Optimal Path Discovery Problem with Homogeneous Knowledge

Christopher Thraves Caro, Josu Doncel, Olivier Brun

Optimal Path Discovery Problem with Homogeneous Knowledge

Christopher Thraves Caro* Josu Doncel† Olivier Brun‡

Abstract

Consider the following problem: given a complete graph $G = (V, E)$, two nodes s and t in V, and a positive hidden value $f(e)$ for each edge $e \in E$, discover an $s - t$-path P that minimizes the value $F(P)$, for some objective function F. The issue is that the edge values $f(\cdot)$ are hidden, hence, to discover an optimal path, it is required to uncover the value of some edges. The goal then is to discover an optimal path by means of uncovering the least possible amount of edge values. This problem, named the *Optimal Path Discovery* (OPD) problem, is an extension of the well known *Shortest Path Discovery* problem in which $f(e)$ represents the length of e, and $F(P)$ computes the length of P.

In this paper, we study the OPD problem when the only previous information known about the $f(\cdot)$ values is that they fall in the interval $(0, \infty)$ for all $e \in E$. We first study the number of uncovered edges as a measure to evaluate algorithms. We see that this measure does not differentiate correctly algorithms according to their performance. Therefore, we introduce the *query ratio*, the ratio between the number of uncovered edges and the least number of edge values required to solve the problem. We prove a $1 + 4/n - 8/n^2$ lower bound on the query ratio and we present an algorithm whose query ratio, when it finds the optimal path, is upper bounded by $2 - 1/(n-1)$, where $n = |V|$. Finally, we implement different algorithms and evaluate their query ratio experimentally.

Keywords: Optimal path, Query ratio, Shortest path discovery problem, Lower and upper bounds.

*Depto. Ing. Matemática, Fac. de Ciencias Físicas y Matemáticas, Univ. de Concepción, Av. Esteban Iturra s/n, Casilla 160-C, Concepción, Chile.

†Corresponding author. Applied Mathematics and Statistics and Operations Research department. University of the Basque Country. Barrio sarriena s/n. 48920 Leioa. Spain. E-mail: josu.doncel@ehu.eus

‡CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France & Univ. de Toulouse, LAAS, F-31400 Toulouse, France.

1 Introduction

Shortest path problems are arguably among the most fundamental problems studied in computer science and have a variety of applications in as diverse areas as computational geometry, geographical information systems, network optimization and robotics, to mention just a few. This is reflected by the large body of literature devoted to shortest path problems and by the many algorithms that have been proposed to solve them (cf. [1]). Nevertheless, in some situations the difficulty lies more in collecting the information on the graph than in computing a shortest path. In this paper, we consider situations in which the cost of obtaining the length of the edges dominates the total cost of computing a shortest path.

For instance, such a situation occurs with routing overlays deployed over the Internet [33]. This is actually our main motivation for studying the problem addressed in the present paper. Consider a set of n nodes located at various spots in the Internet, and imagine that a source node wants to deliver a message to a destination node with the best performance possible according to a certain metric (e.g., minimization of the transmission delay). It may happen that the direct Internet path between the source and destination nodes has an unacceptable performance. In that case, provided that other nodes can act as relays for the message, it may be worth searching for the optimal path in the complete graph formed by all nodes. However, active monitoring of the quality of Internet paths by sending probe packets is costly. Therefore, in that case, the cost of finding a shortest path is clearly dominated by the cost of obtaining the costs associated with the edges [32]. As discussed in [30], the same difficulty appear in segmentation lattice based character recognition systems, speech recognition, hierarchical planning, or exploration of unsafe environments.

In such situations, it makes sense to seek to minimize the amount of information required to compute a minimum-cost path, that is, to collect the value of as few edges as possible enabling to guarantee that a minimum-cost path has been discovered. To study such situations, we introduce the *Optimal Path Discovery* (OPD) problem in graphs. In the OPD problem the costs of the edges are initially unknown but can be discovered by *querying* an oracle. The goal is to find an optimal path between two given nodes querying the minimum number of edges. In some cases, rather than requiring an optimal path, we might be less ambitious and be satisfied with a path whose cost is at most α times that of an optimal path, for some $\alpha \geq 1$.

Note that in our motivation, the Internet, there is (almost) always connection between two nodes. Hence, under the eyes of a user, the network

is complete. That will be actually our assumption, we work with complete graphs. When we assume that the graph is complete, we consider that the actual graph is connected, i. e., there exists a path between any two nodes, nevertheless, we, as users, are blind to the structure of the path and see it as a direct connection between the nodes. We remark that this assumption does not harm our model since it is the weakest assumption we can take: no previous knowledge about the network topology or the actual f-values of the edges.

Problem Statement: Let $G = (V, E)$ be a complete undirected graph with n nodes. It is assumed that each edge $e \in E$ has an *unknown positive value* $f(e) > 0$. The value of a set of edges, and hence of a path in G, depend on the individual values of the edges it is comprised of and is determined by a function $F : 2^E \to [0, \infty)$, which for the moment will be assumed to be $F(H) = \sum_{e \in H} f(e)$.

As mentioned before, given two distinguished vertices s and t in G, the goal is to find a simple path between these vertices that provides some performance guarantee. Let $\mathcal{P}_{s,t}$ be the set of all paths connecting s and t, and let $\delta^*_{s,t}$ be the (unknown) *value of an optimal path*[1]:

$$\delta^*_{s,t} = \min_{P \in \mathcal{P}_{s,t}} F(P)$$

Then, given $\alpha \geq 1$, the goal is to discover an *α-approximation* of an optimal path, that is, a path $P_{s,t} \in \mathcal{P}_{s,t}$ such that $F(P_{s,t}) \leq \alpha \cdot \delta^*_{s,t}$. The issue is that the edge values $f(\cdot)$ are unknown. It is however possible to query an oracle to determine the value of an edge. When the oracle is queried for the value of an edge $e \in E$, it reveals the value $f(e)$. For short, we say that the edge e is *uncovered*.

Obviously, to be able to certify that an α-approximation between s and t has been discovered, the set of uncovered edges has to provide enough information for a lower bound on $\delta^*_{s,t}$ to be inferred. Indeed, as long as nothing is known on $\delta^*_{s,t}$ (except for the fact that it is positive and finite), there is absolutely no guarantee on the values of each path $P_{s,t} \in \mathcal{P}_{s,t}$ with respect to that of an optimal path. A set of uncovered edges allowing to certify that an α-approximation between s and t has been discovered will be called an *α-certificate* of an $s - t$-path. More precisely, an α-certificate of an $s - t$-path is defined as a set of edges $\mathcal{C} \subseteq E$ such that

[1] Although we focus on discovering a minimum-cost path, the extension to a maximization problem is straightforward.

- the values of the edges in $\mathcal{C}$ are known,
- $\mathcal{C}$ contains an $s-t$-path, the so-called *proposed* path, and
- there are enough uncovered edges in $\mathcal{C}$ to guarantee that the proposed path is an α-approximation.

We remark that every edge in the proposed path belongs to the α-certificate, implying that its value is fully known. Note also that for any value of $\alpha \geq 1$, there always exists at least one α-certificate since the set E that contains all the edges of the graph is an α-certificate for any α.

With the above definitions in mind, we can now define the OPD problem as follows:

Definition 1. *Given a complete graph $G = (V, E)$, special nodes s and t and a parameter α:*

$$\begin{aligned} &\textit{minimize } |\mathcal{C}| && \text{(OPD)}\\ &\textit{subject to} \\ &\qquad \mathcal{C} \textit{ is an } \alpha\textit{-certificate of an } s-t\textit{-path}, && (1) \end{aligned}$$

where $|\mathcal{C}|$ denotes the cardinality of $\mathcal{C} \subseteq E$.

Although the problem as presented above assumes an additive cost function $F(H) = \sum_{e \in H} f(e)$, the results presented in the sequel are valid for more general cost functions. All results hold for any function $F : 2^E \to [0, \infty)$ such that for all $H \subset E$, $F(H)$ is fully determined by the f-values of the edges in H (and does not depend on the identity of the edges in H), and meeting the following additional technical conditions:

(i) For all $e \in E$, $F(e) = f(e)$.

(ii) For all $H \subseteq E$, in order to compute $F(H)$ it is required to know $f(e)$ for all $e \in H$.

(iii) $F(H) = 0$ if and only if $H = \emptyset$.

(iv) Given $H, H', H'' \subseteq E$ such that $H \cap H' = \emptyset$ and $H \cap H'' = \emptyset$, it holds that:

$$F(H') \leq F(H'') \iff F(H \cup H') \leq F(H \cup H'').$$

(v) Given $H \subset H' \subseteq E$, it holds that $F(H) < F(H')$.

It is immediate to check that the function $F(H) = \sum_{e \in H} f(e)$ meets the above conditions. Another example is the function $F(H) = \prod_{e \in H} f(e)$ for a maximization problem[2]. In contrast, $F(H) = \min_{e \in H} f(e)$ is an example of a function which does not satisfy these conditions. The rationale behind these conditions is as follows. Condition (iii) imposes that at least one edge of a path is required to have a lower bound on the value of that path. Conditions (iv) and (v) guarantee that $F(\cdot)$ is a set-wise monotonically increasing function.

We assume as well that the function F can be computed in polynomial time. Therefore, the OPD problem arises in a context in which obtaining the value f of the edges dominates the total cost of computing the optimal path according to F.

Two different performance metrics: Given an algorithm $\mathcal{A}$ solving the OPD problem, let $U(\mathcal{A}(I))$ be the α-certificate given by this algorithm as a solution to an instance I. We shall consider two metrics for assessing the performance of the algorithm $\mathcal{A}$.

- The *number of queried edges* of $\mathcal{A}$ for instances of size[3] n is defined as:

$$\beta_n = \max_{I:|I|=n} |U(\mathcal{A}(I))|.$$

- The *query ratio* of $\mathcal{A}$ is defined as:

$$\max_{I:|I|=n} \frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|},$$

  where $|\mathcal{C}^{\alpha}_{\min}(I)|$ is the size of a smallest (in cardinality) α-certificate for instance I.

In Section 3, we shall use the the number of queried edges to prove that there is no efficient algorithm for the OPD problem, in the sense that for any algorithm there always exists a bad instance for which the algorithm will query a number of edges which is of the same order of magnitude than the total number of edges. Interestingly, any algorithm may still have to do

[2]For a maximization problem, the first assumption does not change, the second assumption would read $F(H) = \infty \iff H = \emptyset$, $\forall H \subset E$, whereas the fourth assumption would be $F(H') < F(H)$.

[3]We define the size of an instance I as the number of nodes n of the complete graph G, and it is denoted by $|I|$.

a number of queries of order n^2 even if we are ready to accept a significant performance degradation by requiring only an α-approximation of an optimal path for any large value of α. In that respect, the number of queried edges does not allow to correctly differentiate algorithms according to their performance. As we will see, the *query ratio* is a much more appropriate performance measure.

Contributions: We prove that, for any instance with n nodes, any algorithm will need to query at least $n-1$ edges to find a feasible solution. Moreover, we also show that for any algorithm there exists a bad input such that the number of edges queried by the algorithm will be of the same order of magnitude than the total number of edges. We then use the query ratio to evaluate the performance of algorithms that solve the OPD problem. We prove that any algorithm has a query ratio of at least $1 + \frac{4}{n} - \frac{8}{n^2}$ for any α and propose an algorithm whose query ratio is upper bounded by $2 - \frac{1}{n-1}$ when $\alpha = 1$.

Paper organization: The rest of the paper is organized as follows. In Section 2 we put our work in the context of the existing literature. We prove lower bounds on the number of queried edges in Section 3. In Section 4, we establish lower and upper bounds on the query ratio. Section 5 is devoted to the comparison of the proposed algorithm with other methods from the literature, both from a theoretical point of view and from an experimental point of view. Finally, in Section 6, some conclusions are drawn and future research directions are proposed.

2 Related Work

Shortest path problems have been extensively studied in a setting where the decision-maker has full information about the graph [4, 14, 13, 18, 6]. Researchers have also considered the use of additional knowledge on the graph in order to speedup the search [22]. It is worth noticing that standard algorithms, such as A^* [16] or Dijkstra's algorithm [9], perform poorly for solving the *Shortest Path Discovery Problem*. Indeed, we can give an instance with n nodes where the query ratio of these algorithms is as bad as $n/2$. Consider an input in which the direct edge has value $1/2$, any other edge that is incident to the destination node has value 1, and the rest of the edges have value $\epsilon \approx 0$. Without loss of generality, we assume that the search algorithm starts its search in the source node. The algorithm

will query all the edges incident to the source node. Then, it will pick any node different from the source and the destination, and will query all the edges incident to that node. The algorithm will repeat that process until it has queried all the edges incident to any node that is not the source nor the destination node. Hence, in total, such an algorithm will perform $n(n-1)/2$ queries. Nevertheless, the certificate of minimum size for this instance has size $(n-1)$.

Our work is somewhat related to the vast literature dealing with problems in which the actual data values are not known precisely but can be updated at some cost [26, 19, 11, 21, 5]. Generally speaking, in that type of problems, a problem instance consists of a function of n inputs to be computed (e.g., the maxima) and a specification of the possible values each of the input might take (e.g., a set of n real intervals). The goal is to decide which of the inputs should be updated in order to compute the function with a given precision at minimum cost. A major difference with our work is of course that in these references, no graph structure is assumed. It is also worth mentioning the works on robust shortest path and spanning tree problems [27, 25, 3, 20, 31] with interval data, although it is usually assumed that the uncertain data cannot be updated. A noticeable exception is [12], but which considers only the off-line setting.

The most closely related work is the *Shortest Path Discovery Problem* introduced in [30]. This problem and the OPD problem coincide when the cost of a path is the sum of the cost of the edges it is comprised of and $\alpha = 1$. In [30], the authors propose a greedy algorithm that increments the search following the shortest path known at each step. We extend the *Shortest Path Discovery Problem* in several directions. First, we consider a broader class of cost functions, and we relax the constraint that an optimal path has to be discovered, allowing the discovered path to be an α-approximation. Second, whereas in [30] the performance of algorithms was measured with the number of queries, we show that this performance measure does not provide enough information and propose rather to compare algorithms through their query ratios. We also propose a bidirectional search algorithm [29, 24, 23, 8, 15, 7] for solving the OPD problem. Although this algorithm has performances similar to the greedy algorithm of [30] in the case of an additive cost function, its main interest is that the analysis of its query ratio is far much simpler.

Another closely related work is [2], where the authors investigate the problem of learning a shortest path in a network with unknown link delays thanks to end-to-end measurements (that is, by transmitting probe packets along the different paths and observing their trip times). Although this problem is similar to the *Shortest Path Discovery Problem*, the main differ-

ence is that only the total cost of a path can be queried.

In [17], the authors consider the minimum spanning tree problem with uncertainty and propose an algorithm with query ratio 2. Furthermore, they show that this query ratio is the best possible among deterministic algorithms for the minimum spanning tree problem.

Finally, in [10] the authors extend the framework to cheapest set problems with uncertainty that englobe previously studied problems such as the minimum spanning tree, or the minimum matroid based problem under uncertainty. For the cheapest set problems with uncertainty, the authors present an algorithm that makes $d \cdot OPT + d$ queries, where OPT is the optimal number of queries required to solve the problem and d is the maximum cardinality of a feasible set in a given instance. They also provide an algorithm with query ratio 2 for the minimum matroid base problem. It is worth noticing that in this work we consider a family of problems that can be understood as cheapest set problems with uncertainty. Nevertheless, we consider unbounded uncertainty areas in contrast with the bounded uncertainty areas considered previously. Furthermore, due to the particular structure of our problem, we obtain a query ratio of $2 - 1/(n-1)$, which improves over the general $d \cdot OPT + d$ queries for the general case, and over the $2 \cdot OPT$ for the minimum spanning tree problem.

3 Lower Bound on β_n

In this section, we present lower bounds on the number of queried edges required by any algorithm providing any α-approximation.

Lemma 1. *For any instance of the OPD problem with $\alpha \geq 1$, all α-certificates contain a cut-set in G such that the corresponding cut places s in one set of the partition and t in the other.*

Proof. First, we remark that an α-certificate contains a proposed path by definition. Therefore, the value $F(P_{s,t})$ of the proposed path $P_{s,t}$ is fully determined. Second, we remark that, in order to provide any finite approximation guarantee α, the α-certificate needs to provide a bound for the value of an optimal path between s and t.

Now, consider an instance I of the OPD problem and an α-certificate $\mathcal{C}$. Let us assume that the α-certificate does not contain a cut-set that separates s from t. Hence, there exists a path $P^*_{s,t}$ between s and t such that $P^*_{s,t} \cap \mathcal{C} = \emptyset$. Since only edges in $\mathcal{C}$ have a known value, the value of $P^*_{s,t}$ is totally unknown for the α-certificate $\mathcal{C}$. Hence, according to Condition (iii),

the value of $P^*_{s,t}$ can be estimated by 0. Therefore, $\mathcal{C}$ can not guarantee a lower bound on the value of an optimal path. Thus, there is a contradiction with the fact that $\mathcal{C}$ is an α-certificate, since $\mathcal{C}$ cannot guarantee any approximation for its proposed solution. □

Lemma 2. *For any instance of the OPD problem, let $\mathcal{C}$ be any set of edges that contains a path between s and t and a cut-set in G such that the corresponding cut places s in one part and t in the other. Hence, $\mathcal{C}$ is an α-certificate for some finite $\alpha \geq 1$.*

Proof. Consider a set of edges $\mathcal{C}$ as in the statement of the Lemma. Let us denote by $P^{\mathcal{C}}_{s,t}$ the path in $\mathcal{C}$ between s and t. It holds that $P_{s,t} \cap \mathcal{C} \neq \emptyset$ for any path $P_{s,t} \in \mathcal{P}_{s,t}$. Set

$$\alpha = \frac{F(P^{\mathcal{C}}_{s,t})}{\min\{F(P_{s,t} \cap \mathcal{C}) : P_{s,t} \in \mathcal{P}_{s,t}\}}.$$

Since $\min\{F(P_{s,t} \cap \mathcal{C}) : P_{s,t} \in \mathcal{P}_{s,t}\} \leq \delta^*_{s,t}$, it holds that

$$F(P^{\mathcal{C}}_{s,t}) \quad \leq \quad \frac{F(P^{\mathcal{C}}_{s,t})}{\min\{F(P_{s,t} \cap \mathcal{C}) : P_{s,t} \in \mathcal{P}_{s,t}\}}\, \delta^*_{s,t} \coloneqq \alpha\ \delta^*_{s,t}.$$

We observe that α is finite and not necessarily bounded as a function of n. Therefore, $\mathcal{C}$ is an $\alpha-$certificate for the above value of α.

□

Lemma 1 allows us to present lower bounds on the number of queries β_n required so that an algorithm can guarantee a finite α-approximation.

Corollary 1. *For any algorithm that solves the OPD problem for a finite approximation $\alpha \geq 1$, it holds that $\beta_n \geq n - 1$.*

This is a direct consequence of Lemma 1 and the fact that the smallest cut-set in the complete graph has size $n - 1$.

Nevertheless, the previous lower bound for β_n is optimistic since there exist cases in which any algorithm needs to uncover strictly more than $n-1$ edges in order to provide a finite approximation.

Lemma 3. *For any $\alpha \geq 1$ and any integer $1 \leq p \leq n/2$, there exists an instance of the OPD problem with approximation factor α so that any algorithm requires at least $p \cdot (n - p)$ uncovered edges in order to provide an α-approximation.*

Proof. We prove this Lemma via the construction of an instance that certifies the conditions stated in the Lemma. Consider the complete graph with n nodes. Let us split the set of nodes in one set of size p that contains s and one set of size $n-p$ that contains t. We set the values $f(\cdot)$ of the edges as follows: each edge e with its two endpoints in the same set has value $f(e) = \epsilon$. The direct edge (s,t) has value $f(s,t) = b$, where $b > \epsilon$. Each edge e with its endpoints in different sets (except for the direct edge) has value $f(e) > \alpha b$. In such a graph, we have that $\delta^*_{s,t} = f(s,t)$. Besides, using that $F(e) = f(e)\ \forall e \in E$, and the condition (v) on the function $F(\cdot)$, for any $P_{s,t}$, it holds that $F(P_{s,t})/\delta^*_{s,t} > \alpha$, except for the direct path that verifies that $F(s,t)/\delta^*_{s,t} = 1$. Therefore, the only α-approximation is indeed the direct path (s,t).

Nevertheless, in order to guarantee that the only α-approximation is the direct path (s,t), any algorithm needs to uncover at least the cut-set of size $p\cdot(n-p)$ that contains all the edges of value αb. Thus, any algorithm requires at least $p \cdot (n-p)$ uncovered edges in order to provide an α-approximation. □

From this result, considering $p = n/2$, it follows the following corollary.

Corollary 2. *For any algorithm $\mathcal{A}$ that solves the OPD problem for a finite approximation factor $\alpha \geq 1$, there exists an instance so that $\mathcal{A}$ requires at least $n^2/4$ uncovered edges in order to provide an α-approximation.*

According to Corollary 2, for any algorithm and any value $\alpha \geq 1$, it is always possible to find a bad instance such that the number of edges uncovered by the algorithm will be of the same order of magnitude than the total number of edges. Therefore we change our aim. In the rest of the document, we focus on the study of the query ratio of algorithms that solve the OPD problem. We believe that the query ratio is a fair measure to evaluate the performance of algorithms for this problem since it expresses how far is the number of queries asked by an algorithm with respect to the best possible that any algorithm can perform in that instance.

4 Lower and Upper Bounds on the Query Ratio

In this section, we concentrate on the study of the query ratio. We present a lower bound and an upper bound on the query ratio. The first bound is obtained via the design of a malicious adversary that plays the role of the oracle. This adversary works for any algorithm. For the second bound, we present an algorithm and analyze its query ratio.

4.1 An Adversary for Any Algorithm

Lemma 4. *For any $\alpha \geq 1$ and for any algorithm $\mathcal{A}$ that solves the OPD problem, there exists an instance I with approximation factor α such that the following inequality holds:*

$$\frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} \geq 1 + \frac{4}{n} - \frac{8}{n^2}.$$

Proof. See Appendix A.1. □

A direct consequence of Lemma 4 is the following Theorem.

Theorem 1. *For any $\alpha \geq 1$ and for any algorithm $\mathcal{A}$ that solves the OPD problem, it holds the following inequality for the query ratio,*

$$\max_{I} \frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} \geq 1 + \frac{4}{n} - \frac{8}{n^2}.$$

4.2 An Algorithm that Searches from the Source and the Sink

In this Subsection, we present an algorithm that solves the OPD problem for any approximation factor $\alpha \geq 1$. We remark that in the proposed algorithm $\alpha \geq 1$ is a parameter of the instance, and it can be set arbitrarily. Therefore, the proposed algorithm is a parametrized algorithm that guarantees the approximation factor that we arbitrarily decide to obtain. We compute the query ratio of such an algorithm for the particular case when $\alpha = 1$ proving that it never queries more than two times the size of the smallest certificate of an instance. A precise description of the algorithm is presented in Algorithm 1.

We now explain how the algorithm works. First, it initializes the values s^* and t^* to s and t respectively (see line 1 in Algorithm 1). The algorithm works in rounds. At each round, the algorithm advances one step further in a double search that starts from nodes s and t. First, Algorithm 1 adds s^* (resp. t^*) to the set m_s (resp. m_t). Then, it queries the edge (s^*, t^*) as well as all the edges of the form (s^*, u) and (u, t^*) where u are all the nodes not belonging to m_s or m_t (see lines 5 to 7 in Algorithm 1). The algorithm then computes the optimal path between s and t that contains only uncovered edges which is denoted $PATH_{prop}$ (see lines 8 and 9 in Algorithm 1). Then, the algorithm picks the closest nodes to s and t among the nodes that have

Algorithm 1 Algorithm for Optimal Path Discovery Problem

1: INITIALIZE sets $m_s = \{\emptyset\}$, $m_t = \{\emptyset\}$;
paths $PATH_{prop} = \emptyset$;
paths $PATH_{s,u} = \emptyset$, and $PATH_{u,t} = \emptyset \; \forall u \in V/\{s,t\}$;
variable $approx = \infty$;
variables $s^* = s$ and $t^* = t$;
2: **while** $approx > \alpha$ **do**
3: UPDATE $m_s := m_s \cup s^*$.
4: UPDATE $m_t := m_t \cup t^*$.
5: QUERY $\{(s^*, t^*)\}$.
6: QUERY $\{(s^*, u) : \forall u \in V/\{m_s \cup m_t\}\}$.
7: QUERY $\{(u, t^*) : \forall v \in V/\{m_s \cup m_t\}\}$.
8: COMPUTE the optimal path from s to t containing only uncovered edges.
9: UPDATE $PATH_{prop}$ to the optimal path from s to t containing only uncovered edges.
10: COMPUTE the optimal su-path containing only uncovered edges $\forall\, u \in V/\{m_s \cup m_t\}$.
11: UPDATE $PATH_{s,u}$ to the optimal su-path containing only uncovered edges $\forall\, u \in V/\{m_s \cup m_t\}$.
12: COMPUTE the optimal ut-path containing only uncovered edges $\forall\, u \in V/\{m_s \cup m_t\}$.
13: UPDATE $PATH_{t,u}$ to the optimal ut-path containing only uncovered edges $\forall\, u \in V/\{m_s \cup m_t\}$.
14: COMPUTE $s^* := \text{argmin}_{u \in V/\{m_s \cup m_t\}} F(PATH_{s,u})$.
15: COMPUTE $t^* := \text{argmin}_{u \in V/\{m_s \cup m_t\}} F(PATH_{u,t})$.
16: UPDATE $approx := \frac{F(PATH_{prop})}{F(PATH_{s,s^*} \cup PATH_{t^*,t})}$.
17: **end while**
18: RETURN $P_{s,t} := PATH_{prop}$.

not been previously picked. The closest node to s (resp. t) is denoted by s^* (resp. t^*) (see lines 14 and 15 in Algorithm 1). Finally, the algorithm computes whether $PATH_{prop}$ is an α-approximation in line 16. If that is the case, Algorithm 1 stops and returns $PATH_{prop}$, otherwise, it iterates one more round.

The correctness of the algorithm follows directly from the following two facts. First, the algorithm proposes a fully uncovered path. Second, the proposed path is an α-approximation since in the last round it holds that

$\frac{F(PATH_{prop})}{F(PATH_{s,s^*} \cup PATH_{t^*,t})} \leq \alpha$ and also that $F(PATH_{s,s^*} \cup PATH_{t^*,t})$ is a lower bound on $\delta^*_{s,t}$.

We now analyze the query ratio of Algorithm 1. To do so, we first compute the number of queries performed by the algorithm up to a generic round i.

Lemma 5. *For any instance I of size n with $\alpha \geq 1$, the number of queried edges by Algorithm 1 up to the i-th round is equal to $i(2n - 2i - 1)$.*

Proof. At each round, the algorithm queries $2|V/\{m_s \cup m_t\}| + 1$ edgesaccording to lines 5 to 7 of Algorithm 1. At each round, the sizes of m_s and m_t increases by one. Note that s^* and t^* are different at each round. Otherwise, if at some round $s^* = t^*$, in the previous round the algorithm would have stopped since the union of the optimal path from s to s^* and the optimal path from t^* to t would have been the optimal path from s to t. Hence, *approx* would have been equal to 1. Therefore, at the j-th round, the size of $V/\{m_s \cup m_t\}$ is equal to $(n - 2j)$.

Thus, the number of edges queried by the algorithm up to round i is the sum of the queries at all the previous rounds, i.e.,

$$\sum_{j=1}^{i} (2(n - 2j) + 1) = i(2n - 2i - 1).$$

□

On the other hand, if Algorithm 1 stops after i rounds, we are able to give a lower bound on the size of the smallest certificate of the instance. We give such a lower bound in the following lemma.

Lemma 6. *If Algorithm 1 stops after i rounds in an instance I of size n with $\alpha = 1$, the size of the smallest 1-certificate of that instance is at least $i(n - i)$, i.e.,*

$$|\mathcal{C}^1_{\min}(I)| \geq i(n - i).$$

Proof. See Appendix A.2. □

We are now in position to present the query ratio of Algorithm 1.

Theorem 2. *Let $\mathcal{A}^{OPD}$ denote Algorithm 1. Therefore, for any instance I of size n such that $\alpha = 1$,*

$$\max_I \frac{|U(\mathcal{A}^{OPD}(I))|}{|\mathcal{C}^1_{\min}(I)|} \leq 2 - \frac{1}{n - 1}.$$

Hence, the query ratio of $\mathcal{A}^{OPD}$ is at most $2 - \frac{1}{n-1}$ in these instances.

Proof. From Lemmas 5 and 6, it holds:

$$\frac{|U(\mathcal{A}^{OPD}(I))|}{|\mathcal{C}^1_{\min}(I)|} \leq \frac{i(2n-2i-1)}{i(n-i)} = 2 - \frac{1}{n-i} \leq 2 - \frac{1}{n-1}.$$

□

5 Comparison with Previous Algorithms

In this section we compare the query ratio of Algorithm 1 with the query ratio of two algorithms already presented in the literature. We first analyze the query ratio of a greedy algorithm presented in [30] that aims to solve the SPD problem, a particular case of the OPD problem (nevertheless, it can be easily extended to solve the general OPD problem). Then, we experimentally compare the query ratio of Algorithm 1 with two algorithms also mentioned in [30].

5.1 Analytical Comparison

We first note that our upper bound on the query ratio of Algorithm 1 is strictly less than 2. On the other hand, as mentioned in Section 2, the query ratio of the well known search algorithm A^* (cf. [16]) is $n/2$ which is much higher than the upper bound obtained for the query ratio of Algorithm 1.

We now compare the performance of Algorithm 1 with an extension of the algorithm presented in [30] to solve the SPD problem. In the SPD problem the function $F(\cdot)$ is the sum of the values $f(\cdot)$ of each edge in the set. The SPD problem aims to find the shortest path with a minimum number of queries.

In the OPD problem there is a general function $F(\cdot)$ that applies to set of edges. We denote by $\mathcal{A}^{Gre}_{hom}$ the greedy algorithm of [30] that computes optimal paths according to the function $F(\cdot)$ instead of shortest paths. In this algorithm the values of each edge has to be estimated. We consider in $\mathcal{A}^{Gre}_{hom}$ that all the edges are initially estimated to 0, i.e., the knowledge of the real values of the edges is homogeneous and set to the minimum value.

In its original version for the SPD problem, the $\mathcal{A}^{Gre}_{hom}$ algorithm works as follows. This algorithm works in rounds. In each round, it computes the shortest path between s and t with the current knowledge (all the queried edges up to the current round) and the initial estimations. If all the edges

of the shortest path had been previously queried, it stops and returns that path as the shortest path. Otherwise, it queries all the edges not queried yet on the shortest path and continues. Ties are broken according to the number of edges of the path, where the priority is given to a path with the least amount of edges. Besides, there is a policy θ_{Gre} that establishes priorities given to the paths with the same number of edges.

For this analysis we consider that in each step of Algorithm 1 there are several rounds. In the first round of each step, the algorithm queries the edge (s^*, t^*). In each of the following rounds, the algorithm selects a two-hop path formed by (s^*, u) and (u, t^*) according to a given policy θ_{OPD}, and it queries the edges (s^*, u) and (u, t^*). We also consider that, at each step, the algorithm computes its current approximation factor and it stops if this value is less than the desired α.

Lemma 7. *For any instance I of size n of the OPD problem, it holds that:*

- *If θ_{Gre} and θ_{OPD} provide the same ordering in the selection of paths*

$$\Rightarrow \quad U(\mathcal{A}^{OPD}(I)) = U(\mathcal{A}^{Gre}_{hom}(I)).$$

- *Otherwise,*

$$|U(\mathcal{A}^{OPD}(I)) - U(\mathcal{A}^{Gre}_{hom}(I))| \leq 2(n-2).$$

Proof. See Appendix B.1. □

Using this result we state that the query ratio of algorithm $\mathcal{A}^{Gre}_{hom}$ and Algorithm 1 is the same for any instance of the OPD problem.

Theorem 3. *For any instance I of size n of the OPD problem, it holds that:*

$$\frac{|U(\mathcal{A}^{Gre}_{hom}(I))|}{|\mathcal{C}^1_{\min}(I)|} = \frac{|U(\mathcal{A}^{Gre}_{hom}(I))|}{|\mathcal{C}^1_{\min}(I)|}.$$

In Theorem 2 we prove an upper-bound on the query ratio of Algorithm 1 for instances of the OPD problem with $\alpha = 1$ and in Theorem 3 we show that the query ratio of algorithm $\mathcal{A}^{Gre}_{hom}$ and Algorithm 1 coincide for any instance. We remark that the upper-bounds given in these theorems are tight for these algorithms, and it is achieved when the size of the smallest certificate is $n-1$. In this case the algorithms uncover $2n-3$ edges.

It is also important to note that, for the OPD problem with $\alpha = 1$, Algorithm 1 and $\mathcal{A}^{Gre}_{hom}$ have the same query ratio. However, the query ratio of Algorithm 1 is easier to analyze.

5.2 Numerical Comparison

In this section, we experimentally analyze the algorithm presented in Subsection 4.2. We focus on the particular case of the SPD problem and thus we consider $f(e) \in (0, \infty)\ \forall e \in E$ and that $F(\cdot)$ is the sum of the values of the edges, i.e., $F(H) = \sum_{e \in H} f(e)$.

We first analyze the query ratio of Algorithm 1 for the OPD problem with $\alpha = 1$ to compare it with upper-bound presented in Section 4.2. In order to do that, we first need to compute the size of the minimum certificate for a given graph. We denote by $f(e)$ the value of an edge $e \in E$ and $\delta^*_{(s,t)}$ denotes the length of a shortest path between s and t. The minimum certificate can be obtained as a solution of the following 0-1 integer linear program:

$$\begin{aligned} \min \quad & \sum_{e \in E} u(e) \\ \text{s.t.} \quad & \delta^*_{s,t} \leq \sum_{e \in P_{s,t}} u(e) f(e), \forall P_{s,t} \in \mathcal{P}_{s,t} \\ & u(e) \in \{0, 1\}, \end{aligned}$$

where the $\{u(e)\}_{e \in E}$ is the set of variables. According to this formulation, the minimum certificate is formed by the set of edges such that $u(e) = 1$.

Query Ratio Distribution We have used 100 graphs with 8 nodes where the values of the edges are random numbers uniformly distributed between zero and one. We execute Algorithm 1 and we calculate the size of the minimum certificate. In Table 1 we show the distribution of the values of the query ratio that we obtained. For clarity, in Table 1 we only represent the values of the query ratio with frequency larger than 5%. We observe that in the 23% of the cases the query ratio is 1, which means that Algorithm 1 have solved the OPD problem optimally for 23 graphs out of the 100 graphs. On the other hand, the upper-bound given in Theorem 2, that is $7 - 1/7 = 1.8571$, is attained 20 times. The mean of the query ratio over the 100 cases is 1.437.

Comparison of the Query Ratio In the second set of experiments, we compare the performance of Algorithm 1 (Algorithm "algo" in Figure 1) with two modifications of the greedy algorithm presented in [30].

(a) The first algorithm computes at each step the shortest path according to the value of the uncovered edges and the value of the unknown

Query Ratio of Algorithm 1	Frequency
1	23%
1.22	23%
1.2941	7%
1.6923	12%
1.8571	20%

Table 1: Distribution of the query ratio of the presented algorithm for 100 random graphs of 8 nodes

edges set to 0, and uncovers the edge of this path that is closest to the source. We refer to this algorithm as deterministic and in Figure 1 it is represented as "determ".

(b) The second algorithm computes at each step the shortest path according to the value of the uncovered edges and the value of the unknown edges set to 0, and uncovers an edge of this path picked uniformly at random. We refer to this algorithm as "random" in Figure 1.

We consider an instance with 50 nodes where the values of the edges are uniformly distributed random numbers between zero and one. We execute Algorithm 1, the deterministic algorithm and the random algorithm for all the origin-destination pairs of this graph. In each execution of all the algorithms, we compute the evolution of the approximation factor with respect to the number of uncovered edges. We recall that the approximation factor is given by the ratio $\frac{F(PATH_{prop})}{F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)})}$, where $PATH_{prop}$ is the proposed path and $PATH_{(s,s^*)} \cup PATH_{(t^*,t)}$ is a lower bound on the shortest path. In Figure 1 we plot the average of the approximation factor over all the possible source-destination pairs to show the average performance of different algorithms. The y-axis of this figure is in logarithmic scale.

Figure 1 shows that the mean approximation factor of the deterministic algorithm decreases fast during the first 50 queries. However, this algorithm needs to uncover almost 1000 edges to achieve an approximation factor equal to one which is much higher than for the other algorithms. Moreover, we see that the average approximation factor of Algorithm 1 achieves the value 1 for a lower number of uncovered edges comparing with the random algorithm. Furthermore, the average approximation factor of Algorithm 1 is smaller than the deterministic algorithm when the number of uncovered edges is higher than 100 and always less than the random algorithm.

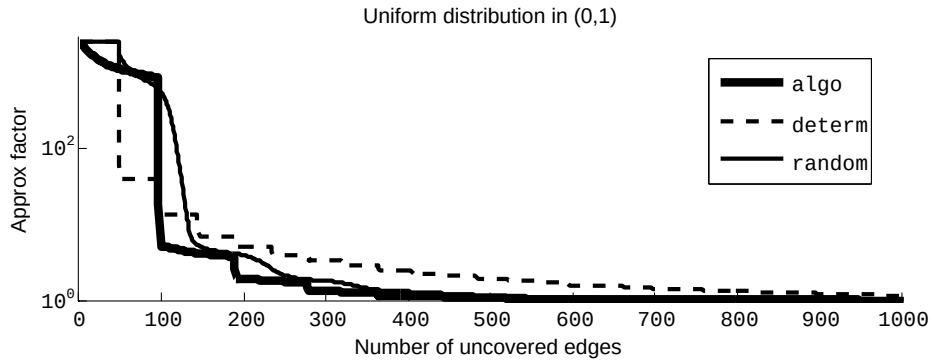

Figure 1: Approximation factor evolution comparison. Uniformly distributed edges and 50 nodes. Y axis in logarithmic scale.

6 Conclusions and Future Work

In this document we present and study the OPD problem. For a given function F that is applied to set of edges, an algorithm that solves the OPD problem aims to find the path that optimizes the value of F when it is applied to a path, while, at the same time, it has to minimize the number of queried edges it uses to find such path.

We show that the number of queried edges, as an absolute measure in order to compare algorithms that solve the OPD problem, does not provide insightful information with respect to algorithms that solve the OPD problem. However, we introduce the query ratio, a measure which provides an important insight into the real quality of an algorithm solving this problem. That is because it compares the number of queries performed by the algorithm with the least amount of queries required to solve the problem. In this document, we give a $1 + 4/n - 8/n^2$ lower bound and we present an algorithm that finds the optimal path with a $2 - 1/(n-1)$ upper bound on the query ratio, where $n = |V|$.

Under our consideration, the most appealing problem that this document leaves open is the gap between the lower and upper bounds. The question is whether there exists an algorithm, or on the contrary an adversary that produces a bad instance for any algorithm, so that the gap is closed. We also consider interesting the comprehension of the trade-off between the approximation factor α for the proposed path and the query ratio of an algorithm. The question is whether when we relax the approximation factor α we obtain better results for the query ratio.

Another open problem is to extend the results presented in this document to instances with heterogeneous knowledge, i.e., when previous knowledge regarding the values of the edges is not necessarily the same for all edges. In the same line, we also consider interesting to study this problem in graphs that are not necessarily complete. Both situations are in fact strongly related since any topology can be modeled as a complete graph with heterogeneous knowledge on the value of the edges. Indeed, it is enough to assume that edges not present in the original graph have values such that they can be directly discarded from any possible solution (e.g. $f(e) = \infty$ for shortest path problems). Therefore, an algorithm would be able to decide not to query an edge that is not present in the graph. In that respect, the two mentioned extensions are closely related and can probably be treated as one single case.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PANACEA Project (www.panacea-cloud.eu), grant agreement n 610764.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1st edition, February 1993.

[2] Noga Alon, Yuval Emek, Michal Feldman, and Moshe Tennenholtz. Economical graph discovery. In *ICS*, pages 476–486, 2011.

[3] Ionu D. Aron and Pascal Van Hentenryck. On the complexity of the robust spanning tree problem with interval data. *Oper. Res. Lett.*, 32(1):36–40, 2004.

[4] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[5] Richard Bruce, Michael Hoffmann, Danny Krizanc, and Rajeev Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005.

[6] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: theory and experimental evaluation. *Mathematical Programming*, 73(2):129–174, 1996.

[7] Henry W. Davis, Randy B. Pollack, and Thomas Sudkamp. Towards a better understanding of bidirectioanl search. In *AAAI*, 1984.

[8] Dennis de Champeaux. Bidirectional heuristic search again. *J. ACM*, 30(1):22–32, January 1983.

[9] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[10] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theor. Comput. Sci.*, 613(C):51–64, 2016.

[11] Tomás Feder, Rajeev Motwani, Rina Panigrahy, Chris Olston, and Jennifer Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003.

[12] Toms Feder, Rajeev Motwani, Liadan O'Callaghan, Chris Olston, and Rina Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1 – 18, 2007.

[13] R.W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[14] L.R. Ford. Network flow theory. Technical Report Paper P-923, RAND Corporation, Santa Monica, California, August 1956.

[15] Subrata Ghosh and Ambuj Mahanti. Bidirectional heuristic search with limited resources. *Information Processing Letters*, 40(6):335 – 340, 1991.

[16] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, pages 100–107, 1968.

[17] Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihal'ák, and Rajeev Raman. Computing Minimum Spanning Trees with Uncertainty. In *25th International Symposium on Theoretical Aspects of Computer Science*, Leibniz International Proceedings in Informatics (LIPIcs), pages 277–288. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2008.

[18] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[19] Simon Kahan. A model for data in motion. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 265–277. ACM, 1991.

[20] Adam Kasperski and PawełZieliński. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Inf. Process. Lett.*, 97(5):177–180, 2006.

[21] Sanjeev Khanna and Wang-Chiew Tan. On computing functions with uncertainty. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 171–182. ACM, 2001.

[22] Richard E. Korf. Optimal path-finding algorithms. In Laveen Kanal and Vipin Kumar, editors, *Search in Artificial Intelligence*, Symbolic Computation, pages 223–267. Springer New York, 1988.

[23] Marco Lippi, Marco Ernandes, and Ariel Felner. Efficient single frontier bidirectional search. In *Proceeding of the Forth International Symposium on Combinatorial Search*, 2012.

[24] Michael Luby and Prabhakar Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(1-4):551–567, 1989.

[25] R. Montemanni and L.M. Gambardella. An algorithm for the relative robust shortest path problem with interval data. Technical Report IDSIA-05-02, Dalle Molle Institute for Artificial Intelligence, 2002.

[26] Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 144–155. Morgan Kaufmann Publishers Inc., 2000.

[27] Hande Yaman Oya E. Karasan, Mustafa C. Pinar. The robust shortest path problem with interval data. Technical report, Bilkent University, Department of Industrial Engineering, 2001.

[28] Christos H Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[29] Ira Pohl. *Bi-directional and Heuristics Search in Path Problems*. PhD thesis, Standford University, 1969.

[30] Csaba Szepesvári. Shortest path discovery problems: A framework, algorithms and experimental results. In *AAAI*, pages 550–555, 2004.

[31] Hande Yaman, Oya E. Karasan, and Mustafa C. Pinar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29(1):31 – 40, 2001.

[32] Olivier Brun, Lan Wang and Erol Gelenbe. Big Data for Autonomic Intercontinental Overlays. *IEEE Jour. Selected Areas in Communications*, 34(3), March 2016.

[33] Nick Feamster, and Hari Balakrishnan, and Jennifer Rexford, and Aman Shaikh, and Jacobus van der Merwe. The Case for Separating Routing from Routers. *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pp 5–12, ACM, Portland, Oregon, USA, 2004.

A Proof of Results in Section 4

A.1 Proof of Lemma 4

Proof of Lemma 4. In order to prove the Theorem, we construct a bad instance I for each algorithm. The construction is made adversarially. We give a malicious adversary that acts as the oracle and, each time that an algorithm uncovers an edge, the adversary gives the value of that edge to the algorithm so that the performance of the algorithm is as bad as possible. The adversary is precisely described in Algorithm 2.

The adversary constructs an instance similar to the instance of Lemma 3, but ad-hoc to each algorithm. The adversary is deciding the values of each edge online at the same time as the algorithm is uncovering each edge. In the instance constructed by the adversary, the optimal path between s and t is always the direct edge (s, t). Furthermore, the direct path is the only α-approximation (for the α determined in the statement). The instance has a partition of the set of vertices with s in one set of the partition and t in the other set.

The exact partition and the size of each set of the partition depends on the algorithm. Nevertheless, the adversary always gives the following values to the edges. Each edge e with its two endpoints in the same set has

Algorithm 2 Adversary that construct a bad instance for any algorithm that solves the OPD problem, where $e_1, e_2 \ldots, e_l$ are the edges queried by the algorithm, i. e., e_i is the edge queried at the i-th step.

INITIALIZE sets $G_s = \{s\}$ and $G_t = \{t\}$;
functions $g(u) = 0$, $\forall u \in V/\{s,t\}$, $g(s) = s$, and $g(t) = t$.
for $i = 1, \ldots, l$ **do**
  if $e_i = (s,t)$ **then**
    REPLAY $f(e_i) = b\gamma$.
  end if
  if $e_i = (s,u)$ such that $g(u) = 0$ **then**
    REPLAY $f(e_i) = \epsilon$ and update $g(u) := s$ and $G_s := G_s \cup \{u\}$.
  end if
  if $e_i = (s,u)$ such that $g(u) = t$ **then**
    REPLAY $f(e_i) = \gamma > \alpha b$.
  end if
  if $e_i = (u,t)$ such that $g(u) = 0$ **then**
    REPLAY $f(e_i) = \epsilon$ and update $g(u) := t$ and $G_t := G_t \cup \{u\}$.
  end if
  if $e_i = (u,t)$ such that $g(u) = s$ **then**
    REPLAY $f(e_i) = \gamma > \alpha b$.
  end if
  if $e_i = (u,v)$ such that $g(u) = g(v) = 0$ **then**
    REPLAY $f(e_i) = \epsilon$.
  end if
  if $e_i = (u,v)$ such that $g(u) = 0$ and $g(v) \in \{s,t\}$ **then**
    REPLAY $f(e_i) = \epsilon$.
    UPDATE $g(u) := g(v)$.
    UPDATE $g(u') := g(v)\ \forall\, u'$ such that there exists a path from u to u' composed by uncovered edges each with value equal to ϵ.
    UPDATE $G_{g(v)} := G_{g(v)} \cup \{u\} \cup \{u' : u' \rightarrow_\epsilon u\}$, where $u' \rightarrow_\epsilon u$ means that u and u' are connected by a path of uncovered edges each with value equal to ϵ.
  end if
  if $e_i = (u,v)$ such that $g(u) \in \{s,t\}$ and $g(v) \in \{s,t\}/g(u)$ **then**
    REPLAY $f(e_i) = \gamma > \alpha b$.
  end if
  if $e_i = (u,v)$ such that $g(u) \in \{s,t\}$ and $g(v) = g(u)$ **then**
    REPLAY $f(e_i) = \epsilon$.
  end if
end for

value $f(e) = \epsilon$. The direct edge (s,t) has value $f(s,t) = b$, where $b > \epsilon$. Each edge e with its endpoints in different sets (except for the direct edge) has value $f(e) > \alpha b$. In any graph with this characteristics, the path with minimum value is the direct path whose value is $\delta^*_{s,t} = b$. Due to the fact that $F(e) = f(e)\ \forall e \in E$, and due to condition (v) on the function $F(\cdot)$, it holds for any path $P_{s,t}$ that $F(P_{s,t})/\delta^*_{s,t} > \alpha$, except for the direct path that verifies that $F(s,t)/\delta^*_{s,t} = 1$. Therefore, the only α-approximation is indeed the direct path (s,t).

Now we describe how the adversary proceeds. Each algorithm can be seen as a sequence of edges to be uncovered. An algorithm queries to the oracle an edge to be uncovered. The oracle answers each query providing the value of that edge. In this particular construction, the adversary plays the role of the oracle. Edges to be uncovered by an algorithm can be grouped in four groups:

i) $\{(s,t)\}$,

ii) $\{(s,u) : u \in V/\{t\}\}$,

iii) $\{(u,t) : u \in V/\{s\}\}$ and,

iv) $\{(u,v) : u \wedge v \in V/\{s,t\}\}$.

The way in which the adversary determines the partition is described in the following lines. Nodes s and t are initially placed one in each set of the partition. Let us denote these sets by G_s and G_t, respectively (see the initialization of function $g(s)$ and $g(t)$ in line 1 of Algorithm 2). If the algorithm queries the edge (s,t) to be uncovered, the adversary answers to the algorithm $f(s,t) = b$ (see lines 3 and 4 in Algorithm 2). If the algorithm queries an edge of the second group *(ii)* to be uncovered and the node u has not been placed in any set of the partition, the adversary answers $f(s,u) = \epsilon$ and places u in the set G_s (see lines 6 and 7 in Algorithm 2). Otherwise, when u has been placed in a set of the partition (it can be only the set G_t), the adversary answers $f(s,u) > \alpha b$ (see lines 9 and 10 in Algorithm 2). The adversary acts equivalently when the algorithm queries an edge of the third group *(iii)*. That is, if u has not been placed in any set of the partition, the adversary answers $f(u,t) = \epsilon$ and places u in the set G_t (see lines 12 and 13 in Algorithm 2). Otherwise, when u has been placed in a set of the partition (it can be only the set G_s), the adversary answers $f(u,t) > \alpha b$ (see lines 15 and 16 in Algorithm 2). Finally, if the algorithm queries an edge of the fourth group *(iv)*, there exist four different cases: u and v have not been placed in any set of the partition, then the adversary answers $f(u,v) = \epsilon$ and

none is placed in any set of the partition (see lines 18 and 19 in Algorithm 2). When one of them has been placed in a set of the partition, say u, and the other one has not, the adversary answers $f(u,v) = \epsilon$ and node v is placed in the same set than u. In this case, node v might have neighbors not yet assigned to a set of the partition as well. In that case, all the nodes in the same connected component than v are placed in the same set as nodes u and v (see lines 21 to 25 in Algorithm 2). Finally, if the two nodes have been placed in a set of the partition, the adversary answers $f(u,v) = \epsilon$ if the two nodes belong to the same set of the partition, or $f(u,v) > \alpha b$ if they belong to different sets (see lines 27 to 31 in Algorithm 2).

We observe that the optimal path from s to t in any instance constructed by the adversary, regardless the algorithm, is the edge (s,t). Moreover, the only α-approximation is the direct path. On the other hand, for any algorithm, we obtain two sets G_s and G_t, that form a partition of V, i.e., $G_s \cap G_t = \emptyset$ and $G_s \cup G_t = V$. The sets G_s and G_t form a partition since every node is added either to the set G_s or to the set G_t. Moreover, since any algorithm needs to present an α-certificate and such certificate must contain a partition of V, every node is added to one set.

The smallest α-certificate consists in all the edges that connect the nodes of both sets of the partition G_s and G_t. Hence, it holds that $|\mathcal{C}^{\alpha}_{\min}(I)| = |G_s| \cdot |G_t|$. Moreover, we see that the number of uncovered edges by the algorithm is the sum of $|\mathcal{C}^{\alpha}_{\min}(I)|$ and the number of uncovered edges in each set of the partition. The graph formed by the uncovered edges in each set of the partition is connected, since by construction there exists a path from s (resp, t) to any node in G_s (resp, G_t). Therefore, the number of uncovered edges in each set of the partition is at least $|G_s|-1$ and $|G_t|-1$, respectively. Thus, it holds that:

$$\begin{aligned}\frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} &\geq \frac{|\mathcal{C}^{\alpha}_{\min}(I)| + |G_s| - 1 + |G_t| - 1}{|\mathcal{C}^{\alpha}_{\min}(I)|} \\ &= 1 + \frac{|G_s| + |G_t| - 2}{|G_s| \cdot |G_t|}.\end{aligned}$$

Since, it also holds that $|G_s| + |G_t| = n$, the fraction $\frac{|G_s|+|G_t|-2}{|G_s|\cdot|G_t|}$ yields its minimum value when $|G_s| = |G_t| = \frac{n}{2}$. Hence, we obtain

$$\frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} \geq 1 + \frac{4}{n} - \frac{8}{n^2}$$

□

A.2 Proof of Lemma 6

Proof of Lemma 6. In this proof we use the following notation. We denote by $m_{s,j}$ (respectively, $m_{t,j}$) the set m_s (respectively m_t) at the end of the j-th round of the algorithm. We also denote by s^*_j (respectively t^*_j) the node that was added to m_s (respectively to m_t) at the j-th round of the algorithm. According to these definitions, we have that in round j,

$$m_{s,j} = m_{s,j-1} \cup \{s^*_j\},\ m_{t,j} = m_{t,j-1} \cup \{t^*_j\},$$

where $m_{s,0} = m_{t,0} = \{\emptyset\}$ and $s^*_1 = s$ and $t^*_1 = t$. We use $m_s = m_{s,i}$ and $m_t = m_{t,i}$ to denote the sets m_s and m_t after Algorithm 1 has finished. Moreover, for any two nodes u, v and a set of nodes $U \subseteq V$, we denote by $P^*_{u,v}|_U$ an optimal path between u and v in the graph induced by the set of nodes U.

First we show that the following inequality holds for all $1 \leq j \leq i$.

$$F(P^*_{s,s^*_{j-1}}|_{m_{s,j-1}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}) \leq F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_j,t}|_{m_{t,j}}) \tag{2}$$

We first consider that the path P^*_{s,s^*_j} visits s^*_{j-1} and $P^*_{t^*_j,t}$ visits t^*_{j-1}. In this case, we observe that:

$$P^*_{s,s^*_{j-1}}|_{m_{s,j-1}} \subset P^*_{s,s^*_j}|_{m_{s,j}},$$

which implies that:

$$P^*_{s,s^*_{j-1}}|_{m_{s,j-1}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}} \subset P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_j,t}|_{m_{t,j}},$$

and it follows that:

$$F(P^*_{s,s^*_{j-1}}|_{m_{s,j-1}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}) \leq F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_j,t}|_{m_{t,j}}).$$

We now consider that the path $P^*_{s,s^*_j}|_{m_{s,j}}$ does not visit the node s^*_{j-1}. Due to the order in which Algorithm 1 chooses nodes s^*_{j-1} and s^*_j, it holds:

$$F(P^*_{s,s^*_{j-1}}|_{m_{s,j-1}}) \leq F(P^*_{s,s^*_j}|_{m_{s,j-1}-\{s^*_{j-1}\}}).$$

which implies that:

$$F(P^*_{s,s^*_{j-1}}|_{m_{s,j-1}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}) \leq F(P^*_{s,s^*_j}|_{m_{s,j-1}-\{s^*_{j-1}\}\cup\{s^*_j\}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}).$$

Otherwise, the algorithm would have picked s^*_j before s^*_{j-1}. Since the path $P^*_{s,s^*_j}|_{m_{s,j}}$ does not visit the node s^*_{j-1}, it also holds that

$$P^*_{s,s^*_j}|_{m_{s,j}} = P^*_{s,s^*_j}|_{m_{s,j-1}-\{s^*_{j-1}\}\cup\{s^*_j\}}.$$

Therefore, it holds:

$$\begin{aligned} F(P^*_{s,s^*_{j-1}}|_{m_{s,j-1}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}) &\leq F(P^*_{s,s^*_j}|_{m_{s,j-1}-\{s^*_{j-1}\}\cup\{s^*_j\}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}) \\ &= F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}). \end{aligned}$$

Now, if $P^*_{t^*_j,t}|_{m_{t,j}}$ visits t^*_{j-1} then with the same argument as before we state that

$$P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}} \subset P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_j,t}|_{m_{t,j}}$$

and thus Equation (2) holds since

$$F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}) \leq F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_j,t}|_{m_{t,j}}).$$

On the contrary, if $P^*_{t^*_j,t}|_{m_{t,j}}$ does not visit t^*_{j-1}, we use the previous reasoning to derive that:

$$\begin{aligned} F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_{j-1},t}|_{m_{t,j-1}}) &\leq F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_{j-1},t}|_{m_{t-1,j}-\{t^*_{j-1}\}\cup\{t^*_j\}}) \\ &= F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_j,t}|_{m_{t,j}}). \end{aligned}$$

In conclusion, Equation (2) holds for all $1 \leq j \leq i$.

Second, we show that:

$$\frac{F(P_{s,t})}{F(P^*_{s,s^*_i}|_{m_s} \cup P^*_{t^*_i,t}|_{m_t})} \geq 1. \tag{3}$$

The optimal path proposed by the algorithm might be fully uncovered either at the very last round of the algorithm or it was uncovered in an earlier round. When the proposed path was uncovered in the last round of the algorithm, Equation (3) holds because in this case the proposed path must visit s^*_i and t^*_i and thus $P_{s,t} = P^*_{s,s^*_i}|_{m_s} \cup P^*_{s^*_i,t^*_i} \cup P^*_{t^*_i,t}|_{m_t}$, where $P^*_{s^*_i,t^*_i}$ is the optimal path from s^*_i to t^*_i. Hence $P^*_{s,s^*_i}|_{m_s} \cup P^*_{t^*_i,t}|_{m_t} \subseteq P_{s,t}$ which means that the ratio $F(P^*_{s,s^*_i}|_{m_s} \cup P^*_{t^*_i,t}|_{m_t}) \leq F(P_{s,t})$, i.e, the ratio of Equation (3) is strictly larger than one.

In the case when the proposed path was uncovered in a round earlier than the last round of the algorithm, Equation (3) holds because in the round $i-1$, the algorithm computes $\frac{F(P_{s,t})}{F(P^*_{s,s^*_i}|_{m_s} \cup P^*_{t,t^*_i}|_{m_t})}$ and the result has to be larger than 1 since the algorithm does not stop. Therefore it holds:

$$\frac{F(P_{s,t})}{F(P^*_{s,s^*_i}|_{m_s} \cup P^*_{t^*_i,t}|_{m_t})} > 1.$$

Now, using Equations (2) and (3), we are able to show that, for all $1 \leq j \leq i$,

$$\frac{F(P_{s,t})}{F(P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_j,t}|_{m_{t,j}})} > 1.$$

Therefore, for any $1 \leq j \leq i$ and any path that intersects $P^*_{s,s^*_j}|_{m_{s,j}}$ and $P^*_{t^*_j,t}|_{m_{t,j}}$, any 1-certificate needs at least one edge not in $P^*_{s,s^*_j}|_{m_{s,j}} \cup P^*_{t^*_j,t}|_{m_{t,j}}$ to show that the proposed path is the optimal path. Now, for any pair of nodes s^*_j, t^*_j, consider the paths of the form $P^*_{s,s^*_j}|_{m_{s,j}} \cup P_{s^*_j,t^*_j}|_{V/\{m_{s,j} \cup m_{t,j}\}} \cup P^*_{t^*_j,t}|_{m_{t,j}}$, where $P_{s^*_j,t^*_j}|_{V/\{m_{s,j} \cup m_{t,j}\}}$ denotes a path between s^*_j and t^*_j in $V/\{m_{s,j} \cup m_{t,j}\}$. There exists at least $n - 2j + 1$ disjoint paths of the form previously described, one per each node in $V/\{m_{s,j} \cup m_{t,j}\}$ plus the path that connects directly s^*_j and t^*_j. Hence, at least $n - 2j + 1$ edges need to be present in any 1-certificate for each pair of nodes s^*_j, t^*_j.

Therefore, if we sum up all these edges, we obtain that any 1-certificate needs to contain at least the following number of edges:

$$\sum_{j=1}^{i} n - 2j + 1 = in - i(i+1) + i = i(n-i).$$

□

B Proof of Results in Section 5

B.1 Proof of Lemma 7

Proof of Lemma 6. In this proof we use a similar notation than in the proof of Lemma 6, that is, we say that in round j Algorithm 1 uncovers the edge (s^*_j, t^*_j) and all the edges (s^*_j, u) and (u, t^*_j) for all u that do not belong to $m_{s,j} \cup m_{t,j}$.

The proof proceeds by an induction in the steps. Therefore, we define a common notion of step for both algorithms. We consider that, at step j, algorithm $\mathcal{A}^{Gre}_{hom}$ does $n - 2j + 1$ rounds where, in each round, it computes the optimal path with the current information and uncovers all the edges of this path. A step for $\mathcal{A}^{OPD}$ is the set of instructions from line 2 to line 16 of Algorithm 1.

We first show that, at the end of each step, the set of edges that both algorithms uncover is the same. To do that, we show that, at each step j, both algorithms uncover the same set of edges by induction on the number of steps.

We now check that in the first step both algorithms uncover the same set of edges. Since all the edges are initially estimated to $a \geq 0$, the algorithm $\mathcal{A}_{hom}^{Gre}$ finds that the optimal path is the direct path (s,t). In the next $2(n-1)$ rounds, $\mathcal{A}_{hom}^{Gre}$ finds that the optimal paths are all the two-hop paths. Hence, algorithm $\mathcal{A}_{hom}^{Gre}$ uncovers the edge (s,t) and the edges (s,u) and (u,t) for all $u \in V/\{s,t\}$. On the other hand, Algorithm 1 also uncovers the mentioned set of edges in its first step.

We then suppose that until step j both algorithms have uncovered the same set of edges and they have not stopped yet. We aim to prove that the set of uncovered edges after step j also coincide. At step j algorithm $\mathcal{A}_{hom}^{Gre}$ does $n-2j+1$ rounds and in each round it computes the optimal path with the known information and uncovers all the edges of this path. We know that the optimal paths computed by $\mathcal{A}_{hom}^{Gre}$ at step j must contain unknown edges since the algorithm has not stopped. Furthermore, at step j, the only edges that are unknown are those that connect nodes that do not belong to $m_{s,j-1} \cup m_{t,j-1}$. Hence, we have that

$$s_j^* = \underset{v \in V/\{m_{s,j-1} \cup m_{t,j-1}\}}{\operatorname{argmin}} F(P_{s,v}),$$

and

$$t_j^* = \underset{v \in V/\{m_{s,j-1} \cup m_{t,j-1}\}}{\operatorname{argmin}} F(P_{v,t})$$

and the unknown edges are estimated to a which is the minimum possible value. This means that the optimal paths computed by $\mathcal{A}_{hom}^{Gre}$ at step j are of the form $PATH_{s,s_j^*} \cup P_{s_j^*,t_j^*} \cup PATH_{t_j^*,t}$, where $P_{s_j^*,t_j^*}$ is either (s_j^*, t_j^*) or a path from s^* to t^* that traverses only one node that does not belong to $m_{s,j} \cup m_{t,j}$. Furthermore, in these paths the unique unknown edges are (s_j^*, t_j^*) and the edges (s_j^*, u) and (u, t_j^*) where u does not belong to $m_{s,j} \cup m_{t,j}$, which are the edges that uncovers Algorithm 1 at step j. Therefore, $\mathcal{A}_{hom}^{Gre}$ uncovers the same set of edges than Algorithm 1 at the end of step j. Thus, if both algorithms provide an $\alpha-$approximation at the end of a step, then we have shown that both algorithms uncover the same set of edges.

We now focus on the case where, at least, one algorithm finds an α-approximation before an step is finished. In the first round of a step, the optimal path that algorithm $\mathcal{A}_{hom}^{Gre}$ finds is $P_{s,s^*}^* \cup (s^*, t^*) \cup P_{t^*,t}^*$ and thus it queries the edge (s^*, t^*), which is the edge that Algorithm 1 queries. In the following rounds of this step, we observe that if θ_{Gre} and θ_{OPD} provide the same of ordering of paths selection, then both algorithms at each round will uncover the same edges and thus they will stop in the same round. On the other hand, if these policies do not provide the same order of paths selection,

then an algorithm can stop in the second round of this step and the other can stop in the last round of the same step. The algorithm that stops in the second round uncovers $1 + 2$ edges in that step and the algorithm that stops in the last round uncovers $1 + 2(n - i)$ edges. Thus, the difference in the number of uncovered edges by both algorithms is $2(2 - i - 1)$. Finally, we conclude that the larger value of this difference is given in the fist step, which is $2(n - 2)$. □