# Finding Next-To-Shortest Paths in a Graph

I. Krasikov and S. D. Noble
Department of Mathematical Sciences
Brunel University
Kingston Lane
Uxbridge
UB8 3PH*

February 16, 2008

**Abstract**

We study the problem of finding the next-to-shortest paths in a graph. A next-to-shortest $(u, v)$-path is a shortest $(u, v)$-path amongst $(u, v)$-paths with length strictly greater than the length of the shortest $(u, v)$-path. In constrast to the situation in directed graphs, where the problem has been shown to be NP-hard, providing edges of length zero are allowed, we prove the somewhat surprising result that there is a polynomial time algorithm for the undirected version of the problem.

**Keywords:** Graph algorithms, computational complexity, shortest paths.

## 1 Introduction

Solutions to the problem of finding the shortest path between two vertices (or all pairs of vertices) in both undirected and directed graphs are well-known [1]. Finding the $K$ shortest paths between two vertices has also been well-studied, in for instance [2, 3]. Problems involving finding a path between two vertices of length strictly greater than the length of the shortest such path have received much less attention due to the fact that in directed graphs, when we allow edges of length zero, the problem has been shown to be NP-hard [4]. Here we consider the problem of finding next-to-shortest paths in an undirected graph. We give a polynomial time algorithm for the undirected version of the problem when all edge lengths are strictly positive, thus answering a problem raised in [4] and suggesting that the undirected version of the problem is significantly easier than the directed version.

*{mastiik,mastsdn}@brunel.ac.uk

Our graph theoretical notation is standard and we restrict our graphs to being simple, that is, having no loops or multiple edges. For the most part, we will deal with weighted graphs consisting of a graph $G = (V, E)$ and a function $l : E \to \mathbb{Q}^+$ giving the length of each edge. Note that by $\mathbb{Q}^+$ we mean the set $\{x : x \in \mathbb{Q}, x > 0\}$ so we do not allow edges with length zero. Given a set, $A$, of edges we define $l(A)$ in the obvious way, that is

$$l(A) = \sum_{e \in A} l(e).$$

We use $n(G)$ and $m(G)$ to denote, respectively, the number of vertices and edges of $G$. Whenever the context is clear we just use $n$ and $m$. Since we need to make frequent use of walks and paths, we give their standard definition. A *walk* is an ordered list of vertices and edges $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$ such that for $1 \leq i \leq k$, the endpoints of $e_i$ are $v_{i-1}$ and $v_i$. Sometimes we will deal with directed graphs in which case we require $e_i = (v_{i-1}, v_i)$. Depending on the context we may just specify a walk by giving either an ordered list of vertices or of edges. A *path* is a walk for which the vertices $v_0, \ldots, v_k$ are distinct. A $(u, v)$-path is a path for which $v_0 = u$ and $v_k = v$. We define a $(u, v)$-walk in similar way.

Given a weighted graph, $G$, the next-to-shortest $(u, v)$-path, is the shortest $(u, v)$-path, amongst those $(u, v)$-paths having length strictly greater than the length of the shortest $(u, v)$-path. If no such path exists, we say that the next-to-shortest $(u, v)$-path has length $\infty$.

# 2 Main Result

This section is devoted to a proof of our main result.

**Theorem 1** *There is a polynomial time algorithm which inputs an undirected graph $G = (V, E)$, a length function $l : E \to \mathbb{Q}^+$ and specified vertices $u$ and $v$, and finds a next-to-shortest $(u, v)$-path.*

The main idea of the proof is to consider the set of edges that occur in a $(u, v)$-path and the direction in which they are traversed. Given a weighted graph $G = (V, E)$ and specified vertices $u$ and $v$, we define the shortest path digraph $D_G(u, v)$ to be the digraph $D = (V, A)$, where the arc $(x, y)$ is in $A$ if there is a shortest length $(u, v)$-path of the form $u, v_1, \ldots, v_i, x, y, v_{i+3}, \ldots, v_m, v$.

We begin with a simple, preliminary and probably well-known lemma.

**Lemma 2** *For all $w$, any $(u, w)$-walk in $D_G(u, v)$ is a shortest $(u, w)$-path in $G$.*

**Proof:** Suppose not and let $v_0 = u, v_1, \ldots, v_k = w$ be a walk in $D_G(u, v)$ with the fewest number of arcs amongst those walks that are not shortest paths. Denote this walk by $W$. Since $(v_{k-1}, v_k)$ is an arc of $D_G(u, v)$, there is a shortest $(u, v)$-path $P$ passing through $(v_{k-1}, v_k)$. Because $W$ is a walk with the fewest

arcs that is not a shortest path, deleting the last arc from $W$ gives a shortest $(u, v_{k-1})$-path $W'$. So if we replace the portion of $P$ between $u$ and $v_{k-1}$ by $W'$, we obtain a $(u, v)$-walk that is no longer than $P$. Since all edge lengths are strictly positive, this must be a shortest $(u, v)$-path and the portion of this path between $u$ and $v_k$ is exactly $W$. Hence $W$ must be a shortest $(u, w)$ path.

□

We say that an arc $(x, y)$ is a *forward* arc (of $D_G(u, v)$) if $(x, y) \in D_G(u, v)$ and a *backward* arc if $(y, x) \in D_G(u, v)$. Lemma 2 implies that an arc cannot be both forward and backward.

The next lemma is a simple consequence of minimum cost flow techniques and is presumably well-known but we give a sketch proof here since it is a key part of the main theorem.

**Lemma 3** *Given a weighted, undirected graph, $G$, and specified vertices, $u$, $v$ and $w$, there is a polynomial time algorithm to find the shortest length $(u, v)$-path passing through $w$.*

**Proof:** Form a network $N$ from $G$ by replacing each edge by two arcs directed in opposite directions and adding a new vertex $t$ together with the arcs $(u, t)$ and $(v, t)$. Assign a capacity of $\infty$ to all the arcs and to the vertices $t$ and $w$, and one to every other vertex. Give the arcs joined to $t$ zero cost and give each other arc cost equal to the length in $G$. Finding the shortest path in $G$ from $u$ to $v$ passing through $w$ corresponds to finding a minimum cost flow of volume two from $w$ to $t$ in $N$. This can be done with two iterations of the augmentation step of the Ford-Fulkerson maximum flow algorithm or equivalently two shortest path computations and hence gives an $O(n^2)$ algorithm, [1].

□

We now give the proof of the main theorem.

**Proof of Theorem:** We first verify that $D_G(u, v)$ can be constructed in polynomial time. Let $u_w$ denote the length of the shortest $(u, w)$-path. Using Dijkstra's algorithm we can compute $u_w$ for each $w$ in time $O(n^2)$. The arc $(x, y)$ is present in $D_G(u, v)$ if and only if

$$u_y = u_x + l(\{x, y\}).$$

Thus $D_G(u, v)$ can be constructed in time $O(n^2)$.

Lemma 2 shows that any walk made up of forward arcs of $D_G(u, v)$ is a shortest $(u, v)$-path. We first find the shortest path containing an arc that is neither forward or backward. The shortest $(u, v)$-path passing through $\{x, y\}$ (in either direction) can be found by adding vertex $w$ to subdivide $\{x, y\}$ and applying Lemma 3. The shortest such path (if it exists) can therefore be found by applying Lemma 3, $O(m)$ times.

The other type of $(u, v)$-path that needs to be considered is one containing only forward and backward arcs but including at least one backward arc. Such a

path must contain a forward arc $(x, y)$ followed by a backward arc $(y, z)$. Given forward arcs $(x, y)$ and $(z, y)$, form $G'$ from $G$ by deleting all edges adjacent to $y$ except $\{x, y\}$ and $\{y, z\}$. Applying Lemma 3 to $G'$ with $w = y$ will find the shortest path containing one of $(x, y)$ and $(z, y)$ as a forward arc and the other as a backward arc. Using the above procedure for each pair of forward arcs of the form $(x, y)$ and $(z, y)$ will give the shortest path of this type and require $O(nm)$ applications of Lemma 3.

The next-to-shortest $(u, v)$-path is the shortest path found in either of the two steps of the algorithm and can be found using $O(nm)$ shortest path computations, that is in $O(n^3m)$ steps overall.

□

# 3 Conclusion

We have shown that next-to-shortest paths can be found in undirected graphs with strictly positive edge lengths in time $O(n^3m)$. The complexity status of the next-to-shortest paths problem in directed graphs when the edge lengths are required to be strictly positive is open, although we suspect that it is NP-complete. Similarly we do not know the complexity of the next-to-shortest paths problem in undirected graphs when edge lengths of zero are allowed.

A further interesting open question is to consider the structure of next-to-shortest paths and whether any analogue of the Bellman optimality equations hold. The position seems far from obvious.

# References


[1] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2001.

[2] D. Eppstein. Finding the $k$ shortest paths. *SIAM Journal of Computing*, 28(2):652–673, 1999.

[3] V. M. Jiménez and A. Marzal. Computing the $k$ shortest paths: a new algorithm and an experimental comparison. In J. S. Vitter and C. D. Zaroliagis, editors, *Algorithm Engineering: 3rd International Workshop, WAE'99*, volume 1668 of *Lecture Notes in Computer Science*, pages 15–29, Berlin, Heidelberg, 1999. Springer-Verlag.

[4] K. N. Lalgudi and M. C. Papaefthymiou. Computing strictly-second shortest paths. *Information Processing Letters*, 63:177–181, 1997.