Vickrey Prices and Shortest Paths: What is an edge worth?

John Hershberger

Mentor Graphics Corp.
8005 SW Boeckman Road
Wilsonville, OR 97070, USA
john_hershberger@mentor.com

Subhash Suri*

Computer Science Department
University of California
Santa Barbara, CA 93106, USA
suri@cs.ucsb.edu

Abstract

We solve a shortest path problem that is motivated by recent interest in pricing networks or other computational resources. Informally, how much is an edge in a network worth to a user who wants to send data between two nodes along a shortest path? If the network is a decentralized entity, such as the Internet, in which multiple self-interested agents own different parts of the network, then auction-based pricing seems appropriate. A celebrated result from auction theory shows that the use of Vickrey pricing motivates the owners of the network resources to bid truthfully. In Vickrey's scheme, each agent is compensated in proportion to the marginal utility he brings to the auction. In the context of shortest path routing, an edge's utility is the value by which it lowers the length of the shortest path—the difference between the shortest path lengths with and without the edge. Our problem is to compute these marginal values for all the edges of the network efficiently. The naïve method requires solving the single-source shortest path problem up to n times, for an n-node network. Our main result is that, in an undirected graph, the Vickrey prices for all the edges can be computed in the same asymptotic time complexity as one single-source shortest path problem.

1 Introduction

Shortest paths are fundamental in many areas of computer science, operations research, and engineering. Their applications include network and electrical routing, transportation, robot motion planning, critical path computation in scheduling, etc. In addition,

*Subhash Suri's research on this paper was partially supported by National Science Foundation grants CCR-9901958 and ANI-9813723.

shortest paths also provide a unifying framework for many optimization problems such as knapsack, sequence alignment in molecular biology, inscribed polygon construction, and length-limited Huffman-coding, etc. (Eppstein [5] is a good reference for shortest paths and their applications.) Most complex applications of the shortest path problem, however, require more than just the calculation of a single shortest path. In some applications, the desired path might be subject to additional constraints that are hard to quantify. In others it might be useful to examine not just *the* shortest but a larger set of "short paths." It some applications, it is desirable to see how the shortest path is influenced by various system parameters, through a "sensitivity analysis." Our problem belongs to this last category. We wish to determine, for each edge e in a graph, what effect e's deletion has on the shortest path between two given nodes.

Our problem is motivated by recent interest in pricing networks and computing resources, which in turn is prompted by the prominent role the Internet has come to play in our lives. One of the distinguishing characteristics of the Internet is that it involves interaction among multiple (often very many) self-interested participants. These participants (organizations, people, computers, software), called "agents" in the AI terminology, cannot be trusted to follow the rules of a protocol, especially if deviating from the protocol is beneficial to the agent. Thus, unlike traditional distributed computing protocols, a protocol for these new settings must be designed explicitly to account for willing manipulation by the users. We briefly describe an example that helps illustrate this phenomenon.

One of the most famous distributed protocols on the Internet is the Transmission Control Protocol (TCP), implemented on each host on the Internet. An important feature of this protocol is its *congestion control* mechanism. The protocol uses packet loss as an indication of network congestion, and is designed to reduce the sending host's transmission rate (using a fairly aggressive exponential back-off). It then gradually increases the transmission rate until another sign of congestion is detected. This cycle of increasing and then decreasing the transmission rate allows the protocol to discover and utilize whatever bandwidth is available between two communicating hosts, while at the same time sharing the overall resource among many such pairs.

TCP is *self-regulating*, meaning that it assumes that individual hosts will respond to congestion exactly as the designers of the protocol intended. But a self-interested host (agent) has motivation *not* to decrease its sending rate in the hope that *others* will reduce their rate, eliminating the congestion, while he can continue to enjoy the higher rate. In one extreme case, no one follows the TCP congestion rules, and the system crashes; in another extreme case, a few misbehaving users enjoy an unfair share of the network resources, while the rule-abiding majority of users suffer. Because a protocol like TCP is easily manipulated, many researchers have proposed game-theoretic and price-based mechanisms to share bandwidth and other network resources [6, 16, 22].

In this context, a natural economic question is this: how much is an edge in a network worth to a user who wants to send data between two nodes along a shortest path? If the network is a decentralized entity, such as the Internet, in which multiple self-interested agents own different parts of the network, then an auction is often the best mechanism to determine the utility of various network elements. A celebrated result from auction theory shows that the use of Vickrey pricing motivates the agents to bid truthfully. In Vickrey's scheme, each agent is compensated in proportion to the marginal utility he brings to the auction. The insight of Vickrey is that although agents have an incentive to lie about their costs in the hope of receiving larger compensation from the network, making an edge's payment depend only on the declarations of *other* agents eliminates this manipulative element.[1]

Suppose we are interested in discovering the shortest path from node x to node y in an undirected graph G, whose edges are owned by self-interested agents. We assume that agents bid on *individual* edges—that is, either each agent owns at most one edge, or if an agent owns multiple edges, he bids on each independently. (We do not consider the setting in which an agent can make strategic bids on *subsets* of edges. In such *bundle auctions*, even determining the winning bids is NP-complete, but under a restricted setting Bikhchandani et al. [3] solve the Vickrey payment problem using linear programming.) The cost of edge e is the bid by the agent who owns e; we assume that all edge costs are non-negative. The network employs the Vickrey pricing mechanism to elicit agents' *true* preferences (costs). The payment p^e made to an edge e is determined as follows:

$$p^e = \begin{cases} d(x,y;\ G \setminus e) - d(x,y;\ G|_{e=0}) & \text{if } e \text{ belongs to the shortest path} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

That is, if edge e does not belong to the shortest path in G, then its agent receives zero payment. Otherwise, the payment to e is the difference between the cost of the shortest path without e, and the cost of the shortest path assuming e is free.

This formulation is from Nisan and Ronen [15], who posed the following question: what is the computational complexity of determining all the Vickrey payments? The straightforward method requires computing the x–y shortest paths up to n times: once in G, and once in $G \setminus e$, for each e that belongs to the shortest path in G. The payment function expressed in equation (1) requires two shortest path computations, but the second term can easily be deduced from the shortest path distance $d(x,y;\ G)$, for each e that belongs to the shortest path.

We partially answer the question of Nisan and Ronen, by showing that in an undirected graph, the Vickrey payments to all the agents can be computed in essentially the same time bound as one single-source shortest path computation. Our algorithm builds two shortest path trees, one based on x and the other based on y, and computes the payment term $d(x,y;\ G \setminus e)$ for each edge e by combining parts of these trees. The total time

[1] The Vickrey mechanism is a generalization of the well-known sealed bid *second price auction*, in which an object is sold to the highest bidder, but the winner pays a price equal to the runner-up's bid. This auction protocol is known to be truthful, in that a rational agent's best bidding strategy is to bid his true valuation. Thus, in a distributed network where network links belongs to rational, self-interested agents, Vickrey pricing elicits truthful responses from agents, leading to economic efficiency in the network—the shortest paths use agents with lowest costs (even though actual costs are private, not public, information). This truthfulness, however, does come at a cost: the mechanism may need to subsidize the agents. For a proof of the Vickrey mechanism's truthfulness, see [13].

complexity is $O(m + n \log n)$, which matches the best bound known for computing one single source shortest path tree.

In the conference version of this paper [7], we erroneously claimed that our algorithm works for *directed graphs* as well. We later discovered a bug in our proof. The error lies in the claim immediately following Lemma 2 in the conference proceedings version of the paper. Specifically, we said that "… if (u, v) is the single edge of $path(x, y;\ G \setminus e)$ in $E(V_x, V_y)$, then $path(v, y;\ G \setminus e) = path(v, y)$." This claim is incorrect and consequently our method of computing the replacement paths for directed graphs breaks down.

In fact, it turns out that the *directed graph* version of the problem is *provably* harder. We have recently shown [8] that there is an $\Omega(m\sqrt{n})$ time lower bound for any *path comparison* algorithm that solves the replacement paths problem for a directed graph with n nodes, m edges, whenever $m = O(n\sqrt{n})$.

2 Related work

Researchers in multi-agent systems, or distributed AI, have studied cooperation and competition among "software agents." Many of these papers use ideas from mechanism design to analyze strategies and responses of these agents in negotiations and resource allocations [17, 18, 23]. Others use market-based ideas to solve distributed computational problems [25, 26]. Researchers in networking have proposed game-theoretic techniques to deal with congestion control in the Internet [6, 9, 16, 22].

Our paper is motivated by the algorithmic mechanism design paper of Nisan and Ronen [15]. They investigate computational complexity and algorithmic issues in mechanism design, and raise some intriguing problems. Specifically, they asked the question that forms the basis of our work: Can the payment functions of the Vickrey mechanism be computed faster than n invocations of the optimization problem? The payment function computation for the network routing problem is equivalent to the following: given a graph G, and two specified nodes x and y, determine, for each edge e in the graph, the effect on $d(x, y)$ of deleting e.

Our problem is related to the topic of "sensitivity analysis" in operations research. In sensitivity analysis, the goal is to determine the robustness of a solution: how much the system parameters can be perturbed before the solution changes. For instance, the sensitivity analysis of the minimum spanning tree requires computing for each edge e the amount $\delta(e)$ by which the cost of e must change before the minimum spanning tree changes; $\delta(e)$ is positive if e is part of the MST, and negative otherwise. Tarjan [24] presents an $O(m\alpha(m, n))$ time algorithm for calculating $\delta(e)$ for all edges of a graph with n nodes and m edges, where α is a functional inverse of Ackermann's function. Tarjan also presents a similar result for performing the sensitivity analysis of a shortest path tree. In our problem, however, we are not interested in computing the cost threshold of an edge, but rather in deleting an edge, and then finding the new shortest path. Our problem also has some similarity to the k-shortest paths problem studied by Eppstein [5], but requires different techniques.

Bikhchandani et al. [3] and Schummer and Vohra [21] have considered general auction settings where the Vickrey payments correspond to *dual variables* in a linear program. Their results depend on a combinatorial condition, which they call the "agents are substitutes" condition. It turns out that the minimum spanning tree problem and the assignment problem satisfy the "agents are substitutes" condition, and therefore the Vickrey payments for those problems can be determined efficiently. However, the "agents are substitutes" condition does not hold for the shortest path problem, so the methods of [3] and [21] do not apply to our setting.

We recently became aware of the work of Nardelli, Proietti, and Widmayer [14], who considered essentially the same problem, under the guise of determining the most vital node of a shortest path. Their work is in turn based on earlier work by Malik, Mittal, and Gupta [12], Ball, Golden, and Vohra [1] and Bar-Noy, Khuller, and Schieber [2]. Our results are equivalent in the sense that our algorithm can be used to determine the most vital node, and their algorithm can be used to compute the Vickrey payments for the edges, within the $O(m + n \log n)$ time bound. However, we believe that our algorithm is considerably simpler than the one presented in [14].

3 Shortest path preliminaries

We assume that our network is modeled by an undirected graph $G = (V, E)$, with $|V| = n$ and $|E| = m$. Each edge $e \in E$ has an associated cost $c(e)$, which is a non-negative real number.

A path in G is a sequence of edges, such that consecutive edges share a common vertex, and each vertex is incident to at most two path edges. The total cost of a path in G is the sum of the costs of the edges on the path. The *shortest path* between two vertices a and b, denoted by $path(a, b)$, is the path joining a to b, assuming one exists, that has minimum cost. The distance between a and b, denoted $d(a, b)$, is the length of $path(a, b)$, or infinity if no path exists.[2]

We denote shortest paths and distances when an edge e has been removed from the graph G by the notations $path(a, b;\ G \setminus e)$ and $d(a, b;\ G \setminus e)$. The distance in the full graph $d(a, b)$ is shorthand for $d(a, b;\ G)$; likewise $path(a, b)$ is shorthand for $path(a, b;\ G)$.

There are two distinguished vertices x and y in the graph, called the *source* and the *target* vertices. The shortest path joining them is $path(x, y) = (v_1,\ v_2,\ \ldots,\ v_k)$, where $v_1 = x$ and $v_k = y$. Recall that we want to compute, for each $i \in \{1, \ldots, k-1\}$, the length of the shortest path from x to y that does not use the edge $e_i = (v_i, v_{i+1})$, which we call the *x-y distance omitting e_i*. This is precisely the term $d(x, y;\ G \setminus e_i)$ in the payment function.

We can analyze the shortest paths from x to y in terms of the set of edges crossing a *cut*. In later sections we will choose the cut according to the structure of the graph,

[2]For convenience we assume that all path lengths are distinct, so the shortest path between any two vertices is unique. This condition is easy to enforce by a symbolic perturbation of the edge costs.

but for now let us simply consider any partition of the vertex set V into two sets V_x and V_y such that $x \in V_x$ and $y \in V_y$. The set of edges crossing the cut is denoted by $E_{\text{cut}} = E(V_x, V_y)$. Each edge $(u, v) \in E_{\text{cut}}$ has $u \in V_x$ and $v \in V_y$. Any path from x to y must include at least one edge from E_{cut}. Therefore, we can express our problem as follows: For each $e_i = (v_i, v_{i+1})$ and some cut (V_x, V_y) possibly dependent on e_i, compute

$$d(x, y;\ G \setminus e_i) = \min_{\substack{(u,v) \in E_{\text{cut}} \\ (u,v) \neq e_i}} \begin{pmatrix} d(x, u;\ G \setminus e_i) + \\ c(u, v) + \\ d(v, y;\ G \setminus e_i) \end{pmatrix} \tag{2}$$

In Section 4 we apply this expression to compute the x-y distance omitting each edge of $\mathit{path}(x, y)$ in the special case in which G is undirected and all the vertices of V lie on $\mathit{path}(x, y)$. In Section 5 we solve the problem for general undirected graphs.

4 Path Graphs

We illustrate our ideas by solving the special case in which $\mathit{path}(x, y)$ includes all the vertices of V. The additional structure in this case makes it particularly easy to compute the distances in $G \setminus e$ required by equation (2). We define the cut (V_x, V_y) based on the natural partition of $\mathit{path}(x, y) = (v_1, v_2, \ldots, v_n)$ induced by the removal of the edge $e_i = (v_i, v_{i+1})$. We choose $V_x = \{v_1, \ldots, v_i\}$ and $V_y = \{v_{i+1}, \ldots, v_n\}$. To simplify the notation, we define $E_i = E(V_x, V_y)$ for e_i. See Figure 1.

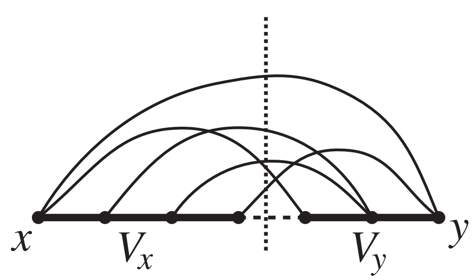

Figure 1: The edges of E_i cross the dashed vertical line.

For a given edge $e_i = (v_i, v_{i+1})$ to be removed, consider a cut edge $(u, v) \in E_i$. Because $\mathit{path}(x, u)$ is contained in V_x, $d(x, u) = d(x, u;\ G \setminus e_i)$. Likewise, because $\mathit{path}(v, y)$ is contained in V_y, $d(v, y) = d(v, y;\ G \setminus e_i)$. These distances are simply the lengths of the subpaths of $\mathit{path}(x, y)$ connecting each point to x and y. Thus equation (2) reduces to

$$d(x, y;\ G \setminus e_i) = \min_{\substack{(u,v) \in E_i \\ (u,v) \neq e_i}} d(x, u) + c(u, v) + d(v, y). \tag{3}$$

To compute the shortest x-y path omitting each edge of $\mathit{path}(x, y)$ efficiently, we evaluate equation (3) for each $e_i = (v_i, v_{i+1})$ in sequence from $i = 1$ to $n - 1$. For a given

i, we minimize the quantity $d(x, u) + c(u, v) + d(v, y)$ over all $(u, v) \in E_i \setminus e_i$. But the difference between E_i and E_{i+1} is easy to compute: it consists of exactly those edges with one endpoint at v_{i+1}. To produce E_{i+1} from E_i, we add to E_i all edges whose left endpoint is v_{i+1}, and we remove from E_i all edges whose right endpoint is v_{i+1}.

To formalize this, let $\mathit{left}(e)$ and $\mathit{right}(e)$ be the indices of the endpoints of e in $\mathit{path}(x, y)$, with $\mathit{left}(e) < \mathit{right}(e)$. In later sections, when the endpoints of e do not necessarily lie on $\mathit{path}(x, y)$, we will redefine $\mathit{left}(e)$ and $\mathit{right}(e)$ to be indices such that $e \in E_i$ for all $\mathit{left}(e) \leq i < \mathit{right}(e)$. We perform the following algorithm:

Path Algorithm

1. Let L and R be k-element arrays whose elements are sets of edges, initially empty. Let Q be a priority queue of (weight, edge) pairs, indexed by weight, initially empty.
2. For each $e \in E \setminus \mathit{path}(x, y)$

   If $\mathit{left}(e) < \mathit{right}(e)$, put e into $L[\mathit{left}(e)]$ and $R[\mathit{right}(e)]$.
3. For $i = 1$ to $k - 1$
   (a) For each $e = (u, v) \in L[i]$
   Insert (w, e) into Q, with weight $w =$
   $d(x, u;\ G \setminus e_i) + c(u, v) + d(v, y;\ G \setminus e_i)$.
   (b) Remove from Q all (w, e) pairs with $e \in R[i]$.
   (c) Report the minimum weight in Q as the x-y distance omitting e_i.

At step i, Q contains the edges of $E_i \setminus e_i$, and so the x-y distance omitting e_i is correctly computed. Only the priority queue operations take non-constant time. A naïve priority queue implementation gives a running time of $O(m \log m)$.

We can improve this time complexity by using a Fibonacci heap for Q with at most $k - 1$ nodes, numbered from 2 to k [4]. The j'th node stores the minimum-weight edge in the current cut that belongs to $R[j]$. We initialize the Fibonacci heap to have $k - 1$ nodes, each with weight ∞. In step 3a, for each $e \in L[i]$, we compare its weight w with the weight of the heap node with index $j = \mathit{right}(e)$; if e has the lesser weight, we perform a DecreaseKey operation on node j and reset its edge to be e. (We do not need to do anything if e has the greater weight, because the edge stored at node j will be in the cut just as long as e.) In step 3b, we delete node i from Q—the minimization in step 3a means that this node is the representative of all the edges in $R[i]$. Thus the heap contains only a subset of the edges that would be stored in a naïve implementation of Q, but it is guaranteed to contain the minimum-weight element of the cut set.

The Fibonacci heap implementation performs $O(m)$ DecreaseKey operations, but only $O(n)$ inserts, deletes, and FindMin operations. Only the delete operation takes $O(\log n)$

time in the Fibonacci heap; the other three operations take $O(1)$ amortized time apiece. Thus, the total time spent on priority queue operations using the Fibonacci heap implementation is $O(n \log n + m)$.

5 General Undirected Graphs

In a general undirected graph, not all vertices lie on $path(x, y)$. This means that the structure of the shortest paths from x to other vertices is unrelated to the structure of the shortest paths from those other vertices to y. The *shortest path tree* with source x is the union of all the shortest paths from x to other vertices in V. Since we assume uniqueness of shortest paths, this union of paths is indeed a tree. Each vertex v has a unique parent u in the tree; $path(x, v)$ is obtained by concatenating $path(x, u)$ with the edge (u, v). Let us denote the shortest path tree with source x by X.

We can also define a shortest path tree with sink y, which we denote by Y. This is the union of all shortest paths from vertices in V to the destination y. (Since we are assuming G is undirected, Y is identical in structure to the shortest path tree with source y. For directed graphs, this is not true.) The shortest path trees X and Y can be computed in $O(n \log n + m)$ time using Dijkstra's algorithm and Fibonacci heaps [4], or in $O(m \log n)$ time using simple data structures.

In the case of a path graph, as discussed in Section 4, $X = Y = path(x, y)$, and the removal of an edge e splits X and Y into identical components. However, for general undirected graphs, this is not true. For example, in Figure 2, the vertices in the upper branch of the graph lie in opposite components of $X \setminus e$ and $Y \setminus e$.

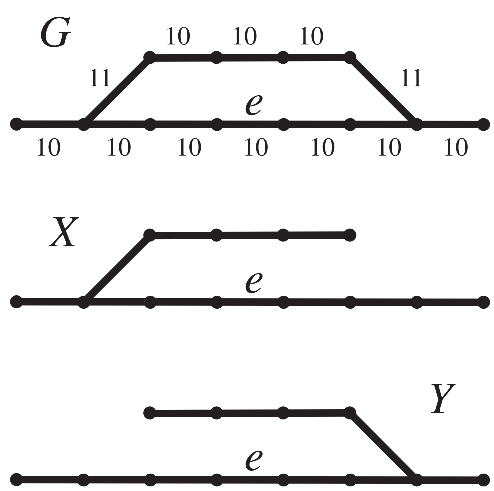

Figure 2: X and Y have different structures.

To compute the x-y distance omitting each edge $e_i = (v_i, v_{i+1})$ in $path(x, y) = (v_1, \ldots, v_k)$, we define the cut (V_x, V_y) based on the shortest path tree X. Because $path(x, y)$ is contained in X, the removal of e_i splits X into two components; we choose the component containing x to be V_x, and the complement to be V_y. To be more specific in the determination of V_x and V_y, we assign vertices of V to *blocks* based on their position in the shortest path tree X. If we delete all the edges of $path(x, y)$ from X, the vertices connected to v_i in the remaining forest form block B_i. If $u \in B_i$, we define $block(u) = i$. Thus for a given edge $e_i = (v_i, v_{i+1}) \in path(x, y)$, $V_x = \cup_{j=1}^{i} B_j$, and $V_y = \cup_{j=i+1}^{k} B_j$. See Figure 3.

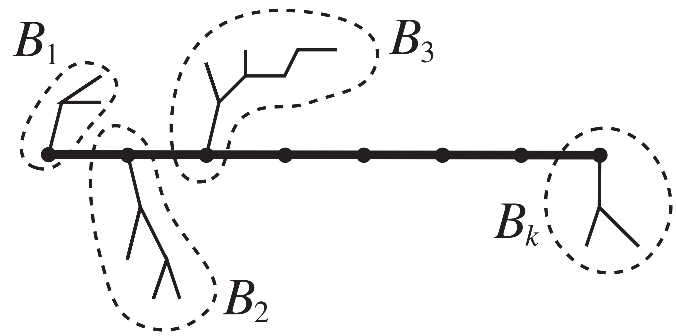

Figure 3: Removing $path(x, y)$ from X defines blocks B_i.

Consider computing the x-y distance omitting e_i according to equation (2). For all $u \in V_x$, $path(x, u)$ is contained in V_x by definition, so we have $d(x, u) = d(x, u;\ G \setminus e_i)$. Because the partition of X induced by deleting e_i is not the same as the corresponding partition of Y (Figure 2), it is not obvious that $path(v, y)$ is contained in V_y for all $v \in V_y$. Nevertheless, it turns out that $path(v, y)$ does not use e_i, which is what we need.

Lemma 1 *Let v be a vertex in $V_y = \cup_{j=i+1}^{k} B_j$ for some $e_i = (v_i, v_{i+1})$. Then $d(v, y) = d(v, y;\ G \setminus e_i)$.*

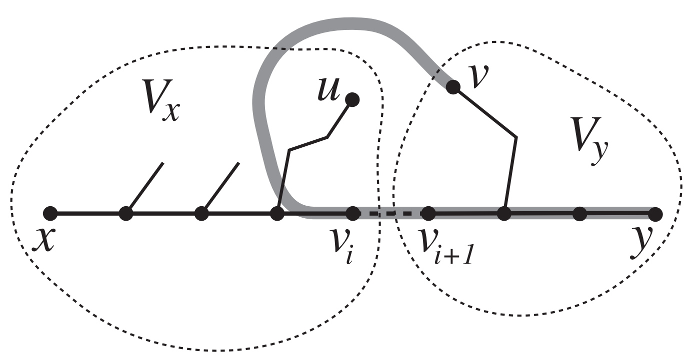

Figure 4: Unlike the shaded path, $path(v, y)$ cannot include e_i, because $path(v_{i+1}, v)$ is contained in V_y.

Proof: The proof is by contradiction. Suppose that $path(v, y)$ uses the edge e_i. It must traverse e_i in the forward direction, from v_i to v_{i+1}, because the shortest path from v_{i+1} to y is fully contained in V_y and does not traverse e_i. Then $path(v, y)$ is the concatenation of $path(v, v_{i+1})$ with $path(v_{i+1}, y)$, and the first subpath contains vertex v_i (which is a vertex of V_x) in its interior (see Figure 4). On the other hand, because $v \in V_y$, the shortest path tree X shows that $path(v_{i+1}, v)$ is completely contained in V_y. Since G is undirected, $path(v, v_{i+1})$ is just the reversal of $path(v_{i+1}, v)$. But one contains v_i and one does not, a contradiction. Therefore $path(v, y)$ does not contain e_i, and the lemma is established. ∎

We have (almost) reduced the general undirected graph case to the case in which all vertices lie along $path(x, y)$. For an edge $e = (u, v) \notin path(x, y)$, we define $left(e) = block(u)$ and $right(e) = block(v)$, assuming $block(u) \leq block(v)$. This ensures that $e \in E_i$ if and only if $left(e) \leq i < right(e)$, and we can apply the algorithm of Section 4 directly. For each edge $(u, v) \in E_i$, the distance $d(x, u)$ is computed using the shortest path tree X, and the distance $d(v, y)$ is computed using Y.

We have established our main result:

Theorem 2 *Given an undirected graph $G = (V, E)$ with non-negative edge weights and a pair of vertices (x, y), we can compute $d(x, y;\ G \setminus e)$ for each edge $e \in path(x, y)$ in total time $O(n \log n + m)$, where $n = |V|$ and $m = |E|$.*

This theorem allows us to compute the Vickrey payments for all edges of a shortest path in a network, as given in equation (1), in the same asymptotic time as is needed to compute the shortest path itself.

6 Concluding remarks

With the emergence of the Internet as a global platform for communication, computation, and commerce, there is an increased need to design efficient protocols that motivate self-interested agents to cooperate. Example applications include resource allocation in computational grids [27], market-based protocols for scheduling or task allocation [25, 26], and congestion control in the Internet [6, 9, 11, 22]. One of the most celebrated results in the field of mechanism design is the Vickrey (or Vickrey-Clarke-Groves) protocol, which uses a payment scheme to motivate selfish agents to bid truthfully.

In this paper, we focused on the algorithmic aspect of computing the Vickrey payments in the context of shortest path routing in an internet, where multiple self-interested agents own portions of the network and links are bi-directional. Naïvely, computing payment functions for n agents requires n shortest path computations. Our main result shows that this computational overhead can be significantly reduced—the payments are computable in the same asymptotic time as a single shortest path tree. Our algorithm is quite simple, and uses only some elementary properties of shortest paths.

In the conference version of this paper [7], we erroneously claimed that our algorithm worked for *directed graphs* as well. We later discovered a bug in our proof. We have recently shown that the *directed graph* version of the problem is *provably* harder [8] than its undirected counterpart. There is an $\Omega(m\sqrt{n})$ time lower bound for any *path comparison* algorithm that solves the replacement paths problem for a directed graph with n nodes, m edges, whenever $m = O(n\sqrt{n})$.

Many interesting and challenging problems remain in the still nascent field of algorithmic mechanism design. For instance, many applications in distributed computing may require designing new mechanisms [15, 25, 27]. There are also many important problems in which computing Vickrey payments requires solving NP-complete problems [10, 19, 20]. In those cases, it would be interesting to use the techniques of approximation or randomization to design new polynomial-time mechanisms. The work of Bikhchandani et al. [3] and Schummer and Vohra [21] also suggests a promising direction for further exploration.

References

[1] M. O. Ball, B. L. Golden, and R. V. Vohra. Finding the most vital arcs in a network. *Oper. Res. Letters*, 8:73–76, 1989.

[2] A. Bar-Noy, S. Khuller, and B. Schieber. The complexity of finding most vital arcs and nodes. Technical Report CS-TR-3539, Institute for Advanced Studies, University of Maryland, College Park, MD, 1995.

[3] S. Bikhchandani, S. de Vries, R. Vohra, and J. Schummer. Linear programming and Vickrey auctions. In *Proceedings of the IMA Workshop on e-Auctions and Markets*, 2001.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1990.

[5] D. Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998.

[6] R. J. Gibbens and F. P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, 35:1969–1985, 1999.

[7] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 252–259, 2001.

[8] J. Hershberger, S. Suri, and A. Bhosle. On the difficulty of some shortest path problems, 2002.

[9] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker. Optimization problems in congestion control. In *Proc. 41st Annu. IEEE Sympos. Found. Comput. Sci.*, 2000.

[10] D. Lehmann, L. O'Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. In *Proc. ACM Conference on Electronic Commerce*, 2000.

[11] J. K. MacKie-Mason and H. R. Varian. Pricing congestible network resources. *IEEE Journal of Selected Areas in Communications*, 1995.

[12] K. Malik, A. K. Mittal, and S. K. Gupta. The k most vital arcs in the shortest path problem. *Oper. Res. Letters*, 8:223–227, 1989.

[13] A. Mas-Collel, W. Whinston, and J. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[14] E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. In *Proc. COCOON*, 2001.

[15] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. 31st Annu. ACM Sympos. Theory Comput.*, 1999.

[16] A. Odlyzko. A modest proposal for preventing Internet congestion. `http://www.research.att.com/~amo/doc/modest.proposal.ps`, 1997.

[17] J. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiations Among Computers*. MIT Press, 1994.

[18] T. Sandholm. Distributed rational decision making. In *Introduction to Multiagent Systems: A Modern Introduction to Distributed Artificial Intelligence*. MIT Press, 1999.

[19] T. Sandholm and S. Suri. Improved algorithm for optimal winner determination in combinatorial auctions and generalizations. In *AAAI 17th National Conference on Artificial Intelligence*, 2000.

[20] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *IJCAI 17th International Joint Conference on Artificial Intelligence*, 2001.

[21] J. Schummer and R. Vohra. Auctions for procuring options. Technical report, Northwestern University, 2001.

[22] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in computer networks: Reshaping the research agenda. *Telecommunications Policy*, pages 183–201, 1996.

[23] Y. Shoham and K. Tanaka. A dynamic theory of incentives in multi-agent systems. In *Intl. Joint Conf. on Artificial Intelligence*, 1997.

[24] R. E. Tarjan. Sensitivity analysis of minimum spanning trees and shortest path trees. *Information Processing Letters*, 14(1):30–33, 1982.

[25] W. E. Walsh and M. P. Wellman. A market protocol for decentralized task allocation. In *Proc. 3rd International Conference on Multi-Agent Systems*, 1998.

[26] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason. Auction protocols for decentralized scheduling. In *Proc. 18th International Conference on Distributed Computing Systems*, 1998.

[27] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. G-commerce: Market formulations controlling resource allocation on the computational grid. In *IPDPS*, 2001.