HAL
archives-ouvertes.fr

Optimal Path Discovery Problem with Homogeneous Knowledge

Christopher Thraves-Caro, Josu Doncel, Olivier Brun

▶ **To cite this version:**

Christopher Thraves-Caro, Josu Doncel, Olivier Brun. Optimal Path Discovery Problem with Homogeneous Knowledge. 2015. hal-01149901

HAL Id: hal-01149901
https://hal.archives-ouvertes.fr/hal-01149901

Submitted on 7 May 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Optimal Path Discovery Problem with Homogeneous Knowledge

Christopher Thraves Caro[a,c], Josu Doncel[a,b] and Olivier Brun[a,c]

[a] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[b] Univ. de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[c] Univ. de Toulouse, LAAS, F-31400 Toulouse, France

Abstract

In this work we study the *Optimal Path Discovery* (OPD) problem. That is: given a complete graph $G = (V, E)$, a function $F : 2^E \to [0, \infty)$ that assigns a positive value to each sub set of E, two nodes s and t in V, and a positive hidden value $f(e)$ for each edge $e \in E$, discover a path P^* between s and t that optimizes the value of $F(P^*)$. The issue is that the edge values $f(\cdot)$ are hidden. Hence, any algorithm that aims to solve the problem needs to uncover the value of some edges to discover an optimal path. The goal then is to discover an optimal path by means of uncovering the least possible amount of edge values. This problem is an extension of the well known *Shortest Path Discovery* problem in which $f(e)$ represents the length of $e \in E$, and $F(P)$ computes the length of P.

In this paper, we present a study of the OPD problem in a setting in which homogeneous information with respect to the values $f(e)$ is given beforehand. As a measure to evaluate the performance of an algorithm, we first study the maximum number of queries asked by an algorithm in any instance of size $|V| = n$. 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 edge values and the least number of edge values required to solve the problem, as a new measure to evaluate algorithms that solve the OPD problem. When an input of size $|V| = n$ is considered, we prove a lower bound of $1 + 4/n - 8/n^2$ on the query ratio of any algorithm that solves the OPD problem. As well, we present an algorithm that solves the problem and we prove that its query ratio is equal to $2 - 1/(n-1)$. Finally, we implement different algorithms and evaluate their query ratio experimentally.

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

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 (see, e.g., [1] for an introduction). Nevertheless, in some situations the difficulty lies more in collecting the information on the graph than in computing a shortest path. In such situations, it makes sense to seek to minimize the amount of information required to compute a shortest path. In this paper, we particularly consider situations in which the cost of obtaining the length of the edges dominates the total cost of computing a shortest path. 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, where the cost of a path is a function of the costs of the edges it is comprised of.

We investigate the OPD problem for a broad class of cost functions, including, but not restricted to, additive functions for which the cost of a path is the sum of the costs of its edges. We handle both the cases when the performance metric has to be minimized or maximized. We consider as well the extension obtained by relaxing the constraint that an optimal path has to be discovered and allowing the cost of a feasible path to be at most α times that of an optimal path for some $\alpha \geq 1$. Of course, we fall back on the original problem when $\alpha = 1$. We note that for additive performance metrics and $\alpha = 1$ the OPD problem coincides with the so-called *Shortest Path Discovery* problem introduced in [18]. As detailed in [18], this problem finds applications in speech recognition, systems based on segmentation lattices, hierarchical planning, or exploration of unsafe environments.

Our particular motivation for studying the OPD problem comes from a routing problem in communication networks. 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 or maximization of the probability of successful transmission). 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 an alternate path passing through one or more intermediate nodes. One can be even more ambitious and search for the optimal 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 since the number of such paths grows as n^2. Hence, it makes sense to minimize the monitoring effort required to discover an optimal path. This scenario perfectly falls within the scope of the OPD problem.

In order to keep the discussion as general as possible, we formulate the OPD problem for arbitrary graphs assuming no prior knowledge on edge costs. Nevertheless, due to our particular motivation, we focus the analysis on the case of complete graphs. 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. On the other hand, 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. The latter results suggest that the number of queries is not an appropriate measure to discriminate between algorithms for the OPD problem. We thus propose a new measure, the *query ratio*, to evaluate the performance of algorithms that solve the OPD 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. We prove that any algorithm has a query ratio of at least $1 + \frac{4}{n} - \frac{8}{n^2}$ and propose an algorithm whose query ratio is upper bounded by $2 - \frac{1}{n-1}$.

The paper is organized as follows. In Section 2 we introduce some definitions and present the mathematical formulation of the OPD problem as well as some assumptions used throughout the paper. In Section 3 we put our work in the context of the existing literature. We prove our lower bounds on the number of queries in Section 4, and in Section 5 we establish lower and upper bounds on the query ratio. Section 6 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 7, some conclusions are drawn and future research directions are proposed.

2 Problem Statement

Before formulating the OPD problem in Section 2.2, we first introduce some definitions in Section 2.1. In order to keep the discussion as general as possible, we present a mathematical formulation of the OPD problem with as few as possible assumptions on the routing metrics to be optimized. Nevertheless, our results require certain assumptions on how the cost of a path is defined from the costs of the edges it is comprised of. These assumptions are stated in Section 2.3. Finally, Section 2.4 is devoted to a discussion on the initial knowledge available prior to decision making, regarding both the knowledge on the values of the edges and the knowledge on the structure of the graph.

2.1 Definitions

Let $G = (V, E)$ be a complete undirected graph with n nodes. We define a *path* in G as a finite ordered set of nodes in which all nodes are distinct. A path that connects nodes s and t is a path whose first node is s and last node is t. An edge belongs to a path *if and only if* the edge is formed by two consecutive nodes in the path. We denote by $P_{s,t}$ a path that connects nodes s and t, and by $\mathcal{P}_{s,t}$ the set of all such paths. We recall that 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 endpoints are in different subsets of the partition.

It is assumed that each edge of G has an *unknown positive value*. We denote by $f(e) > 0$ the unknown value of edge $e \in E$. We assume that a function $F : 2^E \to [0, \infty)$ determines the value of a set of edges. That is, for any set of edges $H \subseteq E$, $F(H)$ is the value of H. According to the above definition, $F(P_{s,t})$ is the value associated to a path $P_{s,t} \in \mathcal{P}_{s,t}$. Depending on the context, an *optimal path* between nodes $s, t \in V$ is either a path $P^*_{s,t}$ minimizing $F(P_{s,t})$ over all paths $P_{s,t} \in \mathcal{P}_{s,t}$, or a path $P^*_{s,t}$ maximizing $F(P_{s,t})$ over all paths $P_{s,t} \in \mathcal{P}_{s,t}$. In the following, we denote by $\delta^*_{s,t}$ *the value of an optimal path* in G between nodes s and t.

In some situations, rather than requiring an optimal path between nodes s and t, we might be less ambitious and be satisfied with a path providing some performance guarantee. To account for such situations, we introduce the concept of an α-approximation of an optimal path, which is formally defined as follows.

Definition 1. *Given $\alpha \geq 1$, a path $P_{s,t}$ provides an* α-approximation *of an optimal path between nodes s and t if and only if it holds that:*

- $F(P_{s,t})/\delta^*_{s,t} \leq \alpha$, *when the goal is to minimize* $F(\cdot)$, *or*
- $\delta^*_{s,t}/F(P_{s,t}) \leq \alpha$ *when the goal is to maximize* $F(\cdot)$.

Given $\alpha \geq 1$ and two nodes s and t in V, in order to discover an α-approximation of an optimal path between these nodes, and to *be able to guarantee* that the discovered path is indeed an α-approximation, an algorithm has to uncover the unknown values of some 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 value of an edge or a set of edges. When an algorithm queries the oracle for the values of a set of edges $H \subseteq E$, the oracle reveals to the algorithm the value $f(e)$ of all the requested edges $e \in H$. We assume that the algorithm knows the function $F(\cdot)$ and can therefore compute the value $F(H')$ of any subset $H' \subseteq H$. For short, we say that an algorithm *uncovers a set of edges* when we refer to all this process.

We emphasize that for an algorithm 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 bound on $\delta^*_{s,t}$ to be inferred (a lower bound in the minimization case, and an upper bound in the maximization case). 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 (in which case we say that $P_{s,t}$ is an ∞-*approximation*). In the following, 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 a (s,t)-path. More precisely, an α-certificate of a (s,t)-path is defined as follows.

Definition 2. *A set of edges* $\mathcal{C} \subseteq E$ *is an* α-certificate *of a* (s,t)*-path in* G if and only if

- *the value of the edges in* $\mathcal{C}$ *is known, whereas edges in* $E \setminus \mathcal{C}$ *have unknown values,*
- $\mathcal{C}$ *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.*

When the nodes s and t are clear from the context, we will simply say an α-certificate. 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.

2.2 The OPD problem

With the above definitions in mind, we now present the OPD problem. We first define what is an *instance* of the problem, that is, the information that is known prior to solving the problem. We then define what is a *feasible solution* as well as the *objective function* to be optimized and formally formulate the OPD problem. Finally, we present the performance measures that we will use to compare algorithms solving the OPD problem.

Instance An *instance* I of the OPD problem is defined by a complete undirected graph $G = (V, E)$ with n nodes, two special nodes s and t in V, a targeted approximation factor $\alpha \geq 1$, a function $F : 2^E \to [0, \infty)$ that defines the value of any set of edges, an optimization goal either maximization or minimization, and an oracle $\mathcal{O}$ that can be accessed by an algorithm to *uncover* an edge or a set of edges. We say that the *size* of an instance I is the number of nodes n of the complete graph G, and it is denoted by $|I|$.

Feasible Solution We define a *feasible solution* of the OPD problem as an α-certificate of a (s, t)-path. In other words, a feasible solution is a set of fully uncovered edges allowing to guarantee that an α-approximation of an optimal path between nodes s and t has been discovered.

Objective Function The objective function to be minimized associates to each subset of edges $H \subseteq E$ its cardinality $|H|$. The goal is therefore to minimize the number of queries made to the oracle in order to obtain an α-certificate of a (s, t)-path.

Mathematical Formulation Formally, we can define the OPD problem as follows

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

Performance Measures Since the OPD problem aims at minimizing the number of queries made to the oracle for discovering an α-approximation of an optimal path, a natural performance measure for the efficiency of an algorithm solving the OPD problem is the worst-case number of queries it does for instances of size n. Let us denote by $U(\mathcal{A}(I))$ the set of edges whose

value has been uncovered by an algorithm $\mathcal{A}$ when solving the OPD 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 OPD problem as follows.

Definition 3. *The* number of queries *for instances of size n of an algorithm $\mathcal{A}$ that solves the OPD problem is β_n if and only if $|U(\mathcal{A}(I))| \leq \beta_n$ for all instances I of size n.*

In Section 4, we shall use the *number of queries* 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 do a number of queries which is of the same order of magnitude than the total number of links. Interestingly, any algorithm may still have to do 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 some large value of α. In that respect, the *number of queries* does not allow to correctly differentiate algorithms according to their performance. Therefore, we introduce a new measure of the performance of an algorithm, the *query ratio*, which is defined as follows.

Definition 4. *The* query ratio *of an algorithm $\mathcal{A}$ that solves the OPD problem is defined by the following ratio.*

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

where the maximum is taken over all instances of size n, and $|\mathcal{C}^{\alpha}_{\min}(I)|$ is the size of a smallest (in cardinality) α-certificate for instance I.

2.3 Assumptions on the function $F(\cdot)$ and two examples

Even though the OPD problem has been defined without any specific assumption on the function $F(\cdot)$, in this section we set conditions on that function. We explain below in detail our assumptions with respect to $F(\cdot)$.

We recall that $F : 2^E \to [0, \infty)$ is applied to sets and it depends on the values of each edge $f(e)$ in the set. W.l.o.g., we also assume that $F(e) = f(e)$, for all $e \in E$. We assume that the function $F(\cdot)$ satisfies the following conditions:

(i) Given $H \subset E$, it holds that:

$$F(H) = 0 \iff H = \{\emptyset\}, \text{ for the } \textit{minimization} \text{ problem, and}$$

$$F(H) = \infty \iff H = \{\emptyset\}, \text{ for the } \textit{maximization} \text{ problem.}$$

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

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

(iii) Given $H \subset H' \subseteq E$ (where H is a proper subset of H'), it holds that:

$$F(H') < F(H), \text{ for the } \textit{minimization} \text{ problem, and}$$

$$F(H') > F(H), \text{ for the } \textit{maximization} \text{ problem.}$$

The role of these three conditions, or the intuition behind them, is the following. Condition (i) imposes that at least one edge of a path is required to have a lower or upper bound on the value of that path, depending on whether the problem has minimization or maximization as optimization goal respectively. Therefore, paths with no edge queried yet have totally unknown value. Secondly, conditions (ii) and (iii) guarantee the monotonicity of the function $F(\cdot)$. Particularly, we require that $F(\cdot)$ is a strictly decreasing function when the optimization goal is minimization, and that it is a strictly increasing function when the optimization goal is maximization. Besides, Condition (iii) establishes that the value of any strict subpath (H') of a path (H) is a lower-bound on the value of H for the minimization case, and an upper-bound for the maximization case.

We assume as well that the optimal value of $F(\cdot)$ can be computed in polynomial time when all the values $f(\cdot)$ of the edges are known. Therefore, the OPD problem arises in a context in which obtaining the value $f(\cdot)$ of the edges dominates the total cost of computing the optimal path according to $F(\cdot)$.

We now briefly describe two interesting problems arising in communication networks for which the above conditions (i), (ii), and (iii) on the function $F(\cdot)$ are satisfied.

> **The sum** Consider the case when $F(H) = \sum_{e \in H} f(e)$. It is easy to see that this function satisfies conditions (i), (ii), and (iii) for the minimization problem when $f(e) \in (0, \infty)\ \forall e \in E$. Assume that $f(e)$ represents the length of the edge $e \in E$. Therefore, the function $F(\cdot)$ applied to a path computes the length of that path. In that case, the OPD problem seeks a shortest path between two nodes uncovering the least possible amount of edges. This problem is known in the literature

as the *Shortest Path Discovery Problem* and was introduced by Csaba Szepesvári in [18].

The product On the other hand, consider the case when $F(H) = \prod_{e \in H} f(e)$. It is easy to see that this function satisfies conditions (i), (ii), and (iii) for the maximization problem when $f(e) \in [0, 1] \, \forall e \in E$. Assume that $f(e)$ represents the probability of successful transmission of a packet over the edge $e \in E$. In that case, assuming independence among probabilities, the OPD problem seeks a path with the highest probability of successful transmission from the source to the destination.

In order to fully exemplify the impact of conditions (i), (ii), and (iii), we also present a problem where these assumptions on the function $F(\cdot)$ are not satisfied.

The minimum Consider the case when $F(H) = \min_{e \in H} f(e)$. We observe that Condition (iii) is not verified by this function for the maximization problem when $f(e) \in (0, \infty)$. Indeed, if e^* is the edge with minimum value of H, we have the following equality $F(e^*) = F(H)$ and Condition (iii) requires a strict inequality for any set of edges of H. Besides, we need to know the value of all $e \in H$ in order to obtain the value of $F(H)$. Hence, none sub path of a path provides a lower-bound of the value of that path.

For clarity of the presentation, in the rest of the paper we consider only the case where the goal is the minimization of $F(\cdot)$. However, all the techniques used for the minimization apply directly to the maximization of $F(\cdot)$. Hence, the results that we obtain apply to both cases, maximization and minimization of $F(\cdot)$.

2.4 Homogeneous knowledge versus complete graph

In our analysis, we assume homogeneous initial knowledge for the value of the edges, in the sense that it is only known at the beginning that this value is finite and that $f(e) > 0 \forall e \in E$. Therefore, an algorithm is not able to give priority to one edge over another when it picks the next edge to uncover. We also assume that the graph is complete. Hence, again, the algorithm is not able a priori to discard any edge to be uncovered and eventually all of them can be queried.

We remark that the problem can be studied assuming heterogeneous knowledge on the value of the edges or a different topology. 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. Therefore, an algorithm would be able to decide not to query an edge that is not present in the graph. For instance, in the case of the sum, this value might be ∞ or any large enough number so that the algorithm considers impossible the presence of that edge in any solution. In the case of the product, as it was presented earlier, the value would be zero.

Hence, our assumption of an homogeneous knowledge can also model situations in which there is no previous knowledge of the topology of the graph, or in other words, a priori all edges are equally likely to be part of a solution because the network is totally unknown. Thus, we focus on the case of complete graphs with homogeneous knowledge while we leave the study of the problem with heterogeneous knowledge as future work.

3 Related Work

Shortest path problems appear as subproblems in a variety of applications, e.g., for planning a route to a destination that avoids obstacles or for routing flows in communication networks. These problems have been extensively studied in a setting where the decision-maker has full information about the graph. Under this assumption, they can be solved using well-known algorithms, such as Bellman-Ford and Dijkstra's algorithms [3, 9, 7] for the single-source shortest path problem, or FloydWarshall and Johnson's algorithms [8, 12] for the all pairs shortest path problem (see also [4] for additional algorithms and associated evaluations). Researches have also considered the use of additionnal knowledge on the graph in order to speedup the search [13]. For instance, the A^* algorithm [11] reduces the amount of time required to find a shortest path between two given nodes by using an heuristic evaluation function providing a lower bound on the distance from each node to the destination.

In contrast, relatively few works have considered the setting in which the decision-maker initially has incomplete information about the graph and acquires this information as the search for an optimal path progresses. In the OPD problem, we assume that the weights of the edges are initially totally unknown and our goal is to uncover as few edges as possible in order to discover a path providing a certain performance guarantee.

As already mentioned, a particular case of the OPD problem, known as

the *Shortest Path Discovery Problem* [18], occurs for additive cost functions, that is, when the cost of a path is the sum of the cost of the edges it is comprised of. In [18], the authors present a greedy algorithm that solves this problem. The algorithm is greedy because it 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 [18] the performance of algorithms was measured with the number of queries, we show that this performance measure is not appropriate and propose rather to compare algorithms through their query ratios. We also propose an algorithm for solving the OPD problem. Although this algorithm has performances similar to the greedy algorithm of [18] in the case of an additive cost function, its main interest is that the analysis of its query ratio is far much simpler. Our algorithm 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 for the shortest path case has a long history, see [17, 15, 14]. For example, in [6] the authors suggest the Birectional Heuristic Front-to-Front algorithm (see also [10, 5] for improved versions of this algorithm).

It is worth noticing that standard algorithms, such as A^* or Dijkstra's algorithm, perform poorly for solving the *Shortest Path Discovery Problem* 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 $(n-1)$.

Apart from [18], 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). In their model, a weighted graph with known topology but unknown edge weights is given, along with designated source and destination nodes, and

the goal is to devise a discovery protocol that identifies the shortest path. The discovery process is operated through agents that traverse the network and report back their total cost. The process proceeds in multiple rounds, where multiple agents can be sent in each round. The objective is to devise such protocols of minimum complexity, where the complexity of a protocol is measured according to the number of rounds of transmissions, and the number of participating agents. Although this problem is similar to the *Shortest Path Discovery Problem*, the main difference is that only the total cost of a path can be queried.

Another situation in which the information about the graph is revealed progressively to the decision-maker is considered in [16]. The authors study shortest-path problems when the graph is not known in advance, but is specified dynamically. For instance, they consider layered graphs, assuming that the graph is given to the decision-maker one stage at a time (i.e., layer by layer). They seek dynamic decision rules that optimize the worst-case ratio of the distance covered to the length of the (statically) optimal path. They describe optimal decision rules for two cases: layered graphs of width two, and two-dimensional scenes with unit square obstacles. Although the problem introduced in [16] and the OPD problem share the assumption that the weights of the edges are initially unknown, they correspond to different models and do not have the same objective.

4 Lower Bounds on the Number of Queries β_n

In this section, we present lower bounds on the number of queries required by any algorithm providing a finite 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 (i), the value of $P^*_{s,t}$ can be estimated by 0 if the problem is a minimization problem, or by ∞ if the problem is a maximization problem. Therefore, $\mathcal{C}$ can not guarantee any bound on the value of an optimal path, neither an upper bound for when the optimization goal is maximization nor a lower bound when the optimization goal is minimization. Thus, there is a contradiction with the fact that $\mathcal{C}$ is an α-certificate, since $\mathcal{C}$ cannot guarantee any finite 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 $P_{s,t} \in \mathcal{P}_{s,t}$. If the instance has minimization as its optimization goal, set

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

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

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

If the instance has maximization as its optimization goal, set

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

Since $\max\{F(P_{s,t} \cap \mathcal{C}) : P_{s,t} \in \mathcal{P}_{s,t}\} \geq \delta^*_{s,t}$, we can obtain the corresponding equivalent conclusion.

This implies that $\mathcal{C}$ is an $\alpha-$certificate for the above finite value of α, respectively defined for the minimization and maximization case.

□

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 and minimization as optimization goal 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 and one set of size $n-p$. 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 (iii) 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 with minimization as optimization goal 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 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 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.

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 a malicious adversary constructed for any algorithm that plays the role of the oracle. 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 solves the OPD problem, there exists an instance I with approximation factor α and with minimization as optimization goal 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. 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 1.

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 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)

Algorithm 1 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

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 (iii) 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 1). 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 1). 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 1). 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 1). 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 1). 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 1). 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 1). 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 1). 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 1).

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 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,*

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

5.2 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 consider only instances with minimization as optimization goal. Nevertheless, all the results also apply for instances with maximization as optimization goal. 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 2.

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 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. First, Algorithm 2 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 2). 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 2). Then, the algorithm picks the closest nodes to s and t among the nodes that have not been previously picked. The closest node to s (resp. t) is denoted by s^* (resp. t^*) (see lines 14 and 15 in Algorithm 2). Finally, the algorithm computes whether $PATH_{prop}$ is an α-approximation in line 16. If that is the case, Algorithm 2 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 $F(PATH_{s,s^*} \cup PATH_{t^*,t})$ is a lower bound on $\delta^*_{s,t}$.

Algorithm 2 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 $P_{s,t} := PATH_{prop}$.

We now analyze the query ratio of Algorithm 2. 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 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 5 to 7 of Algorithm 2. 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 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 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. 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 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 optimal path between u and v in the graph induced by the set of nodes U.

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

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

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

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

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 2 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

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

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

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

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 $F(P^*_{s,s^*_i}|_{m_s} \cup P^*_{t^*_i,t}|_{m_t}) < F(P_{s,t})$, i.e, the ratio of equation (3) is strictly higher 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) to (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:

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

□

We are now in position to present the query ratio of Algorithm 2.

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

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

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:

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

□

6 Comparison with Previous Algorithms

In this section we compare the query ratio of Algorithm 2 with the query ratio of two algorithms already presented in the literature. We first analyze the query ratio of a greedy algorithm presented in [18] 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 2 with two algorithms also mentioned in [18].

6.1 Analytical Comparison

We first note that our upper-bound on the query ratio of Algorithm 2 is strictly less than 2. On the other hand, as mentioned in Section 3, the query ratio of the well known search algorithm A^* (cf. [11]) is $n/2$ which is much higher than the upper bound obtained for the query ratio of Algorithm 2.

We now compare the performance of Algorithm 2 with an extension of the algorithm presented in [18] 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 $\mathcal{A}_{hom}^{Gre}$ the greedy algorithm of [18] 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. I We consider in $\mathcal{A}_{hom}^{Gre}$ 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 $\mathcal{A}_{hom}^{Gre}$ 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 2 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

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

- *Otherwise,*

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

Proof. In this proof we use a similar notation than in the proof of Lemma 6, that is, we say that in round j Algorithm 2 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 2.

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 $\mathcal{A}^{Gre}_{hom}$ finds that the optimal path is the direct path (s, t). In the next $2(n-1)$ rounds, $\mathcal{A}^{Gre}_{hom}$ finds that the optimal paths are all the two-hop paths. Hence, algorithm $\mathcal{A}^{Gre}_{hom}$ 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 2 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 $\mathcal{A}^{Gre}_{hom}$ 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 $\mathcal{A}^{Gre}_{hom}$ 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

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

and

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

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 2 at step j. Therefore, $\mathcal{A}_{hom}^{Gre}$ uncovers the same set of edges than Algorithm 2 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 2 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 $2(n - 2)$. □

Using this result we state that the query ratio of algorithm $\mathcal{A}_{hom}^{Gre}$ and Algorithm 2 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:*

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

In Theorem 2 we prove an upper-bound on the query ratio of Algorithm 2 for instances of the OPD problem with $\alpha = 1$ and in Theorem 3 we show that the query ratio of algorithm $\mathcal{A}_{hom}^{Gre}$ and Algorithm 2 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 2 and $\mathcal{A}_{hom}^{Gre}$ have the same query ratio. However, the query ratio of Algorithm 2 is easier to analyze.

6.2 Numerical Comparison

In this section, we experimentally analyze the algorithm presented in Subsection 5.2. 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 2 for the OPD problem with $\alpha = 1$ to compare it with upper-bound presented in Section 5.2. 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 2 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 2 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 2 (Algorithm "algo" in Figure 1) with two modifications of the greedy algorithm presented in [18].

Query Ratio of Algorithm 2	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

(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 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 algorithm as random and in figure 1 it is represented as "rand".

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 2, 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 2 achieves the value 1 for a lower number of uncovered edges comparing with the random algorithm.

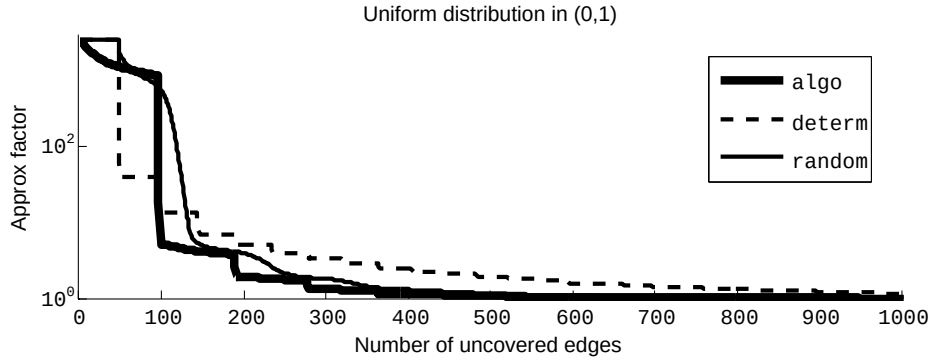

Figure 1: Approximation factor evolution comparison. Uniformly distributed edges and 50 nodes. Y axis in logarithmic scale.

Furthermore, the average approximation factor of Algorithm 2 is smaller than the deterministic algorithm when the number of uncovered edges is higher than 100 and always less than the random algorithm.

7 Conclusions and Future Work

In this document we present and study the OPD problem. For a given function $F(\cdot)$ that is applied to set of edges, an algorithm that solves the OPD problem aims to find the path that optimizes the value of $F(\cdot)$ when it is applied to a path, while at the same time it has to minimize the number of queries it uses to find such path. We observed that interesting functions $F(\cdot)$ arise as particular cases of the OPD problem. For example, when $F(\cdot)$ is the sum of the values of each edge in the evaluated set, the OPD problem becomes the SPD problem. Equivalently, when the function $F(\cdot)$ is the product of tthe values of each edge in the evaluated set, and the value of each edge represents a probability of success, then the OPD problem aims to find the path with maximum probability of successful arrival of packets to the destination.

We show that the number of queries 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 new 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.

Therefore, we consider that the query ratio is the correct measure to take into account in the design of algorithms that solve the OPD 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 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. Nevertheless, we believe that these two mentioned extensions are closely related and that they can 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] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

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

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

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

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

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

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

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

[11] 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.

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

[13] 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.

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

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

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

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

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