

Graph Exploration: The Impact of a Distance Constraint

Stéphane Devismes*

Yoann Dieudonné[†]

Arnaud Labourel[‡]

Abstract

A mobile agent, starting from a node s of a simple undirected connected graph $G = (V, E)$, has to explore all nodes and edges of G using the minimum number of edge traversals. To do so, the agent uses a deterministic algorithm that allows it to gain information on G as it traverses its edges. During its exploration, the agent must always respect the constraint of knowing a path of length at most D to go back to node s . The upper bound D is fixed as being equal to $(1 + \alpha)r$, where r is the eccentricity of node s (i.e., the maximum distance from s to any other node) and α is any positive real constant. This task has been introduced by Duncan et al. [14] and is known as *distance-constrained exploration*.

The *penalty* of an exploration algorithm running in G is the number of edge traversals made by the agent in excess of $|E|$. In [19], Panaite and Pelc gave an algorithm for solving exploration without any constraint on the moves that is guaranteed to work in every graph G with a (small) penalty in $\mathcal{O}(|V|)$. Hence, a natural question is whether we could obtain a distance-constrained exploration algorithm with the same guarantee as well.

In this paper, we provide a negative answer to this question. We also observe that an algorithm working in every graph G with a linear penalty in $|V|$ cannot be obtained for the task of *fuel-constrained exploration*, another variant studied in the literature.

This solves an open problem posed by Duncan et al. in [14] and shows a fundamental separation with the task of exploration without constraint on the moves.

Keywords: exploration, graph, mobile agent.

*MIS Lab., Université de Picardie Jules Verne, France. E-mail: stephane.devismes@u-picardie.fr

[†]MIS Lab., Université de Picardie Jules Verne, France. E-mail: yoann.dieudonne@u-picardie.fr

[‡]Aix Marseille Univ, CNRS, LIS, Marseille, France. Email: arnaud.labourel@lis-lab.fr

1 Introduction

1.1 Background

Exploring an unknown environment is a fundamental task with numerous applications, including target localization, data gathering, and map construction. The considered environments can vary considerably in type: they may be terrains, networks, three-dimensional spaces, etc. In hazardous situations, it is generally more appropriate to delegate exploration to a mobile artificial entity, hereinafter referred to as an agent, which could be an autonomous robot or a vehicle remotely controlled by a human.

In this paper, we focus our attention on the task of exploration by a mobile agent of a network modeled as a graph, and more specifically on a variant where the agent must always know a path from its initial node to its current node that does not exceed a given upper bound. This requirement finds its justification in applications where the agent needs to always keep the possibility of returning quickly to its base or to always maintain a certain distance to ensure ongoing communication with its base.

1.2 Model and Problem Definition

An agent is assigned the task of exploring a simple¹ finite undirected graph $G = (V, E)$, starting from a node s , called the *source node*. The exploration requires visiting (resp. traversing) at least once every node (resp. every edge). Thus, G is supposed to be connected. We denote by $\deg(v)$ the degree of a node v in G , and by $|V|$ (resp. $|E|$) the order (resp. size) of G .

We make the same assumption as in [14] which states that the agent has unbounded memory and can recognize already visited nodes and traversed edges. This is formalized as follows. Nodes of G have arbitrary pairwise distinct labels that are positive integers. The label of s will be denoted by l_s . Distinct labels, called *port numbers* and ranging from 0 to $\deg(v) - 1$, are arbitrarily assigned locally at each node v to each of its incident edge. Hence, each edge has two ports, one per endpoint, and there is no a priori relationship between these two ports. At the beginning, the agent located at node s learns only its degree and its label. To explore the graph, the agent applies a deterministic algorithm that makes it act in steps: at each step, the algorithm selects a port number p at the current node v (on the basis of all information that has been memorized so far by the agent) and then asks the agent to traverse the edge having port p at v . At that point, the agent traverses the given edge at any finite positive speed and, once it enters the adjacent node, it learns (only) its degree, its label as well as the incoming port number. Afterwards, the agent starts the next step, unless the algorithm has terminated.

Roughly speaking, the memory of the agent is the total information it has collected since the beginning of its exploration. We formalize it as follows: the memory of the agent, after the first t edge traversals, is the sequence (M_0, M_1, \dots, M_t) , where M_0 is the information of the agent at its start and M_i is the information acquired right after its i th edge traversal. Precisely, the initial memory M_0 is represented by the 4-tuple $(l_0, d_0, -1, -1)$, where l_0 and d_0 correspond to the label of the source node and its degree respectively. Then, if $i \geq 1$, M_i is the 4-tuple (l_i, d_i, p_i, q_i) where

¹In [14], where the task of distance-constrained exploration that we will study has been introduced, the authors do not mention explicitly whether they consider simple graphs or multigraphs. A closer look at their work can reveal, though, that their results hold for multigraphs. However, it is important to note that, in the context of the impossibility proof that we will conduct, the choice of focusing only on simple graphs does not lead to a loss of generality. In fact, it broadens the scope of our result, ensuring that it holds for both simple graphs and multigraphs.

l_i is the identifier of the occupied node after the i th edge traversal, d_i the degree of that node, p_i is the port by which the agent has left the node l_{i-1} to make the i th edge traversal, and q_i the port by which the agent entered node l_i via its i th edge traversal.

Without any constraint on the moves, the task of exploration in the above model is called *unconstrained exploration*. However, in this paper, we study a variant introduced in [14] and called *distance-constrained exploration*. In this variant, the agent must deal with the additional constraint of always knowing a path (i.e., a sequence of already explored edges) of length at most D from its current location to node s . The value D is fixed to $(1 + \alpha)r$, where r and α respectively correspond to the eccentricity of node s and an arbitrary positive real constant. In [14], two scenarios are considered: one where both r and α are initially known to the agent, and another where only α is known. In this paper, we adopt the more challenging scenario from a proof-of-impossibility perspective, where both r and α are initially known to the agent, in order to give our result a greater impact.

An instance of the problem is defined as the tuple (G, l_s, α) . We will denote by $\mathcal{I}(\alpha, r)$ the set of all instances (G, l_s, α) such that the eccentricity of the source node s (of label l_s) in G is r .

An important notion involved in this paper is that of *penalty* incurred during exploration. First discussed by Pelc and Panaite in [19], the *penalty* of an exploration algorithm running in G is the number of edge traversals in excess of $|E|$ made by the mobile agent. In their seminal paper [19], Panaite and Pelc gave an algorithm for solving unconstrained exploration, which is guaranteed to work in every graph G with a (small) penalty of $\mathcal{O}(|V|)$. In the light of this result, an intriguing question naturally arises:

Would it be possible to design a distance-constrained exploration algorithm working in every graph G with a penalty of $\mathcal{O}(|V|)$?

1.3 Our Results

In this paper, we provide a negative answer to the above question. This negative answer is strong as we show that for any positive real α and every integer $r \geq 6$, there is no algorithm that can solve distance-constrained exploration for every instance $(G = (V, E), l_s, \alpha)$ of $\mathcal{I}(\alpha, r)$ with a penalty of $\mathcal{O}(|V|)$.

Moreover, we observe that the impossibility result remains true for another variant of constrained exploration studied in the literature, known as *fuel-constrained exploration*. In this variant, which was introduced by Betke et al. in [6], the context remains the same, except that now the agent must face a slightly different constraint: it has a fuel tank of limited size that can be replenished only at its starting node s . The size of the tank is $B = 2(1 + \alpha)r$, where α is any positive real constant and r is the eccentricity of s . The tank imposes an important constraint, as it forces the agent to make at most $\lfloor B \rfloor$ edge traversals before having to refuel at node s , otherwise the agent will be left with an empty tank and unable to move, preventing further exploration of the graph. The underlying graph and node s are chosen so that $r\alpha \geq 1$ (if not, fuel-constrained exploration is not always feasible).

Through our two impossibility results, we show a fundamental separation with the task of unconstrained exploration. We also solve an open problem posed by Duncan et al. in [14], who asked whether distance-constrained or fuel-constrained exploration could be achieved with a penalty

of $\mathcal{O}(|V|)$ in every graph.²

1.4 Related Work

The problem of exploring unknown graphs has been the subject of extensive research for many decades. It has been studied in scenarios where exploration is conducted by a single agent and in scenarios where multiple agents are involved. In what follows, we focus only on work related to the former case (for the latter case, a good starting point for the curious reader is the survey of Das [10]).

In the context of theoretical computer science, the exploration problem was initially approached by investigating how to systematically find an exit in a maze (represented as a finite connected subgraph of the infinite 2-dimensional grid in which the edges are consistently labeled North, South, East, West). In [21], Shannon pioneered this field with the design of a simple finite-state automaton that can explore any small maze spanning a 5×5 grid. More than twenty years later, Budach [8] significantly broadened the scope of study by showing that no finite-state automaton can fully explore all mazes. Blum and Kozen showed in [7] that this fundamental limitation can be overcome by allowing the automaton to use only two pebbles. The same authors naturally raise the question of whether one pebble could suffice. A negative answer to this question is provided by Hoffmann in [16].

In the following decades, much of the attention shifted to the exploration of arbitrary graphs, for which the solutions dedicated to mazes become obsolete. Studies on the exploration of arbitrary graphs can be broadly divided into two types: those assuming that nodes in the graph have unique labels recognizable by the agent, and those assuming anonymous nodes.

As for mazes, exploration of anonymous graphs has been investigated under the scenario in which the agent is allowed to use pebbles [5, 9, 12, 13]. In [13], Dudek et al. showed that a robot provided with a single movable pebble can explore any undirected anonymous graph using $\mathcal{O}(|V| \cdot |E|)$ moves. If the pebble is not movable and can only be used to mark the starting node of the agent, it is shown in [9] that exploration is still possible using a number of edge traversals polynomial in $|V|$. In [12], the authors investigated the problem in the presence of identical immovable pebbles, some of which are Byzantine (a Byzantine pebble can be unpredictably visible or invisible to the agent whenever it visits the node where the pebble is located) and designed an exploration algorithm working in any undirected graph provided one pebble always remains fault-free. In the case where the graph is directed, exploration becomes more challenging due to the impossibility of backtracking and may even be sometimes infeasible if the graph is not strongly connected. However, in the scenario where the agent evolves in strongly connected directed graphs, Bender et al. proved that $\Theta(\log \log |V|)$ movable pebbles are necessary and sufficient to achieve a complete exploration.

When no marking of nodes is allowed, exploration of undirected anonymous graphs becomes obviously more difficult, and detecting the completion of the task cannot be always guaranteed without additional assumptions. If randomization is allowed, we know from [2] that a simple random walk of length $\mathcal{O}(\Delta^2 |V|^3 \log |V|)$ can cover, with high probability, all nodes of an undirected graph $G = (V, E)$ of maximal degree Δ . Such a random walk can thus be used to fully explore a graph (and eventually stop), by initially providing the agent with an upper bound on $|V|$ and by making it go back and forth over each edge incident to a node of the random walk. If randomization is not

²In [14], the authors also investigated a third variant known as *rope-constrained exploration*, but they did not include it in their open problem. This point is further detailed at the end of the related work section.

allowed, all nodes of an undirected anonymous graph can still be visited, using universal exploration sequences (known as UXS) introduced in [18]. Formally, a sequence of integers x_1, x_2, \dots, x_k is said to be a UXS for a class \mathcal{G} of graphs, if it allows an agent, starting from any node of any graph $G \in \mathcal{G}$, to visit at least once every node of G in $k + 1$ steps as follows. In step 1, the agent leaves its starting node v_1 by port 0 and enters some node v_2 . In step $2 \leq i \leq k + 1$, the agent leaves its current node v_i by port $q = (p + x_{i-1}) \bmod \deg(v_i)$, where p is the port by which it entered v_i in step $i - 1$, and enters some node v_{i+1} . In [20], it is proven that for any positive integer n , a UXS of polynomial length in n , for the class of graphs of order at most n , can be computed deterministically in logarithmic space and in time polynomial in n . As with random walks, this can be used to fully explore a graph, by initially providing the agent with an upper bound n on $|V|$ and by instructing it to traverse each edge incident to a node visited by a UXS for the class of graphs of order at most n . Therefore, whether deterministically or not, it can be easily shown from [2, 20] that an agent can explore any undirected anonymous graph of order at most n even without the ability to mark nodes as long as it has a memory of size $\mathcal{O}(\log n)$: in [15] Fraigniaud et al. gave a tight lower bound by showing that a memory of size $\Omega(\log n)$ is necessary.

The exploration of labeled graphs was a subject of research in [1, 3, 4, 6, 11, 14, 19], with a particular focus on minimizing the number of edge traversals required to accomplish the task.

In [11], Deng and Papadimitriou showed an upper bound of $d^{\mathcal{O}(d)}|E|$ moves to explore any labeled directed graph that is strongly connected, where d , called the deficiency of the graph, is the minimum number of edges that have to be added to make the graph Eulerian. This is subsequently improved on by Albers and Henzinger in [1], who gave a sub-exponential algorithm requiring at most $d^{\mathcal{O}(\log d)}|E|$ moves and showed a matching lower bound of $d^{\Omega(\log d)}|E|$. For arbitrary labeled undirected graphs, the fastest exploration algorithm to date is the one given by Panaite and Pelc in [19], which allows an agent to traverse all edges using at most $|E| + \mathcal{O}(|V|)$ moves. Their algorithm thus entails at most a linear penalty in $|V|$, which is significantly better than other classic strategies of exploration (for instance, the standard Depth-First Search algorithm, which takes $2|E|$ moves, has a worst-case penalty that is quadratic in $|V|$). This upper bound of $\mathcal{O}(|V|)$ is asymptotically tight as the penalty can be in $\Omega(V)$, even in some Eulerian graphs. In particular, it is the case when an agent is required to explore a simple line made of an odd number $|V| \geq 3$ of nodes, by starting from the median node: the penalty is then necessarily at least $\frac{|V|-1}{2}$.

The papers [3, 4, 6, 14] are the closest to our work. As mentioned in Section 1.3, the problem of fuel-constrained exploration of a labeled graph was introduced by Betke et al. in [6]. In this paper, the authors gave fuel-constrained algorithms using $\mathcal{O}(|E| + |V|)$ moves but for some classes of graphs only. This line of study was then continued in [3] (resp. in [4]) in which is provided an $\mathcal{O}(|E| + |V|\log^2 |V|)$ algorithm (resp. an $\mathcal{O}(|E| + |V|^{1+o(1)})$ algorithm) working in arbitrary labeled graphs. Finally, Duncan et al. [14] obtained solutions requiring at most $\mathcal{O}(|E| + |V|)$ edge traversals for solving fuel-constrained and distance-constrained explorations. Precisely, the authors did not directly work on these two variants, but on a third one called *rope-constrained exploration*, in which the agent is tethered to its starting node s by a rope of length $L \in \Theta(r)$ that it unwinds by a length of 1 with every forward edge traversal and rewinds by a length of 1 with every backward edge traversal. Hence, they designed an $\mathcal{O}(|E| + |V|)$ algorithm for rope-constrained exploration which can be transformed into algorithms solving the two other constrained exploration problems with the same complexity. However, these algorithms all carry a worst-case penalty proportional to $|E|$ even when $|E| \in \Theta(|V|^2)$. As an open problem, Duncan et al. then asked whether we could obtain algorithms for solving constrained explorations with a worst-case penalty linear in $|V|$, as shown

in [19] when there is no constraint on the moves. Precisely, Duncan et al. explicitly raised the question for fuel-constrained and distance-constrained explorations and not for rope-constrained exploration. This is certainly due to the following straightforward observation: at any point of a rope-constrained exploration, the difference between the number of forward edge traversals and the number of backward edge traversals can never exceed L . Thus, the penalty of any rope-constrained exploration algorithm executed in any graph $G = (V, E)$ is always at least $|E| - L$, which can be easily seen as not belonging to $\mathcal{O}(|V|)$ in general, since L belongs to $\Theta(r)$ and thus to $\mathcal{O}(|V|)$.

2 Preliminaries

In this section, we introduce some basic definitions, conventions, operations, and results that will be used throughout the rest of the paper.

We call *consistently labeled graph* a simple undirected labeled graph in which nodes have pairwise distinct identifiers and in which port numbers are fixed according to the rules given in Section 1.2. We will sometimes use simple undirected graphs with no identifiers and with no port numbers, especially when describing intermediate steps of constructions: these graphs will be called *unlabeled graphs*. However, when the context makes it clear or when the distinction between labeled and unlabeled graphs is insignificant, we will simply use the term *graph*.

A graph G is said to be *bipartite* if its nodes can be partitioned into two sets U and U' such that no two nodes within the same set are adjacent. If G is also connected, such partition $\{U, U'\}$ is unique and referred to as the *bipartition* of G . A graph is said to be *regular* (resp. *k-regular*) if each of its nodes has the same degree (resp. has degree k).

Below is a simple lemma about regular bipartite graphs. It follows from the fact that a k -regular bipartite graph of order $2n$, with $n \geq k \geq 1$, can be constructed by taking two sets of nodes $U = \{u_0, u_1, \dots, u_{n-1}\}$ and $U' = \{u'_0, u'_1, \dots, u'_{n-1}\}$ and then by adding edges so that for every i and $j \in [0..n-1]$, there is an edge between u_i and u'_j iff $j = (i + x) \bmod n$ for some $x \in [0..k-1]$.

Lemma 2.1 *For every positive integer k , and for every integer $n \geq k$, there exists a k -regular bipartite graph whose bipartition $\{U, U'\}$ satisfies $|U| = |U'| = n$.*

The impossibility proof described in Section 3 closely relies on a family of graphs $\mathcal{F}(\ell, w, r)$, where $\ell \geq 2$, $w \geq 4$ and $r \geq 6$ are integers. Precisely, $\mathcal{F}(\ell, w, r)$ is defined as the set of all consistently labeled graphs that can be constructed by successively applying the following five steps.

- **Step 1.** Take a sequence of ℓ sets $V_1, V_2, V_3, \dots, V_\ell$, each containing w nodes. For every $1 \leq i \leq \ell$, assign pairwise distinct labels to nodes of V_i ranging from $w(i-1) + 1$ to wi : these nodes will be called the *nodes of level i* . Then, for every $1 \leq i \leq \ell - 1$, add edges from nodes of V_i to nodes of V_{i+1} so that the graph induced by $V_i \cup V_{i+1}$ is a $\lfloor \frac{w}{4} \rfloor$ -regular bipartite graph denoted B_i whose bipartition is $\{V_i, V_{i+1}\}$ (such a graph exists by Lemma 2.1).
- **Step 2.** This step is made of $\ell - 1$ phases, and, in each phase $1 \leq i \leq \ell - 1$, we proceed as follows. Let $\beta = w \lfloor \frac{w}{4} \rfloor$ (which corresponds to the number of edges of the subgraph induced by $V_i \cup V_{i+1}$). Take a sequence of $\lfloor \frac{7\beta}{8} \rfloor$ new nodes $(g_1^i, g_2^i, g_3^i, \dots, g_{\lfloor \frac{7\beta}{8} \rfloor}^i)$ (by “new nodes”, we precisely mean nodes that do not belong (yet) to the graph under construction). These nodes will be called *gadgets*. For each $1 \leq j \leq \lfloor \frac{7\beta}{8} \rfloor$, assign label $w\ell + (i-1)\lfloor \frac{7\beta}{8} \rfloor + j$ to gadget g_j^i . Then, take a node u of V_i and a node v of V_{i+1} such that u and v are neighbors, remove the edge between u and v , and finally add an edge between u and g_j^i as well as v and g_j^i .

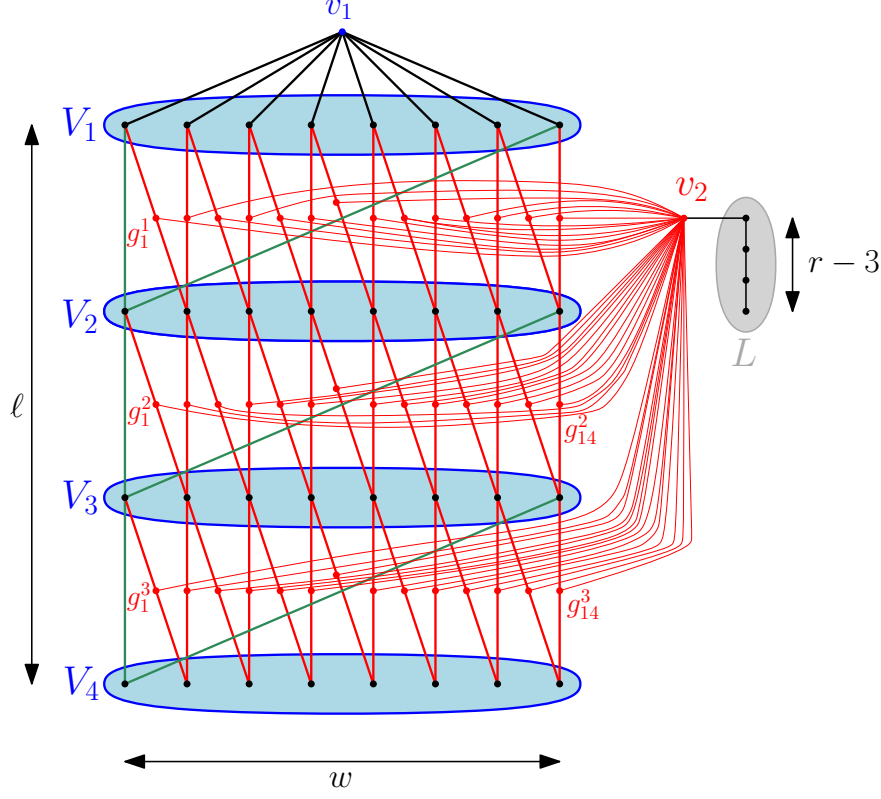


Figure 1: Example of a graph in $\mathcal{F}(4, 8, 7)$.

- **Step 3.** Take a pair of new nodes (v_1, v_2) . Assign label 0 (resp. label $w\ell + (\ell - 1)\lfloor \frac{7\beta}{8} \rfloor + 1$) to node v_1 (resp. node v_2). Then, add an edge between v_1 (resp. v_2) and each node of V_1 (resp. each gadget). Node v_2 will be called the *critical node*.
- **Step 4.** Take a line L made of $(r - 3)$ new nodes. Add an edge between the critical node v_2 and one of the endpoints of L . Then, for each node u of L , assign label $w\ell + (\ell - 1)\lfloor \frac{7\beta}{8} \rfloor + 1 + d$ where d is the distance from node v_2 to u . The nodes of L will form what will be called the *tail* of the graph and the farthest endpoint of L from v_2 will be called the *tail tip*.
- **Step 5.** For every node u belonging to the graph resulting from the successive application of steps 1 to 4, arbitrarily add pairwise distinct port numbers to each of its incident edges ranging from 0 to $\deg(u) - 1$.

Let us give some additional definitions. For every $1 \leq i \leq \ell - 1$, the layer $\mathcal{L}_G(i, i + 1)$ of a graph G resulting from the above construction is the set made of the following edges: those connecting a node of level i with a node of level $i + 1$, which will be called the *green edges* of $\mathcal{L}_G(i, i + 1)$, and those connecting a gadget of $(g_1^i, g_2^i, g_3^i, \dots, g_{\lfloor \frac{7\beta}{8} \rfloor}^i)$ with a node of level i or $i + 1$, which will be called the *red edges* of $\mathcal{L}_G(i, i + 1)$. When it is clear from the context, we will sometimes omit the reference to the underlying graph and use the notation $\mathcal{L}(i, i + 1)$ instead of $\mathcal{L}_G(i, i + 1)$.

An example of a graph belonging to $\mathcal{F}(4, 8, 7)$ is depicted, with its key components, in Figure 1. The next lemma is a direct consequence of our 5-step construction.

Lemma 2.2 *Let $\ell \geq 2$, $w \geq 4$ and $r \geq 6$ be integers. Let $\beta = w \lfloor \frac{w}{4} \rfloor$. For each graph G of $\mathcal{F}(\ell, w, r)$, we have the following properties:*

1. *G is consistently labeled. It contains $\ell - 1$ layers and ℓ levels.*
2. *The order of G is $w\ell + (\ell - 1) \lfloor \frac{7\beta}{8} \rfloor + r - 1$ and its number of gadgets is $(\ell - 1) \lfloor \frac{7\beta}{8} \rfloor$.*
3. *For every $1 \leq i \leq \ell - 1$, the number of red edges (resp. green edges) in $\mathcal{L}_G(i, i + 1)$ is $2 \lfloor \frac{7\beta}{8} \rfloor$ (resp. $\lceil \frac{\beta}{8} \rceil$).*
4. *Two nodes that are incident to the same green edge in G cannot be the neighbors of the same gadget.*

The lemma below addresses the eccentricity of the node labeled 0 as well as the length of some paths originating from it in graphs of the family we have proposed. As mentioned in Section 1.2, a path is viewed as a sequence of edges (and not a sequence of nodes).

Lemma 2.3 *Let $\ell \geq 2$, $w \geq 16$ and $r \geq 6$ be integers. Let G be a graph of $\mathcal{F}(\ell, w, r)$. We have the following two properties:*

1. *The eccentricity of the node of label 0 in G is r and for every $2 \leq i \leq \ell$, each node of level i in G is at distance at most 2 from a gadget.*
2. *For every $1 \leq i \leq \ell$, the length of a path from the node of label 0 to any node of level i in G is at least i if this path does not pass through the critical node of G .*

Proof. TOPROVE 0 □

In the impossibility proof of Section 3, we will need to perform three operations on graphs. Let G be a graph of $\mathcal{F}(\ell, w, r)$, with $\ell \geq 2$, $w \geq 4$ and $r \geq 6$. The first operation is **switch-ports**(G, v, p_1, p_2). If v is a node of G and p_1 and p_2 are distinct non-negative integers smaller than $\deg(v)$, this operation returns a graph G' corresponding to G but with the following modification: the edge having port p_1 (resp. p_2) at v in G now has port p_2 (resp. p_1) at v in G' . Otherwise, it simply returns G without any modification.

The second operation is **switch-edges**(G, v_1, v_2, p_1, p_2). Assume that for some $1 \leq i \leq \ell - 1$, p_1 and p_2 are the ports of green edges of $\mathcal{L}_G(i, i + 1)$ at nodes v_1 and v_2 respectively, and these two nodes belong to a same level in G . Also assume that the node v'_1 (resp. v'_2) that is reached by taking port p_1 (resp. p_2) from v_1 (resp. v_2) is not a neighbor of v_2 (resp. v_1) and that v_2 and v'_1 (resp. v_1 and v'_2) are not the neighbors of a same gadget. In this case, this operation returns a graph G' corresponding to G but with the following modifications: the edge e_1 (resp. e_2), originally attached to port p_1 (resp. p_2) at node v_1 (resp. v_2), is detached from this node and reattached to node v_2 (resp. v_1) with port p_2 (resp. p_1) assigned at its newly attached node. Otherwise, it simply returns G without any modification. An illustration of **switch-edges** is shown in Figure 2.a.

Notice that the requirements for modifying G using **switch-edges**(G, v_1, v_2, p_1, p_2) are essential. Indeed, the assumption that v'_1 (resp. v'_2) is not a neighbor of v_2 (resp. v_1) guarantees that we do not add an already existing edge: consequently, the graph obtained after the modification is still simple with the same number of green edges and node's degrees are left unchanged. Moreover, the assumption that v_2 and v'_1 (resp. v_1 and v'_2) are not the neighbors of a same gadget ensures that

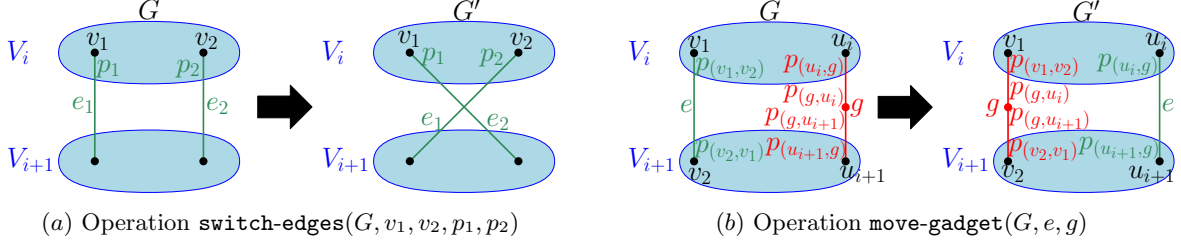


Figure 2: Illustrations of operations `switch-edges` and `move-gadget`.

the fourth property of Lemma 2.2 remains true after the modification. Hence, if G is a graph of $\mathcal{F}(\ell, w, r)$, then the graph returned by `switch-edges`(G, v_1, v_2, p_1, p_2) still belongs to $\mathcal{F}(\ell, w, r)$.

Finally, the third operation is `move-gadget`(G, e, g). Assume that for some $1 \leq i \leq \ell - 1$, e is a green edge of $\mathcal{L}_G(i, i + 1)$ and g is a gadget incident to a red edge of $\mathcal{L}_G(i, i + 1)$. Let v_1 and v_2 be the two nodes connected to e . Let u_i (resp. u_{i+1}) be the neighbor of level i (resp. $i + 1$) of the gadget g . In the following, we denote by $p_{(u, u')}$ the port number at node u of the edge $\{u, u'\}$ in G before applying the operation. The operation then returns a graph G' obtained after applying on G the following modifications in this order:

1. Remove the edges $e = \{v_1, v_2\}$, $\{u_i, g\}$, and $\{u_{i+1}, g\}$.
2. Add the edge $\{u_i, u_{i+1}\}$. Assign to this edge the port number $p_{(u_i, g)}$ (resp. $p_{(u_{i+1}, g)}$) at node u_i (resp. u_{i+1}).
3. Add the edge $\{v_1, g\}$. Assign to this edge the port number $p_{(v_1, v_2)}$ (resp. $p_{(g, u_i)}$) at node v_1 (resp. g).
4. Add the edge $\{v_2, g\}$. Assign to this edge the port number $p_{(v_2, v_1)}$ (resp. $p_{(g, u_{i+1})}$) at node v_2 (resp. g).

In other cases, the operation simply returns G without any modification. An illustration of `move-gadget` is shown in Figure 2.b. Broadly speaking, when `move-gadget` causes a graph modification, a green edge is “split” into two red edges, while two red edges are “merged” into a single green edge.

In view of the construction of $\mathcal{F}(\ell, w, r)$ and of the definitions of the three above operations, we have the following lemma.

Lemma 2.4 *Let $\ell \geq 2$, $w \geq 4$ and $r \geq 6$ be integers. Let G be a graph of $\mathcal{F}(\ell, w, r)$ and let G' be the graph returned after the application of `switch-ports`($G, *, *, *$), `switch-edges`($G, *, *, *, *$) or `move-gadget`($G, *, *$). G' belongs to $\mathcal{F}(\ell, w, r)$.*

3 Impossibility Result for Distance-constrained Exploration

This section is dedicated to our proof showing the impossibility of solving distance-constrained exploration with a penalty that is always linear in the order of the graph. In what follows, we first give the intuitive ingredients that are behind our proof, and then the formal demonstration.

3.1 Overview of the Proof

3.1.1 In a Nutshell

At a very high level, the proof can be viewed as composed of two main stages. The first stage consists in showing that for every positive real α , every integer $r \geq 6$, and every algorithm $A(\alpha, r)$ solving distance-constrained exploration for all instances of $\mathcal{I}(\alpha, r)$, there exists at least one instance $I = (G, 0, \alpha)$ such that $G = (V, E) \in \mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$ and for which algorithm A incurs a penalty of at least $\Omega(w^2)$ (for some arbitrarily large w). This result is achieved through a constructive proof in which an adversary, whose goal is to maximize the penalty, constructs “on the fly” the instance I based on the step-by-step choices of algorithm A . If w belonged to $\Theta(|V|)$, our work would essentially be completed here, as it would mean that algorithm A would sometimes incur a penalty of $\Omega(|V|^2)$. Unfortunately, using Lemma 2.2, we can show that w is in $\mathcal{O}(\sqrt{|V|})$, which prevents us from concluding directly. This is where the second stage comes into the picture. By taking a closer look at our first stage, we will see that a penalty of at least $\Omega(w^2)$ is already incurred by algorithm A with instance I before the agent makes the μ th edge traversal that leads it to explore a red edge of G for the first time. In particular, before this traversal, in each layer, only green edges have been explored and no gadget nodes has been visited. However, the number of nodes that are not gadgets in G is in $\Theta(r \times w)$ (cf. Lemma 2.2). Therefore, if we can transform instance I into an instance $I' = (G' = (V', E'), 0, \alpha)$ of $\mathcal{I}(\alpha, r)$ by “reducing” the number of gadget nodes from $\Theta(r \times w^2)$ (cf. Lemma 2.2) to $\Theta(w)$, while keeping the number of non-gadget nodes unchanged and ensuring that the execution of A with instance I is exactly the same as with I' (from the point of view of the agent’s memory) during the first μ edge traversals, we are done. Indeed, in this case, $|V'| \in \Theta(r \times w)$ and we can prove that algorithm A necessarily incurs at least a penalty of order $\left(\frac{|V'|}{r}\right)^2$ with instance I' . Hence, it suffices to choose w large enough so that r^2 is in $o(|V'|)$, to conclude that algorithm A cannot guarantee a linear penalty in the order of the graph. Actually, it is precisely this transformation that is conducted in the second stage. Notice that proceeding in two stages, instead of a single one (in which the adversary would construct instance I' directly), turns out to be more convenient because instance I , despite having too many nodes regarding the incurred penalty, is easier (from a pedagogical point of view) to construct dynamically than the instance I' that has the appropriate number of nodes.

3.1.2 Under the Hood

Having described the general outline of our proof, let us now go deeper into the details of our two stages, by starting to give more intuitive explanations about the first one and, in particular, the behavior of the adversary. Here, the goal of the adversary is to construct an instance of $\mathcal{I}(\alpha, r)$ that creates a bad scenario for the algorithm. What could be a bad scenario? It could be one in which the agent, starting from node 0 in a graph of $\mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$, frequently “goes down” until nodes at the penultimate level $\lfloor (1 + \alpha)r \rfloor$ while (1) there are still unexplored edges incident to these nodes that lead to the last level and (2) no red edge has been explored yet. Indeed, what ensures that nodes at the last level are within distance r from the source node are paths that pass exclusively through the critical node, which can only be accessed by traversing at least one red edge. Red edges and the critical node are essential components to make the eccentricity of the source node “small”. As long as no red edge is explored, the critical node remains unexplored, and thus the agent lacks information about the shortest paths within the underlying graph. In

particular, under these circumstances, if it reached a node of the last level $\lfloor (1 + \alpha)r \rfloor + 1$, the shortest path it would know to the source node would be of length at least $\lfloor (1 + \alpha)r \rfloor + 1$ (cf. Lemma 2.3), which would violate the distance constraint. Hence, in such a scenario, each time the agent reaches a node of the penultimate level with possible edges leading to nodes of the last level, the algorithm cannot risk instructing the agent to traverse an unexplored edge. Actually, it has no other choice but to ask the agent to go back to the previous level by traversing an already explored edge, thereby incrementing the incurred penalty. And if this occurs at least $\Omega(w^2)$ times, the adversary successfully completes the first stage.

Hence, the question that naturally comes to mind is how to construct a graph implying such a bad scenario. As briefly mentioned earlier, the adversary can achieve this by dynamically modifying the underlying graph according to the choices made by the algorithm. More precisely, the adversary initially takes any instance $(G_0, 0, \alpha)$ of $\mathcal{I}(\alpha, r)$ such that $G_0 \in \mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$. Then, it emulates algorithm $A(\alpha, r)$ with this instance, but by paying particular attention to what happens before each edge traversal and making some possible modifications before each of them, and only if coherent and appropriate. By “coherent”, we mean that a modification must not conflict with the prefix of the execution already made, and, in particular, with the agent’s memory. For example, the adversary must not remove an already explored edge or change the degree of an already visited node. And by “appropriate”, we mean that the modifications must be made only to enforce the bad scenario described above. Consequently, in the situation where the algorithm asks the agent to traverse an edge that has been explored before, the adversary does nothing, in order to preserve the coherence of the ongoing execution, while increasing the penalty. It also does nothing in the situation where the algorithm instructs the agent to take a port that makes it go from the source node (resp. a node of some level i) to a node of level 1 (resp. $i + 1$) that is incident to at least one unexplored green edge leading to a node of level 2 (resp. $i + 2$ if $i + 2 \leq \lfloor (1 + \alpha)r \rfloor + 1$), because this is essential for the bad scenario to unfold. In fact, if the adversary is lucky enough with its initial choice of G_0 to always get one of these situations (not necessarily the same) without intervening, during a sufficiently long prefix of the execution, we can inductively prove that the negative scenario occurs with G_0 . Unfortunately, the adversary may not be so lucky. However, it can circumvent this, and enforce the bad scenario, by bringing modifications to the underlying graph (formally described in Algorithm 2) using clever combinations of the three operations **switch-ports**, **move-gadget**, and **switch-edges** defined in Section 2. Each of these operations has a very specific role. Precisely, whenever possible, the adversary uses **switch-ports** to make the agent “go down”, **move-gadget** to force the agent to traverse a green edge instead of a red one, and **switch-edges** to ensure that a node reached by the agent still has unexplored edges leading downward (which is important to subsequently continue the descent process and eventually enforce the agent to traverse an already explored edge). These operations are performed carefully to always ensure the coherence of the execution, while ensuring that the underlying graph remains in $\mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$ (cf. Lemma 3.1). One of the main challenges of the first stage consists in showing that the adversary can indeed intervene, with these operations, for a sufficiently long time to ultimately guarantee the desired penalty of $\Omega(w^2)$ before the first visit of a red edge and thus of a gadget. In fact, the general idea of the proof revolves around the following three key points (demonstrated through Lemmas 3.2 and 3.3). First, the adversary can prevent the agent from visiting any red edge at least until half the green edges of some layer $\mathcal{L}(i, i + 1)$ have been explored at some time λ . Second, as long as this event has not occurred, after each descending traversal of a green edge of $\mathcal{L}(i, i + 1)$ (i.e., from level i to level $i + 1$), the agent will traverse an already explored edge before making

another descending traversal of a green edge of $\mathcal{L}(i, i+1)$. This will happen either by the “agent’s own choice” or because the adversary forces the agent to do so by bringing it to the penultimate level, as previously described. Hence, the incurred penalty is at least of the order of the number of descending traversals of green edges in $\mathcal{L}(i, i+1)$ made by time λ . Third and finally, as long as no red edge has been explored, the number of times the agent has made a descending traversal of a green edge in $\mathcal{L}(i, i+1)$ is always equal to the number of times the agent has made an ascending traversal of a green edge in $\mathcal{L}(i, i+1)$ (i.e., from level $i+1$ to level i), give or take one. From these three key points and the fact that there are $\Omega(w^2)$ green edges in $\mathcal{L}(i, i+1)$ (cf. Lemma 2.2), it follows that the number of times the agent makes a descending traversal of a green edge of this layer by time λ is in $\Omega(w^2)$, and so is the penalty.

To finish, let us now complete the overview with some intuitive explanations about the second stage. As stated before, the goal of this stage is to transform the instance $I = (G, 0, \alpha)$ obtained in the first stage into an instance $I' = (G', 0, \alpha)$ of $\mathcal{I}(\alpha, r)$ with only $\Theta(w)$ gadgets instead of $\Theta(r \times w^2)$. This reduction on the number of gadgets is done with merge operations that consist in “merging” some gadgets of G into one gadget in G' and setting the neighborhood of the new merged gadget as the union of the neighborhoods of the initial gadgets. An important property to check is that the source node of G' (i.e., the node with label 0) must have eccentricity r for I' to be a valid instance of $\mathcal{I}(\alpha, r)$. This is guaranteed by two facts. Firstly, merging nodes can only reduce distances between nodes and so the eccentricity of the source node in G' is at most the eccentricity r in G . Secondly, by the construction of G (cf. Section 2), all gadgets of G are at distance at least 2 from node v_1 (which plays here the role of the source node) and at distance at least $r-2$ from the tail tip u of L . Since each path from v_1 to u in G contains a gadget, each path in G' from v_1 to u contains a merged gadget that is at distance at least 2 from v_1 and at least $r-2$ from u . It follows that the eccentricity of the source node in G' is also at least r .

Another important property to check, with merge operations, is that the new graph G' must “look” the same from the point of view of non-gadget nodes contained in the layers of G , while being simple (i.e., without multiple edges). Indeed, this is also needed to ensure that I' is a valid instance of $\mathcal{I}(\alpha, r)$, while guaranteeing that the execution of A is the same in I and I' during the first μ edge traversals and thus the penalty up until the visit of the first gadget is the same in I and I' . Hence, we cannot use trivial merge operations such as simply merging all gadgets of each layer into one gadget, or otherwise we immediately get a multigraph. We need something more subtle. Actually, we can show that the above property can be satisfied if the neighborhoods of the merged gadgets are disjoint, notably because this can avoid creating a multigraph, while keeping the same port numbers at non-gadget nodes for all edges connecting them to gadget nodes.

As a result, one of the key parts of the second stage is finding $\Theta(w)$ sets of gadgets that each have gadgets with disjoint neighborhoods. Intuitively, one can proceed in two steps. First, we consider each layer separately. Let us consider some layer $\mathcal{L}_G(i, i+1)$ of a graph G . One associates each gadget in this layer to an edge connecting its two neighbors in the layer. Finding sets of gadgets with disjoint neighborhoods can thus be viewed as finding sets of non-adjacent edges in this new graph. Using the fact that the degree of nodes of level i and $i+1$ in the graph induced by the edges of $\mathcal{L}_G(i, i+1)$ is in $\Theta(w)$, one can find $\Theta(w)$ such sets using a well-known bound for edge-coloring of graphs of bounded degree (cf. König’s Theorem [17]). Hence, one can merge the $\Theta(w^2)$ gadgets in the layer into $\Theta(w)$ gadgets, each corresponding to a color in the edge-coloring. With this first step, the number of gadgets is down from $\Theta(r \times w^2)$ to $\Theta(r \times w)$. The second step uses the fact that two gadgets in non-consecutive layers do not have common neighbors. Hence,

one can partition the layers into odd and even layers such that gadgets in odd (resp. even) do not share neighbors. Then, one can take one merged gadget in each odd (resp. even) layer and merge all these gadgets into one. This permits to merge $\Theta(r)$ gadgets into one and by repeating this operation one can diminish the number of gadgets from $\Theta(r \times w)$ to $\Theta(w)$. This strategy, which is the crux of the second stage, is implemented in the proof of Theorem 3.1 and constitutes the finishing touch to show our impossibility result.

3.2 Formal Proof

Before going any further, we need to introduce some extra notations. Let α (resp. r) be any positive real (resp. any positive integer). Let $A(\alpha, r)$ be an algorithm solving distance-constrained exploration for all instances of $\mathcal{I}(\alpha, r)$ and let (G, l_s, α) be an instance of $\mathcal{I}(\alpha, r)$. We will denote by $\text{cost}(A, (G, l_s, \alpha))$ the number of edge traversals prescribed by $A(\alpha, r)$ with instance (G, l_s, α) . Moreover, for every integer $0 \leq i \leq \text{cost}(A, (G, l_s, \alpha))$, we will denote by $\mathbf{M}(A, (G, l_s, \alpha), i)$ (resp. $\mathbf{E}(A, (G, l_s, \alpha), i)$) the memory of the agent (resp. the set of edges that have been traversed by the agent) after the first i edge traversals prescribed by $A(\alpha, r)$ with instance (G, l_s, α) . We will also denote by $\overline{\mathbf{E}}(A, (G, l_s, \alpha), i)$ the set of edges of G that are not in $\mathbf{E}(A, (G, l_s, \alpha), i)$. Finally, for every integer $1 \leq i \leq \text{cost}(A, (G, l_s, \alpha))$, we will denote by $\text{node}(A, (G, l_s, \alpha), i)$ (resp. $\text{edge}(A, (G, l_s, \alpha), i)$) the node of G that is occupied after the i th edge traversal (resp. the edge of G that is explored during the i th edge traversal) prescribed by $A(\alpha, r)$ with instance (G, l_s, α) . By convention, the source node will be sometimes denoted by $\text{node}(A, (G, l_s, \alpha), 0)$.

Algorithm 1 gives the pseudocode of function **AdversaryBehavior**, which corresponds to the first stage outlined in Section 3.1. It takes as input the parameters r and α of the distance-constrained exploration problem, an integer w and an exploration algorithm A solving the problem for all instances of $\mathcal{I}(\alpha, r)$. Under certain conditions (see Lemma 3.3), the function returns a graph G with the following two properties. The first property is that $(G, 0, \alpha)$ is an instance of $\mathcal{I}(\alpha, r)$ and $G \in \mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$. The second property is that the penalty incurred by A on instance $(G, 0, \alpha)$, by the time of the first visit of a gadget, is in $\Omega(w^2)$ (i.e., the target penalty of the first stage). To achieve this, function **AdversaryBehavior** relies on function **GraphModification**, whose pseudocode is provided in Algorithm 2. Conceptually, **AdversaryBehavior** proceeds in steps $0, 1, 2, 3$, etc. In step 0 (cf. line 1 of Algorithm 1), we choose an arbitrary graph G_0 of $\mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$. In step $x \geq 1$ (cf. lines 3 to 7 of Algorithm 1), we begin with a graph G_{x-1} of $\mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$ for which the first $(x - 1)$ edge traversals prescribed by A from the node labeled 0 are “conformed” with the bad scenario of the first stage. The goal is to ensure that this conformity extends to the first x edge traversals as well. This is accomplished by deriving a graph G_x , which also belongs to $\mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$, from graph G_{x-1} . The derivation is handled by function **GraphModification** that uses as inputs the graph G_{x-1} , the index $x - 1$, the algorithm A and the parameter α . If $x < \text{cost}(A, (G_x, 0, \alpha))$, then there exists a $(x + 1)$ th edge traversal in the execution of A from the node labeled 0 in G_x , and thus we can continue the process with step $x + 1$. Otherwise, the algorithm stops and returns graph G_x .

When we mentioned above that in step $x \geq 1$, we aim at extending the bad scenario till the x th edge traversal included, we mean that **GraphModification** $(G_{x-1}, \alpha, A, x - 1)$ returns a graph G_x such that (1) $\mathbf{M}(A, (G_{x-1}, 0, \alpha), x - 1) = \mathbf{M}(A, (G_x, 0, \alpha), x - 1)$ (i.e., from the agent’s perspective, there is no difference between the first $x - 1$ edge traversals in G_{x-1} and those in G_x) and (2), if possible, the x th edge traversal in G_x brings the agent closer to the penultimate level (in order to eventually force incrementing the penalty). The first point is particularly important for at least

preserving the penalty already incurred, while the second is important for increasing it. However, while the first point will be always guaranteed by **GraphModification** (cf. Lemma 3.1), the second is not always useful or even achievable. This may happen when the agent is at a node of the last level or has previously explored a red edge (in which cases the goal of the bad scenario should have been already met, as explained in Section 3.1) or when the agent is located at the source node (in which case the next move, if any, is in line with the bad scenario as it can only make the agent go down to a node of level 1). This may also happen when the agent decides to recross an already explored edge as modifying the traversal would violate the agent's memory (but, this is not an issue, as it increases the penalty as intended). In all these situations (cf. the test of line 4 of Algorithm 2), no modifications will be made and $G_x = G_{x-1}$. (In the rest of the explanations, all line references are to Algorithm 2, which we will omit to mention for the sake of brevity).

Assume that the test at line 4 evaluates to true during the execution of **GraphModification**($G_{x-1}, \alpha, A, x - 1$) and denote by G'_{x-1} the graph obtained right after the (possible) modifications made to G_{x-1} by lines 5 to 10. When evaluating the test at line 4, the agent is thus located at some node u of some level $i < \ell = \lfloor (1 + \alpha)r \rfloor + 1$ after the first $x - 1$ edge traversals prescribed by A in G_{x-1} . At this point, we may face several possible situations, which will be all addressed in the formal proof. But here, one is perhaps more important to pinpoint than the others. It is when the following two conditions are satisfied right before the x th edge traversal in G_{x-1} . The first condition, call it $c1(u, i)$, is that node u is incident to at least one unexplored edge of $\mathcal{L}(i, i + 1)$ and the second condition, call it $c2(i)$, is that there is at least one unexplored green edge in $\mathcal{L}(i, i + 1)$. This situation is interesting because lines 5 to 10 guarantee that the x th edge traversal in G'_{x-1} will correspond to a traversal of a green edge leading the agent from node u (of level i) to a node u' of level $i + 1$, as intended in the bad scenario. Note that, if $i + 1 = \ell = \lfloor (1 + \alpha)r \rfloor + 1$ (i.e., the index of the last level in the graph) then algorithm A violates the distance constraint (in view of the fact that no red edge has been yet visited), which is a contradiction. Hence, we have $i < \ell - 1$ and, according to the bad scenario we aim at forcing the agent into, we want the $(x + 1)$ th move from node u' (of level $i + 1$) to lead the agent to level $i + 2$ or to be a traversal of a previously explored edge. In the situation where $c1(u', i + 1)$ and $c2(i + 1)$ are also satisfied right before the $(x + 1)$ th edge traversal in G'_{x-1} , the test at line 12 evaluates to false and $G'_{x-1} = G_x$. This is a good situation for the bad scenario. Indeed, if the $(x + 1)$ th move in G_x is not a traversal of a previously explored edge, then, in the next call to **GraphModification**, the test at line 4 evaluates to true and lines 5 to 10 will ensure that the $(x + 1)$ th edge traversal in G_x will correspond to a traversal of a green edge leading the agent to a node u'' of level $i + 2$.

But what if we are not in such a good situation, where both $c1(u', i + 1)$ and $c2(i + 1)$ are satisfied right before the $(x + 1)$ th edge traversal in G'_{x-1} ? In fact, if $c2(i + 1)$ is not satisfied, then there is at least one layer for which at least half the green edges has been explored, which means that the target penalty has already been paid (as explained in Section 3.1 and shown in the proof of Lemma 3.3). Hence, the tricky case is when $c2(i + 1)$ holds but $c1(u', i + 1)$ does not, right before the $(x + 1)$ th edge traversal in G'_{x-1} . This is exactly where lines 13 to 22 come into the picture (the test at line 12 then evaluates to true during the execution of **GraphModification**($G_{x-1}, \alpha, A, x - 1$)). Using function **switch-edges**, these lines attempt to reroute the x th edge traversal in G'_{x-1} towards a node v , different from u' but also of level $i + 1$, such that $c1(v, i + 1)$ holds right before the $(x + 1)$ th traversal in G_x . The feasibility of such rerouting will be guaranteed as long as some conditions are met, particularly that there is no layer for which at least half the green edges has been explored (cf. Lemma 3.2).

The switch performed at line 22 involves two green edges. One of these edges is naturally $\{u, u'\}$. The other is a green edge (from the same layer as $\{u, u'\}$) resulting from the merge of two red edges achieved by `move-gadget`, which is performed, right before the switch, at line 17. Proceeding in this way, rather than simply selecting an existing green edge, turns out to be essential to maximize the number of possible reroutings ensuring condition c1, and thus to get the desired penalty of $\Omega(w^2)$ before the first exploration of a red edge.

Algorithm 1: Function `AdversaryBehavior`(r, α, A, w)

```

/* Step 0 */
1  $G :=$  any graph of  $\mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$ ;  $x := 1$ ;
2 repeat
    /* Step x */
3    $G :=$  GraphModification( $G, \alpha, A, x - 1$ );
4   if  $x < \text{cost}(A, (G, 0, \alpha))$  then
5      $x := x + 1$ ;
6   else
7     return  $G$ ;

```

Algorithm 2: Function `GraphModification`(G, α, A, t)

```

1  $G' := G$ ;  $r :=$  the eccentricity of the node having label 0 in  $G'$ ;
2  $u := \text{node}(A, (G', 0, \alpha), t)$ ;  $e := \text{edge}(A, (G', 0, \alpha), t + 1)$ ;
3  $p_u :=$  the port of  $e$  at node  $u$ ;  $\ell :=$  the number of levels in  $G'$ ;
4 if there is no red edge in  $\mathbf{E}(A, (G', 0, \alpha), t)$  and  $e \notin \mathbf{E}(A, (G', 0, \alpha), t)$  and  $u$  is a node of some level
    $1 \leq i < \ell$  in  $G'$  then
5   if  $e \notin \mathcal{L}_{G'}(i, i + 1)$  and  $u$  is an endpoint of an edge  $f \in \mathcal{L}_{G'}(i, i + 1) \cap \overline{\mathbf{E}(A, (G', 0, \alpha), t)}$  then
6      $p'_u :=$  the port of edge  $f$  at node  $u$ ;  $G' := \text{switch-ports}(G', u, p_u, p'_u)$ ;
7      $e := \text{edge}(A, (G', 0, \alpha), t + 1)$ ;
8   if  $e$  is a red edge of some layer  $\mathcal{L}$  and there is a green edge  $k \in \mathcal{L} \cap \overline{\mathbf{E}(A, (G', 0, \alpha), t)}$  then
9      $u' := \text{node}(A, (G', 0, \alpha), t + 1)$ ;
10     $G' := \text{move-gadget}(G', k, u')$ ;  $e := \text{edge}(A, (G', 0, \alpha), t + 1)$ ;
11     $u' := \text{node}(A, (G', 0, \alpha), t + 1)$ ;
12    if  $i < \ell - 1$  and  $u'$  is a node of level  $i + 1$  in  $G'$  and  $u'$  is not an endpoint of an edge
        $\in \mathcal{L}_{G'}(i + 1, i + 2) \cap \overline{\mathbf{E}(A, (G', 0, \alpha), t)}$  then
13       $D :=$  the set made of the two endpoints of  $e$ , plus all gadgets in  $G'$  that are incident to a
          red edge of  $\mathcal{L}_{G'}(i, i + 1)$  and that are adjacent to at least one of the endpoints of  $e$ ;
14       $N_i :=$  the set of all nodes of level  $i$  in  $G'$  that are not adjacent to a node of  $D$ ;
15       $N_{i+1} :=$  the set of all nodes of level  $i + 1$  in  $G'$  that are not adjacent to a node of  $D$  and
          that are incident to at least one edge of  $\mathcal{L}_{G'}(i + 1, i + 2) \cap \overline{\mathbf{E}(A, (G', 0, \alpha), t)}$ ;
16      if there exist a green edge  $h \in (\mathcal{L}_{G'}(i, i + 1) \cap \overline{\mathbf{E}(A, (G', 0, \alpha), t)}) \setminus \{e\}$  and a gadget  $g$ 
          adjacent to a node of  $N_i$  and to a node of  $N_{i+1}$  then
17         $G' := \text{move-gadget}(G', h, g)$ ;
18         $k :=$  the green edge of  $G'$  created by the previous call to move-gadget;
19         $v :=$  the node of level  $i + 1$  that is incident to edge  $k$ ;
20         $p_v :=$  the port of edge  $k$  at node  $v$ ;
21         $p_{u'} :=$  the port at node  $u'$  of  $e$ ;
22         $G' := \text{switch-edges}(G', u', v, p_{u'}, p_v)$ ;
23 return  $G'$ ;

```

For the next two lemmas, we first observe that for every positive real α , every integer $r \geq 6$, every integer $w \geq 4$, and every $G \in \mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$, we have $(G, 0, \alpha) \in \mathcal{I}(\alpha, r)$, by Lemma 2.3.

Lemma 3.1 *Let α be any positive real. Let $w \geq 16$ and $r \geq 6$ be two integers. Let $A(\alpha, r)$ be an algorithm solving the distance-constrained exploration problem for every instance of $\mathcal{I}(\alpha, r)$. For every $G \in \mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$ and every integer t such that $0 \leq t < \text{cost}(A, (G, 0, \alpha))$, we let $G^{\text{new}} = \text{GraphModification}(G, \alpha, A, t)$ and we have the following three properties:*

1. $G^{\text{new}} \in \mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, w, r)$ and $(G^{\text{new}}, 0, \alpha)$ is an instance of $\mathcal{I}(\alpha, r)$;
2. During the execution of $\text{GraphModification}(G, \alpha, A, t)$, we have $M(A, (G, 0, \alpha), t) = M(A, (G', 0, \alpha), t)$ after each modification of the variable G' , consequently $M(A, (G, 0, \alpha), t) = M(A, (G^{\text{new}}, 0, \alpha), t)$;
3. If $\text{node}(A, (G, 0, \alpha), t)$ is a node of some level $1 \leq i \leq \lfloor (1 + \alpha)r \rfloor + 1$ in G , there is no red edge in $E(A, (G, 0, \alpha), t)$, and, for all $1 \leq j \leq \lfloor (1 + \alpha)r \rfloor$, there is a green edge in $\mathcal{L}_G(j, j + 1) \cap \overline{E}(A, (G, 0, \alpha), t)$, then $\text{node}(A, (G^{\text{new}}, 0, \alpha), t + 1)$ is not a gadget.

Proof. TOPROVE 1 □

Lemma 3.2 *Let α be any positive real. Let $k > 0$ and $r \geq 6$ be two integers. Let $A(\alpha, r)$ be an algorithm solving the distance-constrained exploration problem for every instance of $\mathcal{I}(\alpha, r)$. For every $G \in \mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, 16k, r)$ and every integer t such that $1 \leq t < \text{cost}(A, (G, 0, \alpha))$, if the following three conditions are satisfied:*

- $\text{node}(A, (G, 0, \alpha), t)$ is a node of some level $1 \leq i \leq \lfloor (1 + \alpha)r \rfloor$ corresponding to an endpoint of an edge of $\mathcal{L}_G(i, i + 1) \cap \overline{E}(A, (G, 0, \alpha), t)$ and
- there is no red edge in $E(A, (G, 0, \alpha), t)$ and
- for every layer \mathcal{L} of G , at least half the green edges of \mathcal{L} are not in $E(A, (G, 0, \alpha), t)$;

then we let $G^{\text{new}} = \text{GraphModification}(G, \alpha, A, t)$ and at least one of the following two properties holds:

- $i < \lfloor (1 + \alpha)r \rfloor$ and $\text{node}(A, (G^{\text{new}}, 0, \alpha), t + 1)$ is a node of level $i + 1$ corresponding to an endpoint of an edge of $\mathcal{L}_{G^{\text{new}}}(i + 1, i + 2) \cap \overline{E}(A, (G^{\text{new}}, 0, \alpha), t + 1)$;
- $\text{edge}(A, (G^{\text{new}}, 0, \alpha), t + 1) \in E(A, (G^{\text{new}}, 0, \alpha), t)$.

Proof. TOPROVE 2 □

Roughly speaking, the next lemma gives a lower bound on the penalty incurred by a distance-constrained exploration algorithm when running in some graphs of \mathcal{F} , by the time of the first visit of a gadget. Those graphs are determined by function **AdversaryBehavior**, the execution of which corresponds to the first stage of our overall strategy outlined in Section 3.1.

Lemma 3.3 *Let α be any positive real, $r \geq 6$ be any integer and $A(\alpha, r)$ be any algorithm solving distance-constrained exploration for all instances of $\mathcal{I}(\alpha, r)$. Let $G = \text{AdversaryBehavior}(r, \alpha, A, 16k)$, where k is any positive integer. We have the following two properties:*

1. $(G, 0, \alpha)$ is an instance of $\mathcal{I}(\alpha, r)$ such that $G \in \mathcal{F}(\lfloor (1 + \alpha)r \rfloor + 1, 16k, r)$;
2. The penalty incurred by A on instance $(G, 0, \alpha)$, by the time of the first visit of a gadget, is at least k^2 .

Proof. TOPROVE 3 □

We are now ready to prove the following theorem. The proof is based on a construction that corresponds to the second stage of our overall strategy outlined in Section 3.1.

Theorem 3.1 *Let α be any positive real and let $r \geq 6$ be any integer. Let $A(\alpha, r)$ be an algorithm solving distance-constrained exploration for all instances of $\mathcal{I}(\alpha, r)$. For every integer $k > 1$, there exists an instance $(G = (V, E), l_s, \alpha)$ of $\mathcal{I}(\alpha, r)$ such that $|V| \geq k$ and for which the penalty of A is at least $\left(\frac{|V|}{16(2\lfloor (1+\alpha)r \rfloor + 3)} \right)^2$.*

Proof. TOPROVE 4 □

Note that in the statement of Theorem 3.1, the instance $(G = (V, E), l_s, \alpha)$ can always be chosen so that $r^2 \in o(|V|)$. Therefore, we have the following corollary.

Corollary 3.1 *Let α be any positive real and let $r \geq 6$ be any integer. There exists no algorithm solving distance-constrained exploration for all instances $(G = (V, E), l_s, \alpha)$ of $\mathcal{I}(\alpha, r)$ with a linear penalty in $|V|$.*

4 Impossibility Result for Fuel-constrained Exploration

In this section, we observe that a linear penalty in $|V|$ cannot be obtained for the task of fuel-constrained exploration either. Actually, using much simpler arguments, we can quickly get a lower bound stronger than the one established for distance-constrained exploration in Theorem 3.1. As for the previous variant, the tuple $(G = (V, E), l_s, \alpha)$ can consistently define an instance of the fuel-constrained exploration problem. Hence, we can reuse it analogously, along with the set $\mathcal{I}(\alpha, r)$, in the statements of the following theorem and corollary dedicated to fuel-constrained exploration.

Theorem 4.1 *Let α be any positive real and let $r \geq 2$ be any integer such that $r\alpha \geq 1$. Let $A(\alpha, r)$ be an algorithm solving fuel-constrained exploration for all instances of $\mathcal{I}(\alpha, r)$. For every positive integer k , there exists an instance $(G = (V, E), l_s, \alpha)$ of $\mathcal{I}(\alpha, r)$ such that $|V| \geq k$ and for which the penalty of A is at least $\frac{|V|^2}{8\alpha}$.*

Proof. TOPROVE 5 □

From Theorem 4.1, we immediately get the following corollary.

Corollary 4.1 *Let α be any positive real and let $r \geq 2$ be any integer such that $r\alpha \geq 1$. There exists no algorithm solving fuel-constrained exploration for all instances $(G = (V, E), l_s, \alpha)$ of $\mathcal{I}(\alpha, r)$ with a linear penalty in $|V|$.*

5 Conclusion

In this paper, we addressed open problems posed by Duncan et al. in [14], who asked whether distance-constrained or fuel-constrained exploration could always be achieved with a penalty of $O(|V|)$. For each of these two exploration variants, we completely solved the open problem by providing a strong negative answer, which in turn highlights a clear difference with the task of unconstrained exploration that can always be conducted with a penalty of $\mathcal{O}(|V|)$ (cf. [19]).

It should be noted that we did not make any attempt at optimizing the lower bounds on the penalties: our only concern was to prove their nonlinearity with respect to $|V|$. However, for the case of fuel-constrained exploration, there is not much room for improvement: we showed that a penalty of $o(|V|^2)$ cannot be guaranteed, which is almost optimal as it is known that the problem can always be solved with a penalty of $O(|V|^2)$ (cf. [14]).

In contrast, such an observation cannot be made for the case of distance-constrained exploration. Indeed, the best-known algorithm for solving distance-constrained exploration can sometimes entail a penalty of $\Theta(|V|^2)$ (cf. [14]), while we showed that the penalty for this variant can sometimes be of order $\frac{|V|^2}{r^2}$. Hence, this leaves fully open the intriguing question of the optimal bound on the penalty for the task of distance-constrained exploration.

References

- [1] Susanne Albers and Monika Rauch Henzinger. Exploring unknown environments. *SIAM J. Comput.*, 29(4):1164–1188, 2000.
- [2] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, László Lovász, and Charles Rack-off. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 218–223. IEEE Computer Society, 1979.
- [3] Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. *Inf. Comput.*, 152(2):155–172, 1999.
- [4] Baruch Awerbuch and Stephen G. Kobourov. Polylogarithmic-overhead piecemeal graph exploration. In Peter L. Bartlett and Yishay Mansour, editors, *Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT 1998, Madison, Wisconsin, USA, July 24-26, 1998*, pages 280–286. ACM, 1998.
- [5] Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil P. Vadhan. The power of a pebble: Exploring and mapping directed graphs. *Inf. Comput.*, 176(1):1–21, 2002.
- [6] Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal learning of an unknown environment. *Mach. Learn.*, 18(2-3):231–254, 1995.
- [7] Manuel Blum and Dexter Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 132–142. IEEE Computer Society, 1978.
- [8] Lothar Budach. On the solution of the labyrinth problem for finite automata. *J. Inf. Process. Cybern.*, 11(10-12):661–672, 1975.

- [9] Jérémie Chalopin, Shantanu Das, and Adrian Kosowski. Constructing a map of an anonymous graph: Applications of universal sequences. In Chenyang Lu, Toshimitsu Masuzawa, and Mohamed Mosbah, editors, *Principles of Distributed Systems - 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings*, volume 6490 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2010.
- [10] Shantanu Das. Graph explorations with mobile agents. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 403–422. Springer, 2019.
- [11] Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. *J. Graph Theory*, 32(3):265–297, 1999.
- [12] Yoann Dieudonné and Andrzej Pelc. Deterministic network exploration by a single agent with byzantine tokens. *Inf. Process. Lett.*, 112(12):467–470, 2012.
- [13] Gregory Dudek, Michael Jenkin, Evangelos E. Milios, and David Wilkes. Robotic exploration as graph construction. *IEEE Trans. Robotics Autom.*, 7(6):859–865, 1991.
- [14] Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Trans. Algorithms*, 2(3):380–402, 2006.
- [15] Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theor. Comput. Sci.*, 345(2-3):331–344, 2005.
- [16] Frank Hoffmann. One pebble does not suffice to search plane labyrinths. In Ferenc Gécseg, editor, *Fundamentals of Computation Theory, FCT’81, Proceedings of the 1981 International FCT-Conference, Szeged, Hungary, August 24-28, 1981*, volume 117 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 1981.
- [17] Dénes König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77(4):453–465, 1916.
- [18] Michal Koucký. Universal traversal sequences with backtracking. *Journal of Computer and System Sciences*, 65(4):717–726, 2002. Special Issue on Complexity 2001.
- [19] Petrisor Panaite and Andrzej Pelc. Exploring unknown undirected graphs. *J. Algorithms*, 33(2):281–295, 1999.
- [20] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008.
- [21] Claude E. Shannon. Presentation of a maze-solving machine. In *8th Conference of the Josiah Macy Jr. Found. (Cybernetics)*, pages 173–180, 1951.