Computing Shortest Paths with Uncertainty

Tomás Feder [1], Rajeev Motwani [a,2], Liadan O'Callaghan [a,3], Chris Olston [a,4], Rina Panigrahy [b,5]

[a] *Department of Computer Science, Stanford University, CA 94305.*

[b] *Cisco Systems*

Abstract

We consider the problem of estimating the length of the shortest path from a vertex s to a vertex t in a DAG whose edge lengths are known only approximately but can be determined exactly at a cost. Initially, for each edge e, the length of e is known only to lie within an interval $[l_e, h_e]$; the estimation algorithm can pay w_e to find the exact length of e. We study the problem of finding the cheapest set of edges such that, if exactly these edges are queried, the length of the shortest s-t path will be known, within an additive $\kappa \geq 0$, an input parameter. An actual s-t path, whose true length exceeds that of the shortest s-t path by at most κ, will be obtained as well. The problem of finding a cheap set of edge queries is in neither NP nor co-NP unless NP = co-NP. We give positive and negative results for two special cases and for the general case, which we show is in Σ_2.

Key words: Shortest path.

[1] Email: `tomas@theory.stanford.edu`

[2] Email: `rajeev@cs.stanford.edu`. Research supported by NSF Grant IIS-0118173, an Okawa Foundation Research Grant, and Veritas.

[3] Contact Author. Email: `loc@cs.stanford.edu`. Research supported by NSF Graduate Fellowship and NSF Grant IIS-0118173.

[4] Email: `olston@db.stanford.edu`. Research supported by the National Science Foundation under grants IIS-9817799 and IIS-9811947 and by a National Science Foundation graduate research fellowship.

[5] Email: `rinap@cisco.com`.

1 Introduction

Consider a weighted DAG G with a single source s of in-degree zero and a single sink t of out-degree zero, whose exact edge lengths are not known with certainty. Assume that for every edge e of G the length of e is only known to lie in an interval $[l_e, h_e]$. The length of an s-t path in G can be computed with uncertainty and represented as an interval known to contain the exact length. An interval containing the length of the shortest s-t path can similarly be computed.

Suppose that for every edge e, the exact length of e can be found at query cost w_e. An optimization problem that arises naturally is: *Given a DAG G, a source s, a sink t, a set P of s-t paths, edge-length intervals $[l_{e_1}, h_{e_1}], \ldots, [l_{e_m}, h_{e_m}]$, edge query costs $w_{e_1} \ldots w_{e_m}$, and precision parameter κ, find a minimum-cost set of edges $E' = \{e_{i_1}, \ldots, e_{i_k}\}$ such that if exactly the edges in E' are queried, an interval of width at most κ can be identified, that is guaranteed to contain the length of the shortest $p \in P$.* P may be given explicitly, it may be specified implicitly, or it may not be specified at all, in which case P can be assumed to be the set of all s-t paths in G. It turns out that an actual s-t path, whose true length exceeds that of the shortest s-t path by at most an additive κ, will be obtained as well. Note that there are two natural versions of this problem. In the *online* version, the sequence of queries is chosen adaptively — each query is answered before the next one is chosen. In the *offline* version, the entire set of queries must be specified completely before the answers are provided, and it must be *guaranteed* that, regardless of the results of the queries, the length of the shortest s-t path can be pinned down as desired. In this paper we consider the offline formulation of the problem.

The problem of finding a cheap set of edge queries is neither in NP nor in co-NP unless NP = co-NP. We therefore study the hardness of the problem under various types of restrictions. In the special case in which $\kappa = 0$, which we term the "zero-error case," the problem is solvable in polynomial time if the set of paths is given explicitly or has a particular type of implicit description. If P is given explicitly, then the number of paths we consider is polynomial in the size of the input, and the zero-error problem has a polynomial-time solution; we also show that if P admits a recursive description as defined later, it can be analyzed in polynomial time. If the set of paths is unrestricted, however, for all $\delta > 0$ the zero-error problem is hard to approximate within $n^{1-\delta}$, even if the error *and* cost requirements are relaxed substantially. In order to obtain polynomial algorithms or reasonable approximation bounds, we must therefore consider suitable restrictions on the structure of instances of the problem. In many cases, under such restrictions we obtain matching upper and lower bounds.

We consider the case in which κ may be greater than zero, and examine different types of restrictions on the path structure of the graph. The first such restriction is the unique-upper-length requirement — that all s-t paths in the graph have the same upper bound on length. This restriction alone is not enough to make the problem tractable. With certain assumptions on κ, the length intervals, and the edge structure, however, the problem can be solved in polynomial time or with small approximation factors. We consider restrictions on κ, restrictions on the edge structure of the graph, and restrictions on the number and types of nontrivial edges (i.e., edges whose lengths are not known exactly) on the paths under consideration.

1.1 Motivation and Related Work

Our problem is motivated by the work of Olston and Widom [11] on query processing over replicated databases, where local cached copies of databases are used to support quick processing of queries at client sites. There is a master copy of the data where all the updates to the database are maintained. The frequency of updates makes it infeasible to maintain exact consistency between the cached copies and the master copy, and the data values in the cache are likely to become stale and drift from the master values. However, the cached copies store for each data value an interval that is guaranteed to contain the master value. Systems considerations sometimes make it desirable to perform all queries to the master copy en masse; in this case, the offline formulation is much more relevant.

In some cases, the data appears in the form of a graph with edges whose lengths are updated over time, and queries request a short path between two nodes. It is desirable to obtain a path whose length is within κ of that of the shortest path, where the value of κ is specified along with the query. This problem has applications in areas such as network monitoring and computerized route selection for automobiles in urban areas. In such applications, edge lengths may change rapidly, and an excess of edge-length queries will result in high network communication costs.

Another class of queries, aggregation with uncertainty, has been studied in the context of the interval caching framework. Olston and Widom [11] consider aggregation functions such as *sum*, *min*, and *max* for the offline formulation. The shortest-path problem we consider in this paper is a strict and common generalization of all three of these aggregation functions. Federet al.[4] consider both the online and the offline formulations of the selection problem, with *median* as a special case. Finally, Khanna and Tan [8] extend some of the results for the selection and sum problems, focusing on other precision parameter formulations. Papadimitriou et al. [12,3] study problems related to the online problem described earlier; in their model, a robot tries to discover the map of a two- or three-dimensional Euclidean region with cost competitive with that of the cheapest witness to the map structure; here, cost is the total distance the robot travels to discover the map. Karasan et al. [7] assume a digraph with source and sink and, for each edge, an interval containing the length; they study how to find a path with lowest worst-case absolute deviation in length from the shortest path. Montemanni and Gambardella [10] show, in the same setting, how to find a path with lowest worst-case competitive ratio. Here, the competitive ratio of an s-t path, given some assignment of lengths to edges, is its total length, divided by that of the shortest path. Aingworth, Chekuri, Indyk, and Motwani [1] show how to estimate the *all-pairs* shortest-path distances within an additive error of 2, in time $O(n^{2.5}\sqrt{\log n})$, for graphs in which edge-lengths are known exactly.

1.2 Summary of Main Results

Throughout the remainder of the paper, we will often use the term "path" to indicate "s-t path" when using the simpler terminology does not introduce ambiguity. Unless the endpoints of a path are given explicitly, the path should be assumed to originate at s and end at t.

We begin by considering two special cases. We will show that the zero-error case (i.e., where $\kappa = 0$) is in co-NP and that some of its versions are co-NP-complete. We also consider the unique-upper-length case (in which, for some constant H and all paths p in P, the sum of upper bounds of edges on p is H); we will show that it is in NP and that some its versions can be NP-complete. The general case is therefore neither in NP nor in co-NP if NP$\neq$co-NP: however, it is in the class Σ_2. An interesting open question is whether the general problem is actually complete for Σ_2. For both special cases and in the general version of the problem, we exhibit situations where the problem can be solved exactly, and also obtain both positive and negative results on the approximability of the problem.

Section 2 addresses the zero-error version of the problem. If the collection P of paths is given explicitly, or if it is described implicitly through a recursive decomposition of the graph, then there is a polynomial-time algorithm that finds the optimal set of edges to query in order to guarantee that the length of the shortest path (and the path itself) will be obtained regardless of the answers to the queries. The main example of such an implicit presentation of paths is the case where P consists of all the paths in a series-parallel graph (which can be exponentially many). For general graphs, if P contains all s-t paths, all nontrivial edges (i.e., edges whose lengths are not known exactly) have unit query costs, and each edge has interval either $[0, 0]$ or $[0, 1]$, then the zero-error problem is co-NP-complete and hard to approximate within $n^{1-\delta}$ for any constant $\delta > 0$. In fact, for instances having a zero-error solution of cost w, it is hard to find a solution with error at most $\kappa = n^{\delta_1}$ of cost at most wn^{δ_2} for all constants $\delta_1, \delta_2 > 0$ satisfying $\delta_1 + \delta_2 < 1$.

In Section 3 the unique-upper-length case is explored. In the unique-upper-length case, where H denotes the unique upper length, if P consists of all s-t paths, and $K \leq H$ is the best κ achievable when no interval is queried, then the unique-upper-length problem can be solved exactly in polynomial time for $\kappa = 0, 1, K-1$, or K. The unique-upper-length problem is NP-complete, and hard to approximate within $1 + \delta$ for some $\delta > 0$, if $2 \leq \kappa \leq K - 2$. It has both a $(K - \kappa)$-approximation algorithm and a $(2\sqrt{\kappa} + 1)$-approximation algorithm. A 2-approximation can be obtained if we are willing to approximate the error κ besides approximating the cost. An interesting question is whether the problem remains NP-complete when all edge-length intervals are either zero or one. This case has both a $1 + \log(K - \kappa)$-approximation algorithm and a $(3 + \log \kappa)$-approximation algorithm. The case is equivalent to a generalization of the minimum cut problem, where we seek a set of edges of minimum cost that contains at least r edges on p for each path p from s to t. The case of series-parallel graphs becomes NP-complete for the unique-upper-length problem but has a polynomial time approximation scheme. When the set of paths P is given explicitly, the unique-upper-length problem becomes as hard to approximate as $(\kappa + 1)$-hypergraph vertex cover, and has an $(\kappa + 1)$-approximation algorithm.

The general problem, in which κ may be arbitrary and in which there may not be a unique upper length, is covered in Section 4. The section starts with the exact conditions that a set of queries must satisfy in order to guarantee that the length an algorithm returns is within κ of the true shortest path length. Next, it is shown that if each p in P contains at most r nontrivial edges, then the problem is as easy and as hard to approximate as r-hypergraph vertex cover, for $r = O(\log n)$. If the nontrivial edges can each be assigned one of r colors so that, in each p, the nontrivial edges all have distinct colors, then the problem is exactly as hard to approximate as r-partite hypergraph vertex cover. This implies an $r/2$-approximation algorithm for the case where P consists of all paths and each path has at most r nontrivial edges. The case of series-parallel graphs seems harder to approximate for the general problem. Within this class, we obtain a polynomial time approximation scheme when the graph consists of the combination of a tree attached to the source s and a tree attached to the sink t. More generally, it is possible to combine in series graphs with a good approximation algorithm and still obtain a good approximation algorithm for the combined graph.

We first introduce some notation that will facilitate many of the discussions that follow.

Definition 1 *An edge e is nontrivial just if $l_e \neq h_e$.*

Suppose p and q are s-t paths in P, and that the upper bounds (values h_e) of edges in $q \setminus p$ and the lower bounds (l_e) of edges in $p \setminus q$ are such that p is guaranteed not to be shorter than q, but not guaranteed to have the same length as q. Then we will say p *dominates* q.

Definition 2 *Let p, $q \in P$, and let $L_{pq} = \sum_{e \in p \setminus q} l_e$ and $H_{pq} = \sum_{e \in q \setminus p} h_e$. Then we will say p* dominates *q if $L = H$ and p and q do not have the same nontrivial edges, or if $L > H$.*

For simplicity of notation, when we are discussing whether some path p dominates a path q, we will abbreviate L_{pq} by L, and H_{pq} by H.

Definition 3 *A path $p \in P$ will be called* non-dominant *if there is no $q \in P$ such that p dominates q.*

2 The Zero-Error Case

We consider first the case with no error, i.e., $\kappa = 0$.

Proposition 1 *A choice of a set E' of edges to query guarantees zero error if and only if E' includes all the nontrivial edges in each non-dominant s-t path $p \in P$.*

Proof: Suppose a nontrivial edge e is in a non-dominant path p. We will show that, unless e is queried, even if all other edges are queried, it will be impossible to tell the length of the shortest path in P.

There is a $0 < \delta < h_e - l_e$ such that for every path q, if q and p do not have the same nontrivial edges, $\sum_{e \in p \setminus q} l_e \leq \sum_{e \in q \setminus p} h_e - \delta$. (Informally: since p dominates no q, there is some minimum margin by which it fails to dominate.) If we query all $f \neq e$, it is possible that we will obtain the answer l_f for all $f \in p$, and the answer h_f for all $f \notin p$. In this case, the length of every $q \neq p$ will exceed the length of p by at least δ, regardless of the true length of e, so p will be the shortest path. However, its length will not be known exactly, as $h_e - l_e > \delta$ uncertainty in its length will remain.

If p dominates some q, then we can ignore p because for all possible answers to the queries, the length of p is at least the length of q, and the domination relation is acyclic. $\square$

Proposition 2 *The zero-error problem is in co-NP. That is, given a set E' of nontrivial edges and a set P' of at most $|E'|$ s-t paths containing, for each $e \in$*

E' some $p \in P'$ that includes the edge e, it is possible to decide in polynomial time whether each $p \in P'$ is non-dominant (which implies that all edges in E' must be queried if the shortest-path length is to be known exactly).

Proof: Since the number of paths in P' is polynomial in $|E'|$, we need only show that for each $p \in P'$ it is possible to decide in polynomial time whether p is non-dominant. We will show a condition that can be checked in polynomial time, and we will show that this condition holds if and only if p is non-dominant.

First we will make some useful observations and definitions. Note that, because G is acyclic, for every pair a, b of vertices in the DAG G, there cannot be a path from a to b *and* one from b to a. If a and b are on p, and there is a path from a to b (and therefore none from b to a), say that a *precedes* b on p. For every edge $e \in E$, calculate $t_{e,p}$ as l_e if $e \in p$, and h_e otherwise. For every pair a, b of vertices in p such that a precedes b, let $p[a, b]$ denote the section of p from a to b, and let $G'[a, b]$ denote the graph containing all the edges in G except those in $p[a, b]$, with the length of each edge e set to $t_{e,p}$. Calculate $L[a, b]$ as $\sum_{e \in p[a,b]} l_e$, and calculate $H[a, b]$ as the length of the shortest a-b path in the graph $G'[a, b]$. Note that because a precedes b on p, and because $G'[a, b]$ excludes the edges in $p[a, b]$, this shortest path will not use any edges on p.

Consider the following conditions, which, as we will show later, can be checked in polynomial time:

(1) $L[a, b] < H[a, b]$
(2) (a) $L[a, b] = H[a, b]$,
 (b) $p[a, b]$ has no nontrivial edges,
 (c) and for every nontrivial edge $f \in G'[a, b]$, if there is an a-b path in $G'[a, b]$ containing f, the shortest such path has a total length greater than $H[a, b]$.

Note that for each choice of vertices a and b on p there is a different instantiation of Conditions 1 and 2. We will show that p is non-dominant if and only if for all a and b Condition 1 or 2 holds. We will then show that Conditions 1 and 2 can be checked in polynomial time. Since there are at most a polynomial number of choices for a and b, it will follow that all conditions relevant to p can be checked in polynomial time, and thus that we can decide in polynomial time whether p is non-dominant.

First suppose that there exist a and b for which Conditions 1 and 2 both fail. We will show that p must dominate some s-t path q. Let q'' be the a-b path found during the checking of Condition 1, and let q' denote the path formed by the replacement of $p[a, b]$ with q''. Then a and b are on both paths p and q', and the only difference between p and q' is on the section from a to b, where the paths are fully disjoint. Since Condition 1 fails, it follows that

$L[a,b] \geq H[a,b]$. If this inequality is strict, then $L > H$, where $L = \sum_{e \in p \setminus q'} t_e$, and $H = \sum_{e \in q' \setminus p} t_e$ (note that because of the way the values t_e were calculated, the first sum is a sum of lower bounds, and the second is a sum of upper bounds). In this case, p dominates q'.

Otherwise, $L[a,b]$ must be equal to $H[a,b]$. As Condition 2 fails, either 2b or 2c must fail. If 2b fails, p dominates q' because p and q do not have the same set of nontrivial edges. Otherwise, there must exist a nontrivial edge $f \in G'[a,b]$ such that there is an a-b path in $G'[a,b]$ containing f, and such that the shortest such path has a total length at most $H[a,b]$. Take such an f, and let r' be the shortest a-b path containing f; by the optimality of q'' the length of r must equal $H[a,b]$, and r' contains a nontrivial edge that cannot lie on p (namely f, which cannot come either before a or after b on p since G is a DAG). Therefore, p dominates the path r that is identical to p except between a and b where it is identical to r'.

Next, suppose that p dominates some s-t path q. We will show that there exist vertices a and b on p for which Conditions 1 and 2 fail. Let us say that p and q *diverge* from a vertex x to a vertex y if x and y lie on both p and q, x precedes y, and $p[x,y]$ is fully disjoint from $q[x,y]$. Both p and q are paths from s to t, and so there is at least one choice of x and y such that p and q diverge from x to y. Let $L = \sum_{e \in p \setminus q} t_e$, and $H = \sum_{e \in q \setminus p} t_e$. Note that since p dominates q, $L \geq H$, and if $L = H$, then p and q have different sets of nontrivial edges.

Then p and q may diverge from one vertex x to another vertex y so that $\sum_{e \in p[x,y]} t_e > \sum_{e \in q[x,y]} t_e$; if so, we will say that p *surpasses* q from x to y. If there exist x and y such that p surpasses q from x to y, then consider the shortest x-y path r in $G[x,y]$. $\sum_{e \in r[x,y]} t_e \leq \sum_{e \in q[x,y]} t_e < \sum_{e \in p[x,y]} t_e$ (note r is the same as $r[x,y]$). Then Conditions 1 and 2a fail when $a = x$ and $b = y$ since in this case $L[a,b] > H[a,b]$.

If there are no vertices x and y for which p surpasses q from x to y, then for every pair of vertices c and d such that p and q diverge from c to d, we have $\sum_{e \in p[c,d]} t_e \leq \sum_{e \in q[c,d]} t_e$. Since we have assumed that p dominates q, we know that $\sum_{e \in p \setminus q} t_e \geq \sum_{e \in q \setminus p} t_e$, and so there cannot be vertices c and d such that q surpasses p from c to d. Therefore, for all pairs of vertices c and d such that p and q diverge from c to d, we have $\sum_{e \in p[c,d]} t_e = \sum_{e \in q[c,d]} t_e$. It follows that $\sum_{e \in p \setminus q} t_e = \sum_{e \in q \setminus p} t_e$. Since, by assumption, p dominates q, p and q must have different nontrivial edges.

If p and q diverge from some x to some y so that $p[x,y]$ has at least one nontrivial edge, then the path that replaces $p[x,y]$ with $q[x,y]$ violates Conditions 1 and 2b. Otherwise, p and q diverge from some x to some y so that $q[x,y]$ has at least one nontrivial edge. Then either $q[x,y]$ is itself the shortest x-y path, in which case Conditions 1 and 2c are violated, or else the shortest x-y path in $G[x,y]$ (say $q'[x,y]$) is strictly shorter than $q[x,y]$. In the latter case, $p[x,y]$ is strictly longer than $q'[x,y]$, and so $\sum_{e\in p[x,y]} t_e > \sum_{e\in q'[x,y]} t_e = \sum_{e\in q'} t_e$, violating Conditions 1 and 2a.

Finally, we note that Conditions 1 and 2 can be checked in polynomial time, since finding the shortest a-b path that passes through a particular edge f which goes from vertex u to vertex v, is equivalent to finding the shortest a-u path and the shortest v-b path. □

Theorem 1 *If the collection P is given explicitly, then the zero-error problem can be solved in polynomial time.*

For the proof, observe that by Proposition 1, it suffices to test for each path p in P whether p dominates some q in P, and query all nontrivial edges in p if it does not.

We consider next the following *implicit description* of P for G. Either (1) G consists of a single edge $e = (s,t)$, and P has the single path given by e; or, (2) We are given an implicit description of P' for G' with source s and sink t, where G' contains an edge $e = (s',t')$, and we are also given an explicit description of P'' for G'' with source s' and sink t'. The graph G is obtained by taking G' and replacing e with G''; the paths P are obtained by taking the paths P' and replacing the occurrences of e in these paths with each of the paths in P''.

The following propositions relate to *series-parallel* graphs:

Definition 4 *A graph is* series-parallel *if it is possible to reduce it to a single edge by repeatedly replacing G'' consisting of two edges in series or two edges in parallel with a single edge.*

Proposition 3 *In the implicit description of P for G, suppose that each explicit description of P'' for G'' used contains all the paths in G''. Then P contains all the paths in G. In particular, if G is a series-parallel graph, then each such G'' consists of just two edges (either in series or in parallel), giving a polynomial description for all paths in G.*

For the proof, note that all paths in G correspond to paths in G' which may go through the special edge e, and if they do, to a choice of path in G''. The result follows by induction.

Theorem 2 *If the collection of paths P for G is implicitly presented as described earlier, then the zero-error problem can be solved in polynomial time.*

Proof: Suppose p in P dominates some q in P. We may choose q so that p and q have the same edges from s to some s', are disjoint from s' to some t', and have the same edges from t' to t.

Simplify the graph G by repeatedly replacing the components G'' with single edges, until we obtain a component G'' containing both s' and t', for all choices of s' and t'. In this component G'', consider all pairs of paths p'' and q'' from P''; some such pair corresponds to p and q.

We can then determine the values L and H for p'' and q''. To determine H, choose h_f for all edges f in q'' not in p''. If such an edge f corresponds to a subcomponent, find the shortest path in that subcomponent corresponding to taking h_g for each edge g in the subcomponent. To determine L, choose l_f for all edges f in p'' not in q''. If such an edge f corresponds to a subcomponent, find the shortest path in that subcomponent corresponding to taking l_g for each edge g in the subcomponent; here, however, we only consider paths in the subcomponent that do not dominate other paths, which we can assume have been precomputed by the same algorithm. We also only consider paths going through the edge e being tested for membership in a path p that does not dominate any q. $\square$

Theorem 3 *If P consists of all the paths in the given G, and each $[l_e, h_e]$ is either $[0,0]$ or $[0,1]$, the zero-error problem is co-NP-complete; in the unit-cost case it is hard to approximate within $n^{1-\delta}$, for any constant $\delta > 0$. In fact, it is hard to distinguish the case where there is a zero-error solution of cost w from the case where there is no solution of error at most $\kappa = n^{\delta_1}$ and cost at most wn^{δ_2} for all constants $\delta_1, \delta_2 > 0$ satisfying $\delta_1 + \delta_2 < 1$.*

Proof: We show that testing whether an edge e belongs to some path p that does not dominate any path q is NP-complete. The reduction is from 3-colorability.

We will construct a DAG as follows. We have a DAG S with source s and sink s', where all edges have interval $[0,1]$, and a DAG T with source t' and sink t, where all edges have interval $[0,1]$. The edge e goes from s' to t'. There is also a set E of edges with interval $[0,0]$ from vertices in S to vertices in T. The required path p consists then of a path p_S in S from s to s' and a path p_T in T from t' to t. Such a path does not dominate any other path if and only if no edge in E joins a vertex in p_S and a vertex in p_T.

Let G be the instance of 3-colorability, with vertices $v_1, \ldots, v_n$ and edges $e_1, \ldots, e_m$. The DAG S, in addition to s and s', has vertices $(v_i, 1)$, $(v_i, 2)$ and $(v_i, 3)$ for each v_i in G. The edges in S go from each (v_i, j) to each (v_{i+1}, j'), plus edges from s to each (v_1, j) and from (v_n, j) to s'. The path p_S thus chooses a (v_i, j) for each v_i, i.e., assigns color j to v_i.

The DAG T, in addition to t' and t, has vertices $(e_l, 1, 1)$, $(e_l, 2, 1)$, $(e_l, 3, 1)$, $(e_l, 1, 2)$, $(e_l, 2, 2)$, $(e_l, 3, 2)$ for each e_l in G. The DAG T has a path from t' to t

going through all $(e_l, j, 1)$; the $(e_l, j, 2)$ are connected similarly to the $(e_l, j, 1)$. Thus the path p_T chooses a k in (e_l, j, k) for each e_l, j.

If G has an edge $e_l = (v_i, v_{i'})$, Then E has the edges $((v_i, j), (e_l, j, 1))$ and $((v_{i'}, j), (e_l, j, 2))$. Thus p_T will be able to choose a k in (e_l, j, k) for e_l, j if and only if v_i and $v_{i'}$ are not both assigned color j. Thus the choice of paths p_S, p_T corresponds to a 3-coloring of G.

This gives NP-completeness. The hardness of approximation follows from the fact that the special edge e can be given an arbitrarily large cost, much larger than the sum of all the other costs. In the unit cost case, we can use a large number of parallel edges $e_1, \ldots, e_t$ of unit cost instead of e to achieve the same effect as a large cost, and insert extra edges of length $[0, 0]$ next to each e_i if parallel edges are not allowed. Each e_i is in a non-dominant path if and only if e was in a non-dominant path. Thus, with unit costs, we have some number m_0 edges other than e, and $t = m_0^c \approx n$ edges e_i corresponding to e, giving an approximation hardness of $t/m_0 = m_0^{c-1} = n^{1-1/c} = n^{1-\delta}$ for any choice of sufficiently large constant c and corresponding small constant $\delta = 1/c > 0$.

The stronger hardness of approximation follows by replacing each edge with length $[0, 1]$ with a path of r edges of length $[0, 1]$. Then we have $m_0 r$ edges other than e, and $m_0^c r \approx n$ edges corresponding to e. Setting $r = n^{\delta_1}$, $m_0^{c-1} = n^{\delta_2}$, and $m_0 = n^{1-\delta_1-\delta_2}$ gives the result as follows. If it was not necessary to query e, then querying the $m_0 r$ not corresponding to e costs $w = m_0 r = n^{1-\delta_2}$. If it *was* necessary to query e, then error at most $r/2 = n^{\delta_1}/2$ can only be guaranteed if we query at least $r/2$ of the edges in each of the t paths corresponding to e_i, with cost $tr/2 = m_0^c r/2 = w m_0^{c-1}/2 = w n^{\delta_2}/2$. Deciding whether it was necessary to query e is NP-hard as above. □

3 The Unique-Upper-Length Case

We now allow error $\kappa \geq 0$. For each edge e with interval $[l_e, h_e]$ we assume both l_e and h_e are integers, and so we can assume without loss of generality that κ is also an integer. The unique-upper-length case is the case in which there is an integer H such that for each path p from s to t, $\sum_{e \in p} h_e = H$.

Proposition 4 *In the unique-upper-length case, a choice of queried edges guarantees error κ if and only if it guarantees error κ for each path from s to t in P.*

Proof: Suppose there is a path p such that the sum of the lengths of the intervals $[l_e, h_e]$ for edges e in p not queried is greater than κ. Suppose the queries return h_f for all edges f queried. Then the remaining error is at least

$\kappa + 1$, because the shortest path could have length H if we choose h_e for all edges e not queried, or at most $H - \kappa - 1$ on p if we choose l_e for all edges e in p not queried.

Suppose instead that, for all paths q, the sum of the lengths of the intervals $[l_e, h_e]$ for unqueried edges $e \in q$ is at most κ. Let p be the path minimizing $\sum_{e \in p} l_e$, and let $L' = \sum_{e \in p} l_e$. By assumption, p cannot attain a length greater than $L' + \kappa$, and since no path can be shorter than L', the interval $[L', L' + \kappa]$ must contain the length of the true shortest path. $\square$

Proposition 5 *The unique-upper-length problem is in NP. That is, in the unique-upper-length case, given a set of edges, it is possible to decide in polynomial time whether querying these edges is sufficient to guarantee that an interval of width at most κ can be identified which is known to contain the length of the shortest s-t path.*

Proof: By Proposition 4, we need only show a polynomial-time procedure for deciding whether, once the queries in E' are posed, no path has more than κ uncertainty.

For all edges $e \in E$, set $t_e = h_e$ if $e \in E'$ and $t_e = l_e$ otherwise. Let L' be the length of the shortest s-t path when every edge e has length t_e. Then for every s-t path p, once the queries in E' are posed, the remaining uncertainty will be

$$\begin{aligned}\sum_{e \in p \setminus E'} (h_e - l_e) &= \sum_{e \in p \setminus E'} h_e - \sum_{e \in p \setminus E'} l_e \\ &= \sum_{e \in p} h_e - \sum_{e \in p \cap E'} h_e - \sum_{e \in p \setminus E'} l_e \\ &= H - \sum_{e \in p} t_e\end{aligned}$$

Therefore, once the queries in E' are posed, the most uncertainty that any path can have is $H - L'$, since L' is the minimum possible value, over all s-t paths p, of $\sum_{e \in p} t_e$.

Finding L', and deciding whether $H - L' \leq \kappa$, can be done in polynomial time. The result follows. $\square$

We assume next that P consists of all paths from s to t. Let $K \leq H$ denote the interval between the minimum and maximum possible values the shortest path length could take on if no interval were queried.

Theorem 4 *The unique-upper-length case can be solved in polynomial time for $\kappa = 0, 1, K - 1, K$.*

Proof: Error $\kappa = K$ requires no queries, while $\kappa = 0$ requires querying all nontrivial intervals (by Proposition 3).

Suppose $\kappa = K - 1$. We must query at least one edge in each path p such that the sum of the lengths of the intervals $[l_e, h_e]$ over edges e in p equals K. Choose the l_e values for all edges e, and compute for each vertex v the length l_v of the shortest path from s to v. Then $K = H - l_t$. The sum of the lengths of the intervals $[l_e, h_e]$ over edges $e = (v, v')$ in p equals K if and only if $l_v + l_e = l_{v'}$ for all such e. Therefore, if we remove all edges e with $l_v + l_e > l_{v'}$, then we must query at least one edge for each path p from s to t in the resulting graph. This is the same as computing a minimum cost cut between s and t, which can be obtained by a maximum flow computation.

Suppose next $\kappa = 1$. Then all edges e with interval $[l_e, h_e]$ such that $h_e - l_e > 1$ must be queried, and all but at most one of the edges e with interval $[l_e, h_e]$ with $h_e - l_e = 1$ on a path p must be queried. Define an associated graph G' whose vertices are the edges e with $h_e - l_e = 1$, with an edge from e to e' in G' if e precedes e' in some path p. The graph G' is transitively closed, and paths p correspond to cliques in G'; we must therefore query all but at most one e in each such clique, i.e., the queried edges must form a vertex cover in G'.

Suppose e has incoming edges from A and outgoing edges to B in G'. Since G' is transitively closed, it also has all edges from A to B, and thus a vertex cover must choose all of A or all of B. Replace e with two vertices e_1, e_2, so that e_1 has the incoming edges from A and e_2 has the outgoing edges to B. Then a vertex cover will only choose at most one of e_1, e_2, since it chooses all of A or all of B. Furthermore, a vertex cover choosing e corresponds to a vertex cover choosing one of e_1, e_2, and vice versa. If we apply this transformation for each e, then each vertex e_1 has only incoming edges and each vertex e_2 has only outgoing edges. Therefore the graph is a bipartite graph, with vertices e_1 in one side and vertices e_2 in the other side. A minimum cost vertex cover in a bipartite graph can be obtained by a maximum flow computation. $\square$

Theorem 5 *The unique-upper-length case is NP-complete, and hard to approximate within $1 + \delta$ for some $\delta > 0$, if $2 \leq \kappa \leq K - 2 \leq H - 2$. This includes the case $K = H = 4$ and $\kappa = 2$, even if: (1) all $[l_e, h_e]$ intervals are $[0, 0], [0, 1]$, or $[0, 2]$, with at most four nontrivial intervals per path (which has a 2-approximation algorithm); or, (2) all $[l_e, h_e]$ intervals are $[0, 1], [0, 2]$ or $[1, 1]$, with at most three nontrivial intervals per path (which has a 1.5-approximation algorithm).*

Proof: Suppose $K = H = 4$ and $\kappa = 2$. We do a reduction from vertex cover for a graph G. In fact, we consider vertex cover for a graph G' obtained from G by replacing each edge (v, v') in G with a path (v, x, y, v') of length 3 in G'. The optimal vertex covers are related by $\text{opt}(G') = \text{opt}(G) + |E(G)|$.

The DAG has a vertex v for each v in G, and a corresponding edge (s, v) with interval $[0, 1]$. For each edge (v, v') in G, the DAG has two extra vertices a, b, an edge (v, a) with interval $[0, 2]$ corresponding to x, an edge (a, t) with interval $[0, 1]$ corresponding to y, an edge (v', b) with interval $[0, 1]$ of very large cost that will not be queried, and an edge (b, a) with interval $[0, 1]$ of zero cost that will be queried for Case (1) or with interval $[1, 1]$ for Case (2).

For the paths (s, v, a, t), we must either query (v, a) corresponding to x or query both (s, v) and (a, t) corresponding to v and y respectively. This corresponds to covering the two edges (v, x) and (x, y). For the paths (s, v', b, a, t) we must query either (s, v') or (a, t) corresponding to v' and y respectively. This corresponds to covering the edge (y, v'), completing the reduction.

If only unit costs are allowed, then for Case (2) use a large number of parallel paths (v', b, a) from v' to a to simulate a large cost. For Case (1), use parallel edges to simulate cost, and add edges with interval $[0, 0]$ if parallel edges are not allowed.

This proves NP-completeness. For the hardness of approximation, take G of maximum degree d for some constant $d \geq 3$, for which vertex cover is known to be hard to approximate within $1 + \delta$ for some $\delta > 0$ [2]. We know that the optima r, r' for G, G' are related by $r' = r + m$, where $m = |E(G)|$. Furthermore $r \geq \frac{m}{d}$. Therefore $r = \frac{r}{d+1} + \frac{rd}{d+1} \geq \frac{r}{d+1} + \frac{m}{d+1} = \frac{r'}{d+1}$, so an excess of δr for the vertex cover obtained in G corresponds to an excess of $\frac{\delta}{d+1} r'$ for the vertex cover obtained in G', giving hardness within $1 + \frac{\delta}{d+1}$ for G' and hence for the corresponding DAG with source s and sink t.

To obtain the result for any κ, K, H with $2 \leq \kappa \leq K - 2 \leq H - 2$, insert right after s a path with $H - K$ edges with interval $[1, 1]$, and $K - 4$ edges with interval $[0, 1]$ of which $\kappa - 2$ have very large cost and will not be queried, and $K - \kappa - 2$ have zero cost and will be queried. Again large different costs can be simulated with different numbers of parallel paths of two edges in the unit cost case.

The approximation upper bounds follow from Corollary 1 in Section 4. $\square$

Next we show pseudopolynomial approximation algorithms for the unique-upper-length case, under various assumptions.

Theorem 6 *The unique-upper-length case has a $(K - \kappa)$-approximation algorithm; and for all integers d has an $(\frac{\kappa}{d} + d)$-approximation algorithm, in particular a $(2\sqrt{\kappa} + 1)$-approximation algorithm. We can get a 2-approximation with respect to the optimum cost for a given κ if we allow replacing the error κ by 2κ as well. In the case where all intervals $[l_e, h_e]$ satisfy $h_e - l_e \leq 1$, we get an $H_{K-\kappa} \leq 1 + \log(K - \kappa)$ approximation algorithm, where $H_v = \sum_{1 \leq i \leq v} \frac{1}{v} \leq$*

$1 + \log v$; *and for all integer* d *an* $(\frac{\kappa}{d} + 1 + H_{d-1})$*-approximation algorithm, in particular a* $(3 + \log \kappa)$*-approximation algorithm.*

Proof: We give the $(K - \kappa)$-approximation algorithm. Let $\mu = K - \kappa$, and let w be the cost of the optimal solution r_0. Suppose we have found a solution r that reduces the error by $\lambda \leq \mu - 1$. We shall find extra edges of total cost at most w which when combined with the solution r reduce the error by at least $\lambda + 1$. Applying this at most μ time yields the result.

For each edge e in r, replace its interval $[l_e, h_e]$ with $[h_e, h_e]$. The extra edges must reduce the error in this new graph by at least 1, and an optimal solution for this problem can be obtained from Theorem 4. This optimal solution has cost at most w because the edges in r_0 and not in r provide such a solution and have cost at most w.

We give the $(\frac{\kappa}{d} + d)$-approximation algorithm. We consider the linear programming relaxation of the problem. We introduce a variable $0 \leq x_e \leq h_e - l_e$ for each edge e. The idea is that $x_e = 0$ if e is queried and $x_e = h_e - l_e$ if e is not queried. We want the longest path using the x_e values to be at most κ. This can be expressed by introducing a variable y_v for each vertex, and adding the conditions $y_s = 0$, $y_t \leq \kappa$, and $y_{v'} \geq y_v + x_e$ for each edge $e = (v, v')$. The objective function to be minimized is $\sum_e w_e(1 - \frac{x_e}{h_e - l_e})$. Recall that w_e is the cost of querying the edge e; this LP relaxation effectively allows the cost to be prorated. Let w be the value of the optimum. If we query all edges for which $1 - \frac{x_e}{h_e - l_e} \geq \frac{d}{\kappa + d}$, we incur cost at most $\frac{\kappa + d}{d} w$. For each path p from s to t we have $\sum_{e \in p} x_e \leq \kappa$. For each edge e in p not queried, we have $1 - \frac{x_e}{h_e - l_e} < \frac{d}{\kappa + d}$, or $h_e - l_e < \frac{\kappa + d}{\kappa} x_e$. Therefore the sum of the lengths $h_e - l_e$ of intervals over edges e in p not queried is at most $\kappa + d - 1$. So we must reduce the error further by $d - 1$. After replacing the intervals for edges e previously queried with $[h_e, h_e]$, this will cost at most $(d - 1)w$ by the previous result. The total cost is thus at most $(\frac{\kappa + d}{d} + d - 1)w = (\frac{\kappa}{d} + d)w$, as required.

The remark on doubling both the cost of the optimum and the error allowed follows from setting $d = \kappa$ in the linear programming rounding.

Suppose now all intervals $[l_e, h_e]$ satisfy $h_e - l_e \leq 1$. We give the $H_{K-\kappa}$-approximation algorithm. Let $\mu = K - \kappa$, and let w be the cost of the optimal solution r_0. Suppose we have found a solution r that reduces the error by $\lambda \leq \mu - 1$. We shall find extra edges of total cost at most $\frac{w}{\nu}$ for $\nu = \mu - \lambda$ which when combined with the solution r reduce the error by at least $\lambda + 1$. Applying this at most μ times yields the result.

For each edge e in r, replace its interval $[l_e, h_e]$ with $[h_e, h_e]$. The extra edges must reduce the error in this new graph by at least 1, and an optimal solution for this problem can be obtained from Theorem 4. We must show that this optimal solution costs at most $\frac{w}{\nu}$. The edges in r_0 and not in r reduce the error in this graph by at least ν: call these edges r_1. Assign to each edge e the value l_e, and compute for each vertex v the length l_v of a shortest path from s to v. A solution must query at least one nontrivial edge in each path p such that every nontrivial edge $e = (v, v')$ in p has $l_{v'} - l_v = l_e$, that is, in every path p after we remove all edges e that have $l_{v'} - l_v < l_e$. Now every path p has at least ν edges in r_1, so we can divide the edges in r_1 into ν sets such that each set is a cut: the ith set contains all the edges that occur at the earliest as the ith edge from r_1 in some path p. Thus, some such set constitutes a cut with cost at most $\frac{w}{\nu}$. The optimal solution from Theorem 4 then has cost at most $\frac{w}{\nu}$ as well.

The $(\frac{\kappa}{d} + 1 + H_{d-1})$-approximation algorithm follows as before. We incur cost $\frac{\kappa+d}{d}w$ to reduce the error to at most $\kappa + d - 1$ by linear programming, and we then incur cost $H_{d-1}w$ to reduce the error to κ by the algorithm just described. □

Theorem 7 *The unique-upper-length case on series-parallel graphs has a $(1+\delta)$-approximation algorithm for all $\delta > 0$, and an exact polynomial time algorithm if κ is bounded by a polynomial.*

Proof: We compute an optimal solution on G for all $\kappa' \leq \kappa$. If G consists of two graphs G_1, G_2 in parallel, then we must have a solution for κ' on each of G_1, G_2. If G consists of two graphs G_1, G_2 in series, then we must have a solution for κ_1 on G_1 and a solution for κ_2 on G_2 with $\kappa_1 + \kappa_2 = \kappa'$. We can try all such decompositions of κ'. The result follows by induction on the structure of G, when κ is bounded by a polynomial.

For larger κ, we introduce an approximation factor of $1 + \frac{\delta}{n}$, and do a binary search for possible values of κ', stopping each branch of the binary search when the solutions for two consecutive values of κ' differ by a factor of at most $1 + \frac{\delta}{n}$. Thus, a polynomial number of solutions will be maintained. When the solutions for G_1, G_2 are combined to obtain solutions for G, the number of solutions obtained for G may be larger, but can be reduced again by incurring another factor of $1 + \frac{\delta}{n}$. Since graphs are combined at most n times, the total factor incurred is at most $(1 + \frac{\delta}{n})^n \approx 1 + \delta$. □

Note that the special case where the series-parallel graph is a single path is the same as knapsack [11] and thus NP-complete.

The case where the collection of paths P is given explicitly and does not contain all paths is harder.

Theorem 8 *For a given collection P of s-t paths for G, with intervals $[l_e, h_e]$ given by $[0,0]$, $[0,1]$ or $[1,1]$, the unique-upper-length case is as hard to approximate as $(\kappa+1)$-hypergraph vertex cover even if $\kappa = K - 1$. With arbitrary* *intervals $[l_e, h_e]$, the problem has an $(\kappa+1)$-approximation algorithm.*

Proof: Let R be an instance of $(\kappa+1)$-hypergraph vertex cover consisting of n vertices, and hyperedges each having $\kappa+1$ vertices. Construct a DAG consisting of a path of length n, where each edge is replaced by two parallel edges with intervals $[0,1]$ and $[1,1]$. (Use an additional edge with interval $[0,0]$ if parallel edges are not allowed.) Each edge on the path of length n corresponds to a vertex of the hypergraph. We choose a path for each hyperedge, selecting the edges $[0,1]$ corresponding to the vertices in the hyperedge, otherwise selecting the edges $[1,1]$. A solution must query at least one edge in each such path, and thus corresponds to a vertex cover in the hypergraph.

The $(\kappa+1)$-approximation algorithm is obtained again by the linear programming relaxation. We introduce a variable $0 \le x_e \le h_e - l_e$ for each edge e. The idea is that $x_e = 0$ if e is queried and $x_e = h_e - l_e$ if e is not queried. Write $\sum_{e\in p} x_e \le \kappa$ for each s-t path p in P. The objective function to be minimized is $\sum_e w_e(1 - \frac{x_e}{h_e - l_e})$. Let w be the value of the optimum. If we query all edges for which $1 - \frac{x_e}{h_e-l_e} \ge \frac{1}{\kappa+1}$, we incur cost at most $(\kappa+1)w$. For each edge e in p not queried we have $1 - \frac{x_e}{h_e-l_e} < \frac{1}{\kappa+1}$, or $h_e - l_e < \frac{\kappa+1}{\kappa} x_e$. Therefore the sum of the lengths $h_e - l_e$ of intervals over edges e in p not queried is at most κ. □

4 The General Case

We now consider the case of arbitrary $\kappa \ge 0$ and arbitrary intervals $[l_e, h_e]$.

Proposition 6 *A choice of queried edges guarantees error κ if and only if for each s-t path p in P there is a path q in P (possibly $q = p$) such that $H - L + x \le \kappa$. Here H is the sum of h_e over edges e in q and not in p, L is the sum of l_e over edges e in p and not in q, and x is the sum of the interval lengths $h_e - l_e$ over edges e in both p and q that are not queried.*

Proof: Suppose there is a path p such that $H - L + x > \kappa$ for each path q. Answer the queries to edges e in p by l_e, and answer the queries to all other edges f by h_f. Let L_0 be the minimum possible length on p. Then for every path q, the maximum possible length is at least $L_0 + (H - L) + x > L_0 + \kappa$.

Suppose for every path p there is a path q such that $H - L + x \le \kappa$. After the queries, the minimum possible length of a shortest path is the minimum possible length L_0 for some p. The maximum possible length for the corresponding q is at most $L_0 + (H - L) + x \le L_0 + \kappa$. □

Proposition 7 *The general problem is in Σ_2. That is, given a set E' of edges, for every s-t path p it can be decided in polynomial time whether, if the edges* *in E' are queried, there will be a path q with $H - L + x \leq \kappa$, where L, H, and x are as defined above.*

Proof: Suppose we are given a set E' of edges to query. Then fix an arbitrary s-t path p. Compute L' as $\sum_{e \in p} l_e$.

For every edge e in E we will define t_e as follows: if $e \in p \cap E'$, that is, if e is a queried edge on the path p, then $t_e = l_e$; otherwise $t_e = h_e$. Find the shortest s-t path q' in the graph G, using the lengths defined by t_e, and let H' be the length of q' using these lengths. Then we claim that p has a q satisfying $H - L + x \leq \kappa$ if and only if $H' - L' \leq \kappa$. Clearly H', L', and $H' - L'$ can be calculated in polynomial time, and so the result will follow.

In fact, it is easy to see that q' itself is a q satisfying $H - L + x \leq \kappa$ for p. Recall that H is the sum of upper bounds for edges in q but not p, L is the sum of lower bounds for edges in p but not q, and x is the sum of the interval widths of unqueried edges shared by p and q. Let $E_Q = p \cap q \cap E'$ denote the set of queried edges shared by p and q, and let $E_U = p \cap q \cap (E \setminus E')$ denote the set of unqueried edges shared by the two paths. Then $H' \leq H + \sum_{e \in E_Q} l_e + \sum_{e \in E_U} h_e$, and $L' = L + \sum_{e \in E_Q} l_e + \sum_{e \in E_U} l_e$. Therefore, $H' - L' = (H - L) + \sum_{e \in E_U} (h_e - l_e)$ $= H - L + x$, and so $H' - L' \leq \kappa \Leftrightarrow H - L + x \leq \kappa$. □

Theorem 9 *If each path in P contains at most r nontrivial edges, then the problem is as easy and as hard to approximate as r-hypergraph vertex cover, for $r = O(\log n)$; in particular, it has an r-approximation algorithm.*

Proof: The hardness was shown in Theorem 8 with $r = \kappa + 1$. For the easiness, consider each path p. For a choice of edges X to be queried in p, we can determine whether $H - L + x > \kappa$ for each path q as in Proposition 7. If this is the case, then at least some nontrivial edge in p not in X must be queried, giving a set of at most r nontrivial edges to be covered, and thus defining a corresponding hyperedge of size at most r. The number of choices X to be considered for p is at most 2^r, thus polynomial in n. □

Theorem 10 *If the nontrivial edges of G have r colors, so that the nontrivial edges in each path p from P have distinct colors, then the problem is as easy and as hard to approximate as r-partite hypergraph vertex cover, for $r = O(\log n)$; in particular, it has an $\frac{r}{2}$-approximation algorithm [9,5].*

Proof: Let R be an instance of r-partite hypergraph vertex cover, with r parts R_i. Construct a DAG consisting of a path of length r, where the ith edge is replaced by $|R_i|$ parallel edges with intervals $[0, 1]$ of color i. Each hyperedge in R thus corresponds to a path in the DAG since it selects an element from R_i for each i. Setting $\kappa = r - 1$ forces each hyperedge to be covered, proving the hardness.

The easiness is as in Theorem 9. The hyperedges correspond to the nontrivial edges in p not in X for some X, so they all have elements of different colors, i.e., elements in different parts R_i of they r-partite hypergraph. $\square$

Corollary 1 *If P consists of all paths in G, and each such path has at most r nontrivial edges, then the problem has an $\frac{r}{2}$-approximation algorithm, if r is constant. In particular, the case $r = 2$ can be solved exactly. (The case $r = 3$ is NP-hard to approximate within $1 + \delta$ for some $\delta > 0$.)*

Proof: Assign to each nontrivial edge e a color i which is the maximum i such that e occurs as the ith nontrivial edge in a path p. There are at most r colors, and all the nontrivial edges in each path p have different colors. The choice of at most r nontrivial edges for a path p can be assumed to determine p, since we can choose the remaining trivial edges so as to minimize the length of the path. Thus the number of paths to be considered is $O(m^r)$.

The hardness for $r = 3$ was established in Theorem 5. $\square$

A *bi-tree* is a series-parallel graph consisting of a tree S with root s, edges oriented down from the root, a tree T with root t, edges oriented up to the root, where the leaves of S coincide with the leaves of T.

Theorem 11 *Bi-trees have a $(1 + \delta)$-approximation algorithm for all $\delta > 0$, and an exact polynomial time algorithm if κ is bounded by a polynomial.*

Proof: Consider the optimal solution. Consider a series-parallel subgraph G' of the given bi-tree which is given by a subtree rooted at some s' of the tree rooted at s, and a subtree rooted at some t' of the tree rooted at t. The paths p going through G' such that the corresponding q satisfying Proposition 7 does not go through G' are determined by a lower bound L_0 on the sum of l_e over the edges e of p in G'. Thus, there are at most n different scenarios for which paths p going through G' must have their corresponding q going through G' as well. If the set of such p is nonempty, then the sum x_0 of the $h_e - l_e$ for edges from s to s' or from t' to t not queried must satisfy $x_0 \leq \kappa$, for a total of κn scenarios for each such G'.

Each such G' is obtained by combining two G'_1, G'_2 in parallel, or by taking a G'_1 having some s'_1, t'_1 as source and sink, and adding either an edge (s', s'_1) or an edge (t'_1, t'). The optimal solution for each scenario for G can be obtained from the optimal solution for the scenarios for G'_1 for this last case or for the scenarios for both G'_1, G'_2 in the previous case, because given x_0 it is easy to determine which paths q in G'_1 take care of paths p in G'_2 and vice versa.

For large κ, we proceed as in Theorem 7 and introduce at each step an approximation factor of $1 + \frac{\delta}{n}$, so that the total factor incurred is at most $(1 + \frac{\delta}{n})^n \approx 1 + \delta$. $\square$

Note that the special case where the bi-tree is a single path is the same as knapsack [11] and thus NP-complete.

Theorem 12 *If P consists of all paths in G, and G is made up of components G_i in series, where each G_i has an α-approximation algorithm, then G has an $\alpha(1+\delta)$-approximation algorithm for all $\delta > 0$, and an α-approximation algorithm if κ is bounded by a polynomial.*

Proof: A solution must combine κ_i for each G_i that add up to at most κ. We proceed inductively for the possible values of partial sums $\kappa_1 + \cdots + \kappa_i$, and combine solutions accordingly.

For large κ, we proceed again as in Theorem 7 and introduce at each step an approximation factor of $1 + \frac{\delta}{n}$, so that the total factor incurred is at most $(1 + \frac{\delta}{n})^n \approx 1 + \delta$. $\square$

Acknowledgments

The authors gratefully acknowledge the reviewers, who suggested many clarifying revisions.

References

[1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. "Fast estimation of diameter and shortest paths (without matrix multiplication)." *SIAM Journal on Computing* 28(1999):1167–1181.

[2] P. Berman and M. Karpinski. "On some tighter inapproximability results." DIMACS Technical Report 99–23 (1999).

[3] X. Deng, T. Kameda, and C. Papadimitriou. "How to learn an unknown environment." To appear in the *Journal of the ACM.*

[4] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. "Computing the median with uncertainty." In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 2000, pages 602–607.

[5] Z. Füredi. "Matchings and covers in hypergraphs." *Graphs and Combinatorics* 4(1988):115–206.

[6] O.H. Ibarra and C.E. Kim. "Fast approximation algorithms for the knapsack and sum of subsets problems." *Journal of the ACM* 22(1975):463–468.

[7] O. Karasan, M. Pinar, and H. Yaman. "The robust shortest path problem with interval data." Manuscript, August 2001.

[8] S. Khanna and W. Tan. "On computing function with uncertainty." In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2001, pages 171–182.

[9] L. Lovász. "On minimax theorems of combinatorics." Doctoral Thesis, Mathematikai Lapok 26(1975):209–264. (Hungarian)

[10] R. Montemanni and L. M. Gambardella. "An algorithm for the relative robust shortest path problem with interval data." Tech. Report IDSIA-05-02, 2002.

[11] C. Olston and J. Widom. "Offering a precision-performance tradeoff for aggregation queries over replicated data." In *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000, pages 144–155.

[12] C.H. Papadimitriou and M. Yannakakis. "Shortest paths without a map." In *Proceedings of the 16th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science 372(1989):610–620.