Shortest Path Discovery Problem, Revisited (Query Ratio, Upper and Lower Bounds)

Olivier Brun, Josu Doncel, Christopher Thraves-Caro

Shortest Path Discovery Problem, Revisited (Query Ratio, Upper and Lower Bounds)

Olivier Brun[1,3], Josu Doncel[1,2] and Christopher Thraves Caro[1,3]

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ. de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[3] Univ. de Toulouse, LAAS, F-31400 Toulouse, France

Abstract. Consider the following problem: given two nodes s and t in a fully connected graph where each edge of the graph has a hidden length, discover the shortest path between s and t. The issue is that edge lengths are hidden. Hence, any algorithm that aims to solve the problem needs to uncover the length of some edges to discover the shortest path. The goal then is to discover the shortest path by means of uncovering the least possible amount of edge lengths. This problem is known as the Shortest Path Discovery (SPD) problem. The SPD problem is introduced by [13] in AAAI 2004. In this document, we study the SPD problem. We show that the previously proposed measure to evaluate the quality of an algorithm does not differentiate correctly algorithms according to their performance. Hence, it does not provide insightful information. Therefore, we introduce the *query ratio*, the ratio between the number of uncovered edge lengths and the least number of edge lengths required to solve the problem, as a new measure to evaluate algorithms that solve the SPD problem. When an input of size n is considered, we prove a lower bound of $1 + 4/n - 8/n^2$ on the query ratio of any algorithm that solves the SPD problem. As well, we present an algorithm that solves the problem and we prove that its query ratio is equal to $2 - 1/(n-1)$.

1 Introduction

In a given graph, searching for the shortest path from an origin node to a destination node is a widely studied problem, see for instance the following surveys [14, 3, 6, 9]. Finding a shortest path is a problem that comes into sight in the area of Artificial Intelligence as subproblem of different problems, for example in symbolic planning, parsing, vision, or speech recognition.

The work [13] presented in AAAI 2004 introduces the Shortest Path Discovery (SPD) problem. In the SPD problem the length of the edges are unknown, but they can be queried. The SPD Problem aims to find the shortest path between two nodes querying the minimum number of edges. The SPD problem arises in a context in which obtaining the length of the edges dominates the total cost of computing a shortest path. As explained in [13], such a context appears in character or speech recognition, systems based on segmentation lattices, hierarchical planning, or exploration of unsafe environments.

Our particular motivation emerges from a routing problem in communication networks. Consider a set of nodes located at various spots in the Internet, and imagine that a source node wants to deliver a message to a destination node with a latency as low as possible. It may happen that the direct Internet path between the source and destination nodes has an unacceptably high latency. In that case, provided that other nodes can act as relays for the message, it may be worth searching for an alternate path passing through one or more intermediate nodes. One can be even more ambitious and search for the minimum latency path in the complete graph formed by all nodes. However, monitoring the quality of the Internet paths between all pairs of nodes by sending probe packets can be excessively costly, and it makes sense to minimize the monitoring effort required to discover the shortest path. This scenario perfectly falls whithin the scope of the SPD problem.

In this paper, we study the SPD problem. Due to our particular motivation, we restrict ourselves to the case of fully connected graphs. On the other hand, we extend the definition of the SPD problem by relaxing the constraint that a shortest path has to be discovered, and allowing a feasible path to be at most α times longer than the sortest one for some $\alpha \geq 1$. Of course, we fall back on the original problem when $\alpha = 1$. Even if the restriction added to the problem appears to simplify it, all together with the relaxation they form an appealing version of the SPD problem.

Our contributions are as follows. We first prove that, for any complete graph of size n, any algorithm will need to query at least $n-1$ edges to find a feasible solution. On the other hand, we also show that for any algorithm there exists a bad instance such that the number of edges queried by the algorithm will be of the same order of magnitude than the total number of edges. The latter results suggest that the number of queries is not an appropriate measure to discriminate between algorithms for the SPD problem. We thus propose a new measure, the query ratio, to evaluate the performance of algorithms that solve SPD problem. The query ratio of an algorithm is defined as the worst-case ratio (over all instances) of the number of queries made by an algorithm on an instance to the minimum number of queries required to find a feasible path for that instance. For instances of size n, we prove that any algorithm has a query ratio of at least $1+\frac{4}{n}-\frac{8}{n^2}$ and propose an approximation algorithm whose query ratio is upper bounded by $2-\frac{1}{n-1}$.

The paper is organized as follows. We first introduce some definitions and present related works. We then prove our lower bounds on the number of queries, before establishing lower and upper bounds on the query ratio. Finally, some conclusions are drawn and future research directions are proposed at the end of the paper.

2 Definitions

Let $G = (V, E)$ be a fully connected undirected graph with n nodes where each edge has an unknown positive length. Let us denote by $\ell(e) > 0$ the *unknown*

length of the edge $e \in E$. We define a *path* in G as a finite ordered set of nodes in which all nodes are distinct. A path that connects nodes u and v is a path whose first node is u and last node is v. An edge belongs to a path *if and only if* the edge is formed by two consecutive nodes in the path. We denote by $P_{(u,v)}$ a path that connects nodes u and v, and by $\mathcal{P}_{(u,v)}$ the set of all such paths. The *length* of a path is measured as the sum of the lengths of the edges in the path, i.e., the length of a path $P_{(u,v)}$ is $\ell(P_{(u,v)}) := \sum_{e \in P_{(u,v)}} \ell(e)$. We extend the previous notation to any set of edges $F \subseteq E$ by letting $\ell(F) := \sum_{e \in F} \ell(e)$.

A *shortest* path between nodes u and v is a path $P^*_{(u,v)}$ in $\mathcal{P}_{(u,v)}$ so that $\ell(P^*_{(u,v)}) \leq \ell(P_{(u,v)})$ for all $P_{(u,v)} \in \mathcal{P}_{(u,v)}$. We denote by $\delta_{(u,v)}$ the length of a shortest path in G between nodes u and v.

On the other hand, we follow the standard definitions used in graph theory. A *cut* is a partition of the set V of nodes into two disjoint subsets. The *cut-set* of a cut is the set of edges whose end points are in different subsets of the partition.

In order to discover the shortest path between two nodes, an algorithm has to uncover the unknown length of the edges. We use the abstraction of an oracle for that purpose. Let $\mathcal{O}$ be an oracle that can be accessed by an algorithm to *uncover* the length of an edge or a set of edges, i. e., the oracle is accessed by an algorithm by requesting the length of an edge or a set of edges. Hence, the oracle reveals to the algorithm the length of all the requested edges. For short, we say that an algorithm *uncovers an edge* when we refer to all this process.

We remark that in the original definition of the SPD problem, it seeks the shortest path between two nodes in a graph uncovering the minimum possible amount of edges. Nonetheless, sometimes the main ambition can be lowered down and, instead to seek the shortest path, it might be enough to find a path with a guarantee with respect to the shortest path. This is the case that we study.

Consider a path $P_{(s,t)}$ between two nodes s and $t \in V$. We say that $P_{(s,t)}$ is an α-*approximation* if the following condition holds:

$$\frac{\ell(P_{(s,t)})}{\delta_{(s,t)}} \leq \alpha.$$

When $\delta_{(s,t)}$ is totally unknown (no lower bound is known except for the fact that $\delta_{(s,t)}$ is positive), there is no guarantee on the length of path $P_{(s,t)}$ with respect to that of a shortest path, and we say that $P_{(s,t)}$ is an ∞-*approximation.*

We say that a set of edges $\mathcal{C} \subseteq E$ is an α-*certificate* of G *if and only if*

- the length of the edges in $\mathcal{C}$ is known, whereas edges in $E \setminus \mathcal{C}$ have unknown lengths,
- $\mathcal{C}$ contains a path from s to t, 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. Hence, its length is fully known. In all instances there exists at least one α-certificate for any α. Indeed, the set E that contains all the edges of the graph

is an α-certificate for any α. In [13], the authors prove that a set of edges is a 1-certificate (i. e., a certificate enough to guarantee that the proposed path is a shortest path) if the proposed path is a shortest path in the graph considering the known lengths of the edges in the certificate and lengths equal to zero for all other edges. Note that such a result can be extended as follows. A set of edges $\mathcal{C}$ is an α-certificate *if and only if* it contains a proposed path from s to t and the proposed path is an α-approximation in the graph where the length of the edges in $\mathcal{C}$ is known and the length of the edges in $E \setminus \mathcal{C}$ is zero.

Instance and Feasible Solution An *instance* I of the SPD problem is defined by a complete undirected graph $G = (V, E)$ with n nodes and unknown positive edge lengths, two special nodes s and $t \in V$, a targeted approximation factor $\alpha \geq 1$, and an oracle $\mathcal{O}$ that can be accessed by an algorithm to *uncover* a set of edges. We define the *size* of an instance I as the number of nodes of the fully connected graph G, and it is denoted by $|I|$. As well, we define a *feasible solution* for the SPD problem as an α-certificate whose lengths are fully uncovered.

Two Objective Functions Let us denote by $U(\mathcal{A}(I))$ the set of edges whose length has been uncovered by an algorithm $\mathcal{A}$ when solving the SPD problem on the instance I. In other words, the set $U(\mathcal{A}(I))$ is the α-certificate given by algorithm $\mathcal{A}$ as a solution for the instance I. We define the *number of queries* of an algorithm that solves the SPD problem as follows.

Definition 1. *The* number of queries *for instances of size n of an algorithm that solves the SPD problem is β_n if and only if $|U(\mathcal{A}(I))| \leq \beta_n \; \forall \; I$ such that $|I| = n$.*

On the other hand, we point out the fact that there might be many α-certificates in an instance and an algorithm has to search for one. We are interested in the size of a smallest α-certificate of an instance. Let us denote by $|\mathcal{C}^{\alpha}_{\min}(I)|$ the size of a smallest α-certificate of instance I. Henceforth, we define the *query ratio* of an algorithm as follows.

Definition 2. *The* query ratio *of an algorithm $\mathcal{A}$ that proposes an α-approximation as a solution to solve the SPD problem is defined by the following ratio:*

$$\max_{I} \frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|}.$$

Our Contributions Our main contributions are as follows:

- We present a new meaningful measure, the query ratio, to evaluate the performance of algorithms that solve SPD problem.
- We argue why the number of queries (as measure for the SPD problem) does not provide meaningful information about the algorithms that solve the SPD problem.
- We prove that any algorithm has a query ratio of at least $1 + \frac{4}{n} - \frac{8}{n^2}$, which means that no algorithm solves the SPD problem optimally.

– We present an algorithm that solves the SPD problem for any $\alpha \geq 1$. Furthermore, we prove that such an algorithm has a query ratio of $2 - \frac{1}{n-1}$ when α is equal to 1.

3 Related Work

The interest of researchers in discovering the shortest path in graphs with unknown information have increased in the last years. The survey [12] presents a compilation of results in the context of Shortest-Path Queries in Static Networks. The main difference of these problems with the SPD problem is that Shortest-Path Queries in Static Networks includes a preprocessing algorithm that may compute some information of the network to then be able to answer shortest-path or distance queries. The goal is to construct an efficient data structure via the preprocessing algorithm so that queries are answered rapidly. While, in the SPD problem, operations to obtain information about the network dominates the total cost of computation. In the context of bioinformatics, the survey [2] presents a collection of results and methods in the area of combinatorial search, focusing on graph reconstruction using queries of different type. The goal is to reconstruct a hidden graph from a class of graphs making as few queries as possible. Again, the goal of such problem differs from the goal of the SPD problem, since the SPD problem aims to find a shortest path.

The authors of [10] study shortest path problems when the graph is unknown in advance, but specified dynamically. These problems seek dynamic decision rules so that the total traversed distance has the best possible ratio to the shortest path. The authors describe optimal decision rules for two cases: layered graphs of width two, and two-optimal scenes with unit square obstacle. The objective function in this case clearly differs form the objective function of the SPD problem.

Another related work is [1] where the authors present the use of agents to discover the edges of the graph or a shortest path from a source to a sink node. They give bounds on the number of agents required to discover directed and undirected graphs. The objective function of this problem is the number of agents used to discover the solution, which differs from the objective function of the SPD problem.

In this document, we present an algorithm that searches from the source and the sink node at the same time. This type of algorithms are known in the literature as bidirectional search algorithms and their study has a long history [11]. These algorithms are shown to be faster than standard search algorithms such as Dijkstra's algorithm or A* [8] in most of the instances. However, make the two search fronts meet in the middle is a difficult task. Several algorithms have been presented in the literature to solve this issue. For example, in [5] the authors suggest the BHFFA, which is a front-to-front algorithm that it is computationally expensive. The authors of [7, 4] present improved versions of the BHFFA that are computationally less expensive.

4 Lower Bounds on the Number of Queries β_n

In this section, we present lower bounds for the number of queries required by any algorithm providing a finite α approximation.

Lemma 1. *For any finite $\alpha \geq 1$, all α-certificate 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 length $\ell(P_{(s,t)})$ of the proposed path $P_{(s,t)}$ is fully determined. Second, we remark that, in order to provide any finite approximation guarantee, the α-certificate needs to provide a strictly positive lower bound for the length of a shortest path between s and t. Otherwise, the only known lower bound for the length of a shortest path between s and t is 0.

Now, consider an instance I of SPD problem and an α-certificate $\mathcal{C}$. Let us assume that the α-certificate does not contain a cut-set that separates s and t. Hence, there exists a path $P^*_{(s,t)}$ between s and t so that $P^*_{(s,t)} \cap \mathcal{C} = \emptyset$. Since only edges in $\mathcal{C}$ have a known length, the length of $P^*_{(s,t)}$ is totally unknown for the α-certificate $\mathcal{C}$, implying that the only lower bound on $\delta_{s,t}$ known by $\mathcal{C}$ is 0. Thus, the α-certificate $\mathcal{C}$ cannot guarantee any finite approximation for its proposed solution. Therefore, there is a contradiction because $\mathcal{C}$ is not an α-certificate. □

Lemma 2. *A 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, 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 $P_{(s,t)} \in \mathcal{P}_{(s,t)}$. Since $0 < \min\{\ell(P_{(s,t)} \cap \mathcal{C}) : P_{(s,t)} \in \mathcal{P}_{(s,t)}\} \leq \delta_{(s,t)}$, it holds that

$$\ell(P^{\mathcal{C}}_{(s,t)}) \leq \frac{\ell(P^{\mathcal{C}}_{(s,t)})}{\min\{\ell(P_{(s,t)} \cap \mathcal{C}) : P_{(s,t)} \in \mathcal{P}_{(s,t)}\}}\, \delta_{(s,t)} < \infty,$$

implying that $\mathcal{C}$ is an $\alpha-$certificate for

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

□

Lemma 1 enables us to present lower bounds for 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 SPD 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 real $\alpha \geq 1$ and integer $1 \leq p \leq n/2$, there exists an input 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 input that certifies the conditions stated in the lemma. Let us split the set of nodes in one set of size p and one set of size $n-p$. Each edge with its two end points in the same set has length $\epsilon > 0$. The direct edge (s,t) has length 1. Each edge with its end points in different sets (except for the direct edge) has length $\alpha \geq 1$. In such a graph, the only α-approximation is indeed the direct path s,t. But, in order to guarantee such condition, any algorithm needs to uncover at least the cut-set of size $p \cdot (n-p)$ that contains all the edges of length α. □

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

According to Corollary 2, we consider meaningless the number of queries β_n of an algorithm. Indeed, for any algorithm and whatever the value of $\alpha \geq 1$, 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 proposes α-approximations to solve the SPD problem. We believe that the query ratio is a fair measure to evaluate the performance of algorithms for the SPD problem since it expresses how far is the number of queries asked by an algorithm with respect to the best possible any algorithm can perform in that instance.

5 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 an adversary constructed for any algorithm. For the second bound, we present an algorithm and analyze its query ratio.

5.1 An Adversary for Any Algorithm

Lemma 4. *For any $\alpha \geq 1$ and for any algorithm $\mathcal{A}$ that proposes an α-approximation, there exists a particular instance I of size n such that*

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

Algorithm 1 Adversary that construct a bad instance for any algorithm, 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 $\ell(e_i) = 1$.
  end if
  if $e_i = (s,u)$ such that $g(u) = 0$ **then**
    REPLAY $\ell(e_i) = \epsilon$ 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 $\ell(e_i) = \alpha$.
  end if
  if $e_i = (u,t)$ such that $g(u) = 0$ **then**
    REPLAY $\ell(e_i) = \epsilon$ 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 $\ell(e_i) = \alpha$.
  end if
  if $e_i = (u,v)$ such that $g(u) = g(v) = 0$ **then**
    REPLAY $\ell(e_i) = \epsilon$.
  end if
  if $e_i = (u,v)$ such that $g(u) = 0$ **and** $g(v) \in \{s,t\}$ **then**
    REPLAY $\ell(e_i) = \epsilon$.
    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 length equal to ϵ.
    UPDATE $G_{g(v)} := G_{g(v)} \cup \{u\} \cup \{u' : u' \to_\epsilon u\}$, where $u' \to_\epsilon u$ means that u and u' are connected by a path of uncovered edges each with length 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 $\ell(e_i) = \alpha$.
  end if
  if $e_i = (u,v)$ such that $g(u) \in \{s,t\}$ **and** $g(v) = g(u)$ **then**
    REPLAY $\ell(e_i) = \epsilon$.
  end if
end for

Proof. 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 length 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 1.

The adversary constructs an instance similar to the instance of Lemma 3, but ad hoc to each algorithm. In the instance constructed by the adversary, the shortest path between s and t is always the direct edge (s,t). Furthermore, such a path is the only α-approximation. On the other hand, 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 partition and the size of each sets of the partition depend on the algorithm. Edges that connect two nodes of the same set of the partition have length ϵ. While, edges that connect two nodes each in a different set of the partition have length α.

Each algorithm can be seen as a sequence of edges to be uncovered. An algorithm poses a query to the oracle that is an edge to be uncovered. The oracle answers each query providing the length 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 1). If the algorithm queries the edge (s,t) to be uncovered, the adversary answers to the algorithm $\ell(s,t) = 1$ (see lines 3 and 4 in Algorithm 1). If the algorithm queries an edge of the second group to be uncovered and the node u has not been placed in any set of the partition, the adversary answers $\ell(s,u) = \epsilon$ and places u in the set G_s (see lines 6 and 7 in Algorithm 1). Otherwise, when u has been placed in a set of the partition (it can be only the set G_t), the adversary answers $\ell(s,u) = \alpha$ (see lines 9 and 10 in Algorithm 1). The adversary acts equivalently when the algorithm queries an edge of the third group. I. e, if u has not been placed in any set of the partition, the adversary answers $\ell(u,t) = \epsilon$ and places u in the set G_t (see lines 12 and 13 in Algorithm 1). Otherwise, when u has been placed in a set of the partition (it can be only the set G_s), the adversary answers $\ell(u,t) = \alpha$ (see lines 15 and 16 in Algorithm 1). Finally, if the algorithm queries an edge of the fourth group, there exist four different cases: u and v have not been placed in any set of the partition, then the adversary answers $\ell(u,v) = \epsilon$ and none is placed in any set of the partition (see lines 18 and 19 in Algorithm 1). When one of them has been placed in a set of the partition, says u, and the other one has not, the adversary answers $\ell(u,v) = \epsilon$ 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 1). Finally, if the two nodes have been placed in a set of the partition, the adversary answers $\ell(u,v) = \epsilon$ if the two nodes belong to the same set of the partition, or $\ell(u,v) = \alpha$ if they belong to different sets (see lines 27 to 31 in Algorithm 1).

We observe that the shortest path from s to t in any instance constructed by the adversary, regardless the algorithm, is the edge (s,t). Moreover, it is the only α-approximation. On the other hand, for any algorithm, we obtain two sets G_s

and G_t, according to the notation defined in Algorithm 1, 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, therefore, 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 direct consequence of Lemma 4 is the following Theorem.

Theorem 1. *For any $\alpha \geq 1$ and for any algorithm $\mathcal{A}$ that proposes an α-approximation, it holds the following inequality for the query ratio,*

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

According to this result, we observe that the query ratio of any algorithm is strictly higher that 1, which means that no algorithm solves the SPD problem optimally.

5.2 An Algorithm that Searches From the Source and the Sink

In this subsection, we present an algorithm that proposes an α-approximation as a solution of the SPD problem. We remark that in the proposed algorithm α is a parameter, 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

Algorithm 2 Algorithm for SPD problem

1: INITIALIZE sets $m_s = \{\emptyset\}$, $m_t = \{\emptyset\}$;
functions $\ell_{\inf}(e) = 0$, and $\ell_{\sup}(e) = \infty \; \forall \; e \in E$;
functions $s(v) = \infty \; \forall v \in V/\{s\}$, and $s(s) = 0$;
functions $t(v) = \infty \; \forall v \in V/\{t\}$, and $t(t) = 0$;
paths $PATH_{\inf} = \emptyset$, and $PATH_{\sup} = \emptyset$;
paths $PATH_{(s,u)} = \emptyset$, and $PATH_{(u,t)} = \emptyset \; \forall u \in V/\{s,t\}$;
variable $approx = \infty$.
2: **while** $approx > \alpha$ **do**
3: COMPUTE $s^* := \text{argmin}_{v \in V/\{m_s \cup m_t\}} \, s(v)$.
4: COMPUTE $t^* := \text{argmin}_{v \in V/\{m_s \cup m_t\}} \, t(v)$.
5: UPDATE $m_s := m_s \cup s^*$.
6: UPDATE $m_t := m_t \cup t^*$.
7: QUERY $\{(s^*, t^*)\}$.
8: QUERY $\{(s^*, u) : \forall u \in V/\{m_s \cup m_t\}\}$.
9: QUERY $\{(u, t^*) : \forall v \in V/\{m_s \cup m_t\}\}$.
10: UPDATE $\ell_{\inf}(e) := \ell(e)$ for all queried e.
11: UPDATE $\ell_{\sup}(e) := \ell(e)$ for all queried e.
12: COMPUTE the shortest st-path according to $\ell_{\inf}(\cdot)$.
13: UPDATE $PATH_{\inf}$ to the shortest st-path according to $\ell_{\inf}(\cdot)$.
14: COMPUTE the shortest st-path according to $\ell_{\sup}(\cdot)$.
15: UPDATE $PATH_{\sup}$ to the shortest st-path according to $\ell_{\sup}(\cdot)$.
16: COMPUTE the shortest su-path according to $\ell_{\sup}(\cdot) \; \forall \; u \in V/\{m_s \cup m_t\}$.
17: UPDATE $PATH_{(s,u)}$ to the the shortest su-path according to $\ell_{\sup}(\cdot) \; \forall \; u \in V/\{m_s \cup m_t\}$.
18: COMPUTE the shortest ut-path according to $\ell_{\sup}(\cdot) \; \forall \; u \in V/\{m_s \cup m_t\}$.
19: UPDATE $PATH_{(t,u)}$ to the shortest ut-path according to $\ell_{\sup}(\cdot) \; \forall \; u \in V/\{m_s \cup m_t\}$.
20: UPDATE $s(u) := \ell(PATH_{(s,u)})$ for all $u \in V/\{m_s \cup m_t\}\}$.
21: UPDATE $t(u) := \ell(PATH_{(t,u)})$ for all $u \in V/\{m_s \cup m_t\}\}$.
22: UPDATE $approx := \frac{\ell(PATH_{\sup})}{\ell(PATH_{\inf})}$.
23: **end while**
24: RETURN $P_{(s,t)} := PATH_{\sup}$.

proving that it never queries more than two times the size of the smallest certificate of an instance, when it proposes an optimal solution. A precise description of the algorithm is presented in Algorithm 2.

The algorithm defines lower and upper bounds on the length of each edge that are initially set to zero and infinity, respectively (see line 1 in Algorithm 2). 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. At each round, Algorithm 2 picks the closest nodes to s and t according to the current uncovered edges. The closest node to s (resp. t) is denoted by s^* (resp. t^*) (see lines 3 and 4 in Algorithm 2). Then, the algorithm queries all the edges of the form (s^*, u) and (u, t^*) that have not been queried yet, and that approaches in one edge s^* to t and t^* to s, respectively (see lines 7 to 9 in Algorithm 2). The algorithm

then computes the shortest path between s and t using the real length for the uncovered edges and the lower bound set at the beginning for the edges not yet uncovered ($PATH_{\text{inf}}$). The algorithm also computes the shortest path between s and t using the real length for the uncovered edges and the upper bound set at the beginning for the edges not yet uncovered ($PATH_{\text{sup}}$). Finally, the algorithm computes whether $PATH_{\text{sup}}$ is an α-approximation in line 22. If that is the case, Algorithm 2 stops and returns $PATH_{\text{sup}}$, 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 $\frac{\ell(PATH_{\text{sup}})}{\ell(PATH_{\text{inf}})} \leq \alpha$ and $\ell(PATH_{\text{inf}})$ is a lower bound on $\delta_{s,t}$.

In order to proceed with the analysis of the query ratio of Algorithm 2, we first compute the number of queries performed by the algorithm up to a generic round i.

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

Proof. At each round, the algorithm queries $2|V/\{m_s \cup m_t\}|+1$ edges according to lines 7 to 9 of Algorithm 2. At each round, the sizes of m_s and m_t increases in one element. 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 shortest path from s to s^* and the shortest path from t^* to t would have been the shortest 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 2 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 2 stops after i rounds in an instance I, the size of the minimum 1-certificate of the instance is at least $i(n-i)$, i.e., $|\mathcal{C}^1_{\min}(I)| \geq i(n-i)$.*

Proof. 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,

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

where $m_{s,0} = m_{t,0} = \{\emptyset\}$. We use $m_s = m_{s,i}$ and $m_t = m_{t,i}$ to denote the sets m_s and m_t after Algorithm 2 has finished. Moreover, for any two nodes u, v and

a set of nodes $U \subseteq V$, we denote by $P^*_{(u,v)}|_U$ the shortest path between u and v in the graph induced by the set of nodes U.

First we show that the following two inequalities hold for all $1 \leq j \leq i$.

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

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

Since the arguments are equivalent for both equations, we only argue that Equation (1) holds.

We point out the fact that the path $P^*_{(s,s^*_j)}|_{m_{s,j}}$ either visits s^*_{j-1} or it does not visit such node. In the first case, Equation (1) holds because the path $P^*_{(s,s^*_j)}|_{m_{s,j}}$ can be decomposed in the path $P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}}$ (the shortest path from s to s^*_{j-1} in $m_{s,j-1}$) union some other edges. Hence, the length $\ell(P^*_{(s,s^*_j)}|_{m_{s,j}})$ can also be decomposed in the sum of $\ell(P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}})$ plus a positive value.

In the second case, when the path $P^*_{(s,s^*_j)}|_{m_{s,j}}$ does not visit the node s^*_{j-1}, due to the order in which Algorithm 2 chooses nodes s^*_{j-1} and s^*_j, it holds:

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

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:

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

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

Second, we show that the following equation holds.

$$\frac{\ell(P_{(s,t)})}{\ell(P^*_{(s,s^*_i)}|_{m_s}) + \ell(P^*_{(t,t^*_i)}|_{m_t})} > 1. \tag{3}$$

The shortest 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. Therefore the length of the proposed path $P_{(s,t)}$ is of the form $\ell(P^*_{(s,s^*_i)}|_{m_s}) + \ell(P^*_{(t,t^*_i)}|_{m_t}) + A$, where A is a positive value given by the length of the last uncovered edges added to the shortest path proposed by the algorithm. Hence, it holds:

$$\frac{\ell(P_{(s,t)})}{\ell(P^*_{(s,s^*_i)}|_{m_s}) + \ell(P^*_{(t,t^*_i)}|_{m_t})} = \frac{\ell(P^*_{(s,s^*_i)}|_{m_s}) + \ell(P^*_{(t,t^*_i)}|_{m_t}) + A}{\ell(P^*_{(s,s^*_i)}|_{m_s}) + \ell(P^*_{(t,t^*_i)}|_{m_t})} > 1.$$

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 $\frac{\ell(P_{(s,t)})}{\ell(P^*_{(s,s^*_i)}|_{m_s})+\ell(P^*_{(t,t^*_i)}|_{m_t})}$ and the result has to be larger than 1 since the algorithm does not stop. Therefore it holds:

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

Now, using Equations (1) to (3), we are able to show that, for all $1 \leq j \leq i$, it holds the following:

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

Therefore, for any $1 \leq j \leq i$ and any path that intersects $P^*_{(s,s^*_j)}|_{m_{s,j}}$ and $P^*_{(t,t^*_j)}|_{m_{t,j}}$, any 1-certificate needs at least one edge not in $P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t,t^*_j)}|_{m_{t,j}}$ to show that the proposed path is the shortest path. Now, for any pair of nodes s^*_j, t^*_j, consider the paths of the form $P^*_{(s,s^*_j)}|_{m_{s,j}} \cap P(s^*_j, t^*_j)|_{V/\{m_{s,j} \cup m_{t,j}\}} \cap P^*_{(t,t^*_j)}|_{m_{t,j}}$, 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 needs 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 amount of edges.

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

□

Now, we state the main result of this Subsection in the following theorem.

Theorem 2. *Let $\mathcal{A}^*$ denote Algorithm 2. Therefore, for the query ratio of $\mathcal{A}^*$, it holds:*

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

Proof. From Lemmas 5 and 6, it holds:

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

□

6 Conclusions and Future Work

In this document we introduced the query ratio, a new measure to analyze algorithms that solve the SPD problem. In our opinion, the query ratio of an algorithm provides an important insight into the real quality of an algorithm solving the SPD 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. Therefore, we consider that the query ratio is the correct measure to take into account in the design of algorithms to solve the SPD problem.

In this document we presented lower and upper bounds on the query ratio. Under our consideration, the most appealing problem that this document leaves open is the gap between these 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 general graphs (not only when the graph in consideration is fully connected).

References

1. Noga Alon, Yuval Emek, Michal Feldman, and Moshe Tennenholtz. Economical graph discovery. In *ICS*, pages 476–486, 2011.
2. Mathilde Bouvel, Vladimir Grebinski, and Gregory Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In *WG 2005. LNCS*, pages 16–27. Springer, 2005.
3. Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, pages 516–525, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.
4. Henry W. Davis, Randy B. Pollack, and Thomas Sudkamp. Towards a better understanding of bidirectioanl search. In *AAAI*, 1984.
5. Dennis de Champeaux. Bidirectional heuristic search again. *J. ACM*, 30(1):22–32, January 1983.
6. Stuart E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, May 1969.
7. Subrata Ghosh and Ambuj Mahanti. Bidirectional heuristic search with limited resources. *Information Processing Letters*, 40(6):335 – 340, 1991.
8. 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.
9. ALISTAIR Moffat. Empirical survey of shortest path algorithms. *NZ OPER. RES.*, 11:153–164, 1983.
10. Christos H Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.
11. Ira Pohl. *Bi-directional and Heuristics Search in Path Problems.* PhD thesis, Standford University, 1969.

12. Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4), March 2014.
13. Csaba Szepesvári. Shortest path discovery problems: A framework, algorithms and experimental results. In *AAAI*, pages 550–555, 2004.
14. Dorothea Wagner and Thomas Willhalm. Speed-up techniques for shortest-path computations. In *24th International Symposium on Theoretical Aspects of Computer Science (STACS*, pages 23–36. Springer, 2007.