM FOR COMPUTING M SHORTEST PATHS IN STAGE GRAPH

We consider the problem of computing \( m \) shortest paths between a source node \( s \) and a target node \( t \) in a stage graph. Polynomial time algorithms known to solve this problem use complicated data structures. This paper proposes a very simple algorithm for computing all \( m \) shortest paths in a stage graph efficiently. The proposed algorithm does not use any complicated data structure and can be implemented in a straightforward way by using only array data structure. This problem appears as a sub-problem for planning risk reduced multiple \( k \)-legged trajectories for aerial vehicles.

1. INTRODUCTION

Computing the shortest collision-free path in the presence of obstacles is a well investigated problem in algorithm analysis and robotics. Rather efficient algorithms are known to solve this problem in two dimensions and the problem is intractable in three dimensions [1-4]. One of the effective approaches for solving the collision-free path problem is to model it as a path problem in weighted graphs. A graph structure called the visibility graph is very effective in capturing shortest paths in two dimensions. In fact, it is known that the shortest collision-free path connecting two points in the presence of two dimensional obstacles is contained in the corresponding visibility graph [7]. However, the concept of visibility graph does not work in three dimensions for capturing the shortest path.

In recent years, researchers have considered the problem of computing \( m \) shortest paths connecting two given nodes in a weighted graph. The objective is to compute the shortest path, second shortest path, third shortest path, up to \( m^{th} \) shortest path in a weighted graph. Solution for \( m \) shortest paths problem has important applications for planning risk reduced trajectories for aerial vehicles [4,5]. In real world applications, several short length collision-free paths are required. In a changing environment, a selected collision-free path may become forbidden due to the emergence of new obstacle. In such situations it would be highly desirable to have alternative short length collision-free paths. All known fast

\[ W(v_{ij}) \]

1 HRH College of Engineering, University of Nevada Las Vegas, Las Vegas, NV, USA
algorithms for computing $m$ shortest paths reported in the literature use very complicated data structures [3]. For reducing asymptotic complexity, researchers are forced to use highly sophisticated data structures which are extremely hard to implement. The main contribution of this paper is the development of a fast and simple algorithm for computing all $m$ shortest paths in a stage graph. The proposed algorithm outputs all $m$ shortest paths in $O(km^3 \log m)$ time using only array data structure and can be very easily implemented.

2. PRELIMINARIES

Consider a stage graph $G(V,E)$ where $V$ is the set of vertices and $E$ is the set of edges. In a stage graph the vertices are partitioned into $k+2$ disjoint sets (stages) $V_0$, $V_1$, $V_2$, ..., $V_k$, $V_{k+1}$. The first ($V_0$) and the last ($V_{k+1}$) stage contain one vertices each, the source vertex $s$ and the target vertex $t$, respectively. The vertices in stage $i$ ($1 \leq i \leq k$) are denoted as $V_{i1}$, $V_{i2}$, $V_{i3}$, ..., $V_{im}$. In a stage graph edges are present only between the vertices of consecutive stages.

For clarity of presentation, we assume without loss of generality that the number of vertices in each stage is the same and is equal to $m$. If not, we can introduce extra vertices in stages containing fewer than $m$ vertices to make the number of vertices equal throughout. We then introduce edges between the extra vertices and the vertices of the previous/next stages with weights equal to infinity. This weight assignment ensures that the shortest route will never go through any of the extra vertices. We could apply Dijkstra's algorithm on the graph to compute the shortest path which would take $O(km^2)$ time. However we can exploit the structure of the graph to obtain a faster algorithm.

Let $w(v_{ij})$ denote the weight of the shortest path from source vertex $s$ to the $j^{th}$ vertex in stage $i$ and let $w(v_{ij}, v_{il, k})$ denote the weight of the edge connecting vertex $v_{ij}$ to vertex...
Suppose we know the shortest path from source vertex \( s \) to all vertices \( v_1, v_2, \ldots, v_m \) in region \( i \). Observe that the weight of the shortest path from \( s \) to \( v_{i+1,j} \) can be written as:

\[
W(v_{i+1,j}) = \min\{W(v_{ij}) + w(v_{ij}, v_{i+1,j}), W(v_{i,j}) + w(v_{i,j}, v_{i+1,j}), \ldots, W(v_{i,m}) + w(v_{i,m}, v_{i+1,j})\}
\]

**Lemma 2:** If we know the weights of the shortest paths from source vertex \( s \) to all vertices in stage \( i \), then we can compute the weights of the shortest paths from \( s \) to all vertices in stage \( V_{i+1} \) in \( O(m^2) \) time where \( m \) is the number of vertices per leg region.

Proof: (Omitted)

By repeating the above lemma on stages from left to right, starting from the first stage, the weight of the shortest path from \( s \) to \( t \) can be constructed in a straightforward way. A formal sketch of the algorithm is as listed below.

**Stage-Step Shortest Path Algorithm**

**Input:** Weighted Stage Graph \( G(V,E) \)

**Output:** Weight and description of the shortest \( k \)-legged path connecting \( s \) and \( t \).

**Step 1:** //Compute the path weights from \( s \) to each vertex in stage 1.

\[
\text{for } (j=1; j=m; j+1) \quad W(v_{1j}) = w(s, v_{1j});
\]

**Step 2:** Compute the path weight to each vertex in all subsequent stages.

\[
\text{for } (i=1; i<k; i+1) \quad \text{for } (j=1; j=m; j+1) \quad W(v_{ij}) = \min\{W(v_{ij}) + w(v_{ij}, v_{i+1,j}), W(v_{ij}+w(v_{i,j}, v_{i+1,j}), \ldots, W(v_{i,m}) + w(v_{i,m}, v_{i+1,j})\};
\]

**Step 3:** //Compute the path weight to \( t \).

\[
W(t) = \min\{W(v_{ij}) + w(v_{ij}, t), W(v_{i,j}) + w(v_{i,j}, t), W(v_{i,j}) + w(v_{i,j}, t), \ldots, W(v_{i,m}) + w(v_{i,m}, t)\};
\]

A simple analysis shows that the above Stage Step Shortest Path algorithm takes \( O(km^2) \) time.

**3. ALGORITHM DEVELOPMENT**

A pair of paths connecting \( s \) to \( t \) is obviously the simplest example of multiple paths. It is therefore useful to consider the construction of the second shortest path in the stage graph. The second shortest path and the first shortest path could be completely disjoint in their interior or could share some edges. It is critical to note that the first shortest path and the second shortest path must have at least one edge not common between them; otherwise both paths will be identical. Hence, if we execute the shortest path algorithm on the graph by removing an "appropriate" edge of the shortest path then the resulting path will be the second shortest path. But it is possible that the second shortest path may contain one or more edges of the first shortest path. We need to somehow modify our description of the first shortest path in order to construct the second shortest path. The first shortest path is denoted by \( p_1 \). The second shortest path is denoted by \( p_2 \).

**Edge Elimination Second Shortest Path Algorithm**

**Step 1:** Run the stage-step algorithm on graph \( G(V,E) \).

**Step 2:** Let \( e_b, e_1, \ldots, e_4 \) denote the four edges of the first shortest path \( p_1 \).

**Step 3:** Successively delete or replace the previously deleted edge (if any) and apply Algorithm again on the resulting graph.

Note: replace the previously deleted edge with the second shortest path \( p_2 \).

**Step 4:** The shortest path from \( s \) to \( t \) is now the second shortest path \( p_2 \).

A straightforward analysis of the edge elimination step shows that if the second shortest path \( p_2 \) is available, then we can construct it correctly.

**Lemma 3:** The edge elimination step correctly constructs the second shortest path.

Proof: (Omitted)

**3.1: COMPUTING ALL SHORTEST PATHS**

As seen above, computing the weights of all shortest paths in the stage graph is difficult because we cannot just use the shortest path algorithm directly. We need to somehow modify our description of the shortest path. Let \( W'_i(v_{ij}) \) denote the weight of the shortest path from \( s \) to \( v_{ij} \).

We can observe that the length of the shortest path weights to \( v_{ij} \) is \( 2m \) edges of the first shortest path and \( m \) possible choices in the positions of the new edges connecting the vertices corresponding to the second shortest path. Let the shortest path to \( v_{ij} \) be given by \( \tilde{P} \). Let the shortest path to all \( v_{ij} \) in stage \( I \) be given by \( \tilde{P}_1 \). The second shortest path to all \( v_{ij} \) in stage \( I \) is given by \( \tilde{P}_2 \) and is available. We can observe that the length of the shortest path weights to \( v_{ij} \) is \( 2m \) choices in the positions of the new edges connecting the vertices corresponding to the second shortest path.
Edges vertex \( s \) to all vertices from \( s \) to all vertices in the first stage, the straightforward way. A formal algorithm takes \( O(km^2) \) time of multiple paths. It is of path in the stage graph. Completely disjoint in their first shortest path and the new them; otherwise both algorithm on the graph by solving path will be the second shortest path. But it is not clear how to identify the appropriate edge. So we try all edges of the first shortest path one by one. The algorithm based on this approach is sketched below.

**Edge Elimination Second Shortest Path Algorithm**

**Step 1:** Run the stage-step algorithm on the stage graph to determine the first shortest path \( P_1 \).

**Step 2:** Let \( e_1, e_2, \ldots, e_k \) denote the edges of \( P_1 \).

**Step 3:** Successively delete one edge at a time \( (e_1, e_2, \ldots, e_k) \) from \( P_1 \) and run Stage Step 2 Algorithm again on the resulting graph to create a pool of potential second shortest paths. Note: replace the previously deleted edge before deleting a new edge.

**Step 4:** The shortest path from the pool of potential second shortest paths is the second shortest path \( P_2 \).

A straightforward analysis of the edge elimination second shortest path algorithm reveals that the second shortest path can be computed in \( O(k^2m^2) \) time, if the weighted graph is available.

**Lemma 3:** The edge elimination second shortest path algorithm constructs the second shortest path correctly.

Proof: (Omitted)

### 3.1: COMPUTING ALL SHORTEST PATHS

As seen above, computing the second and subsequent shortest paths in a \( k \)-legged stage graph is difficult because we only capture the shortest path to any particular vertex. We need to somehow modify our data structure to retain the \( m \) shortest paths into a vertex. Let \( W_q(v; \) \( denote the weight of the \( q \)'th shortest path from source vertex \( s \) to \( j \)'th vertex in stage \( i \).

We can observe that the length of the shortest path to a vertex is the minimum of the sums of the shortest path weights to the prior vertices plus the weights of the corresponding connecting edges. Imagine for a moment that each vertex in \( j \)'th stage is really two co-positioned vertices. Let the shortest path weights to the new vertices vary while the weights of the new edges connecting to vertex \( v_{i+,j} \) remain the same. The weights to the new vertices correspond to the second shortest path weights to the original vertex. Our new graph results in \( 2m \) possible choices for the shortest path to vertex \( v_{i+,j} \). It is of interest to note that while in the original graph, the second shortest path to vertex \( v_{i+,j} \) might not have been one of the \( m \) possible choices, in the new graph, the second shortest path must be...
contained in the $2m$ possible choices. Adding an additional vertex representing the third shortest path to each vertex in stage $i$ would result in $3m$ possible choices for the shortest path to vertex $v_{i,1,i}$ and these $3m$ possible choices must include the three shortest paths to vertex $v_{i-1,i}$. We can continue adding vertices until there are $m$ vertices per vertex in stage $i$ for a total of $m^2$ possible shortest paths to vertex $v_{i-1,i}$ which must contain the $m$ shortest paths to vertex $v_{i-1,i}$.

Suppose we know the $m$ shortest paths from source vertex $s$ to all vertices $v_{i,1}$, $v_{i,2}$, ..., $v_{i,m}$ in $i^{th}$ stage. Observe that the weight of the $q^{th}$ shortest path from $s$ to $v_{i,1}$ can be written as:

$$W_q(v_{i,1}) = \min_q\{W_1(v_1) + w(v_1, v_{i,1}), W_2(v_1) + w(v_1, v_{i,1}), ..., W_m(v_1) + w(v_1, v_{i,1})\},$$

$$\ldots,$$

$$W_m(v_1) + w(v_1, v_{i,1}), W_2(v_1) + w(v_1, v_{i,1}), ..., W_m(v_1) + w(v_1, v_{i,1})\}.$$}

where, $\min_q[a_1, a_2, ..., a_m]$ returns the $q^{th}$ smallest of the arguments.

By using the above relation, the weights of the $m$ shortest paths from $s$ to all vertices in stage $i+1$ can be found by scanning the weights of edges between stages $i$ and $i+1$ and adding them to the weights of the $m$ shortest paths from $s$ to stages $i$. The time to compute weights $(W_{1(i-1,1)}, W_{2(i-1,1)}, ..., W_{m(i-1,1)}, W_{1(i-1,2)}, W_{2(i-1,2)}, ..., W_{m(i-1,2)}, ..., (W_{1(i+1, m)}, W_{2(i+1, m)}, ..., W_{m(i+1, m)})$ is bounded by the time to sort the number of edges connecting vertices in $V_i$ to vertices in $V_{i+1}$ multiplied by the number of weights per vertex. Each vertex is taken as an array of size $m$ to record entries for $m$ shortest paths. Each element in the this array contains the weight and index of the $q^{th}$ shortest path from source vertex $s$ to vertex $v_{i,j}$. The following algorithm is a modified version of the stage-step algorithm and computes all $m$ shortest paths. In the algorithm the function $ordered(a_1, a_2, a_3 ..., a_m)$ returns the $q^{th}$ smallest of the arguments. The array paths[1...m^2] stores the shorted weights of $m^2$ shortest paths coming from the previous stage.

### 3.2 STAGE-STEP ALGORITHM FOR M SHORTEST PATHS

**Input:** Weighted Stage Graph $G(V, E)$

**Output:** Weight and description of the shortest $k$-legged path connecting $s$ and $t$.

**Step 1:** //Compute the path weights from $s$ to each vertex in stage $V_i$.

for ($j=1; j<=m; j+1$) $W_i(v_{i,j}) = w(s, v_{i,j});$

**Step 2:** //Compute weights of $m$ shortest paths weights to subsequent stages.

for ($i=1; i<k; i+1$) {

for ($q=1; q<=m; q+1$) $W_i(v_{i,j}) = ordered(W_i(v_{i,j}) + w(v_{i,j}, v_{i-1,j});$

...}$

Theorem 2: Given a stage graph, The time to compute all $m$ shortest paths is $O(m \log m)$ time.

Proof: (Omitted)

### 4. CONCLUSION

We presented a fast and easy to program algorithm. The sketched algorithm is a modified version of the stage-step algorithm and computes all shortest paths. We also showed how the proposed technique can be applied to triangulation.

**REFERENCES**

vertex representing the third
the shortest
vertices per vertex in stage i
must contain the m shortest
ll vertices v_{1,i}, v_{2,i}, ..., v_{m,i}
> v_{i+1,j}

\)

Step 3: //Compute the m smallest path weights to t.
paths[1...m]

\)

Theorem 2: Given a stage graph, all m shortest k-legged paths can be computed in \(O(km^3 \log m)\) time.
Proof: (Omitted)

4. CONCLUSION

We presented a fast and easy to implement algorithm for computing all m shortest paths in a stage graph. The sketched algorithm can be directly coded in C/C++ or any other high level programming language. It would be interesting to further investigate this approach for computing all shortest paths on general graph. If a general graph can be approximated somehow as a k stage graph then the application of the proposed algorithm can be used to generate approximate solution for m shortest paths problem. It would also be interesting how the proposed technique works for some planar graphs such as the Delaunay triangulation.

REFERENCES