Label-Guided Graph Exploration by a Finite Automaton

Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, David Peleg

HAL Id: hal-00341609

https://hal.science/hal-00341609v1

Submitted on 25 Nov 2008

Label-Guided Graph Exploration
by a Finite Automaton

REUVEN COHEN
Boston University, USA
PIERRE FRAIGNIAUD
CNRS and Université Paris Diderot - Paris 7, France
DAVID ILCINKAS
CNRS and Université Bordeaux I, France.
AMOS KORMAN
CNRS and Université Paris Diderot - Paris 7, France
and
DAVID PELEG
The Weizmann Institute, Rehovot, Israel.

A finite automaton, simply referred to as a *robot*, has to explore a graph, i.e., visit all the nodes of the graph. The robot has no a priori knowledge of the topology of the graph or of its size. It is known that, for any k-state robot, there exists a graph of maximum degree 3 that the robot cannot explore. This paper considers the effects of allowing the system designer to add short labels to the graph nodes in a preprocessing stage, and using these labels to guide the exploration by the robot. We describe an exploration algorithm that given appropriate 2-bit labels (in fact, only 3-valued labels) allows a robot to explore all graphs. Furthermore, we describe a suitable labeling algorithm for generating the required labels, in linear time. We also show how to modify our labeling scheme so that a robot can explore all graphs of bounded degree, given appropriate 1-bit labels. In other words, although there is no robot able to explore all graphs of maximum degree 3, there is a robot $\mathcal{R}$, and a way to color in black or white the nodes of any bounded-degree graph G, so that $\mathcal{R}$ can explore the colored graph G. Finally, we give impossibility results regarding graph exploration by a robot with no internal memory (i.e., a single state automaton).

Categories and Subject Descriptors: C.2.1 [**Computer-communication networks**]: Network Architecture and Design; C.2.2 [**Computer-communication networks**]: Network Protocols; C.2.4 [**Computer-communication networks**]: Distributed Systems; E.1 [**Data Structures**]:

---

1. INTRODUCTION

1.1 Background:

This paper concerns graph exploration by finite automata. Let $\mathcal{R}$ be a finite automaton, simply referred to in this context as a *robot*, moving in an unknown undirected graph $G = (V, E)$. The robot has no a priori information about the topology of G and its size. To allow the robot $\mathcal{R}$ to distinguish between the different edges of a node u while visiting it, the $d = \deg(u)$ edges incident to u are associated with d distinct *port numbers* in $\{0, \ldots, d-1\}$, in a one-to-one manner. The port numbering is given as part of the input graph, and the robot has no a priori information about it. For convenience of terminology, we henceforth refer to the edge incident to port number l at node u simply as "edge l of u". (Clearly, if this edge connects u to v, then it may also be referred to as "edge l' of v" for the appropriate l'.) The robot has a finite number of states and a transition function f. If $\mathcal{R}$ enters a node of degree d through port i in state s, then it switches to state s' and exits the node through port i', where $(s', i') = f(s, i, d)$. The objective of the robot is to *explore* the graph, i.e., to visit all its nodes. The *size* of the robot's memory equals the logarithm of the number of states.

The first known algorithm designed for graph exploration was introduced by Shannon [Shannon, 1951]. Since then, several papers have been dedicated to the feasibility of graph exploration by a finite automaton. Rabin [Rabin 1967] conjectured that no finite automaton with a finite number of indistinguishable pebbles can explore all graphs (a *pebble* is a marker that can be dropped at and removed from nodes). The first step towards a formal proof of Rabin's conjecture is generally attributed to Budach [Budach, 1978], for a robot without pebbles. Blum and Kozen [Blum and Kozen, 1978] improved Budach's result by proving that a robot with three pebbles cannot perform exploration of all graphs. Kozen [Kozen, 1979] proved that a robot with four pebbles cannot explore all graphs. Finally, Rollik [Rollik 1980] gave a complete proof of Rabin's conjecture, showing that no robot with a finite number of pebbles can explore all graphs. The result holds even when restricted to planar 3-regular graphs. Without pebbles, it was proved [Fraigniaud et al., 2005] that a robot needs $\Theta(D \log \Delta)$ bits of memory for exploring all graphs of diameter D and maximum degree Δ. On the other hand, if the class of input graphs is restricted to trees, then exploration is possible even by a robot with no memory (i.e., zero states), simply by depth-first search (DFS) using the transition function $f(i, d) = i + 1 \bmod d$ (see, e.g., [Diks et al., 2004]).

Two variants of the exploration problem have been addressed in the literature: exploration with stop (in which the robot has to complete exploration and eventually stop) and exploration with return (in which the robot has to perform exploration and return to its starting position). It is shown in [Diks et al., 2004] that exploration

with stop of n-node trees requires a robot with memory size $\Omega(\log\log\log n)$ bits, and in [Gasieniec et al., 2007] that exploration with return of n-node trees can be achieved by a robot with $O(\log n)$ memory bits. It is impossible to explore arbitrary graphs with stop by a single robot if no marking of nodes is allowed. In [Fraigniaud et al., 2005], the authors show that also using one pebble, the exploration with stop of n-node graphs requires a robot with memory size $\Omega(\log n)$ bits.

1.2 Label-guided exploration:

The ability of dropping and removing pebbles at nodes can be viewed alternatively as the ability of the robot to dynamically *label* the nodes. If the robot is given k pebbles, then at any time during the exploration, $\sum_{u\in V}|l_u| \leq k$, where l_u is the label of node u and $|l_u|$ denotes the size of the label in unary.

In certain settings, it is expected that the graph will be visited by many exploring robots, and consequently, it is desired to preprocess the graph by leaving *fixed* (preferably small) road-signs, or *labels*, that will aid the robots in their exploration task. As a possible scenario, one may consider a network system where finite automata are used for traversing the system and distributing information in a sequential manner. Recently, a number of papers studied the affects of assigning (short) labels in a preprocessing stage, in order to ease a distributed task (e.g., [Korman et al., 2005; Fraigniaud et al., 2006; Fraigniaud et al., 2007]). This paper considers the affects of allowing the system designer to assign labels to the nodes in a preprocessing stage, and using these labels to guide robot explorations. The transition function f is augmented to utilize labels as follows. If the robot $\mathcal{R}$ in state s enters a node of degree d, labeled by l, through port i, then it switches to state s' and exits the node through port i', where

$$(s', i') = f(s, i, d, l).$$

This model can be considered stronger than Rabin's pebble model since labels are given in a preprocessing stage, but it can also be considered weaker since, once assigned to nodes, the labels cannot be modified.

Remark. It should be noted that the general function f presented here may depend on d, i and l, all of which may require $\log n$ bits to be described in an n-node graph. However, our formalism fits in the standard context of Mealy automata, i.e., the number of states of the robot is independent from the sizes of the inputs and outputs of the transition function. In contrast, for Moore automata, the output of the transition function is a state, in which the output port number is encoded, and thus the number of states of a Moore automaton is at least the degree of the graph. Nevertheless, the algorithm described in this paper actually deals with only three possible outputs: $i' = i$, $i' = i + 1 \bmod d$, and $i' = i - 1 \bmod d$, which can be interpreted as "go back", "go clockwise", and "go counterclockwise" commands, respectively. Therefore, the algorithm can be executed by a Moore automaton in contexts for which a natural spacial orientation does exist, as, e.g., for robots moving in a physical environment.

We address the design of *exploration labeling schemes*. Such schemes consist of a pair $(\mathcal{L}, \mathcal{R})$ such that, given any graph G with any port numbering, the algorithm $\mathcal{L}$ labels the nodes of G, and the robot $\mathcal{R}$ explores G with the help of the labeling produced by $\mathcal{L}$. In particular, we are interested in exploration labeling schemes for

which: (1) the preprocessing time required to label the nodes is polynomial, (2) the labels are short, and (3) the exploration is completed after a small number of edge traversals.

1.3 Our results:

As a consequence of Rollik's result, any exploration labeling scheme must use at least *two* different labels. Our main result states that just *three* labels (e.g., three colors) are sufficient for enabling a robot (with constant-size memory) to explore all graphs. Moreover, we show that our labeling scheme gives to the robot the power to stop once exploration is completed, although, in the general setting of graph exploration, the robot is not required to stop once the exploration has been completed, i.e., once all nodes have been visited. In fact, we show that exploration is completed in time $O(m)$, i.e., after $O(m)$ edge traversals, in any m-edge graph. In addition, we prove that the node-labeling can be done online while the robot is exploring the graph. The completion time of the algorithm then becomes $O(Dm)$, where D is the diameter of the graph.

For the class of bounded degree graphs, we design an exploration scheme using even smaller labels. More precisely, we show that just *two* labels (i.e., 1-bit labels) are sufficient for enabling a robot to explore all bounded degree graphs. The robot is however required to have a memory of size $O(\log \Delta)$ to explore all graphs of maximum degree Δ. The completion time $O(\Delta^{O(1)}m)$ of the exploration is larger than the one of our previous 2-bit labeling scheme, nevertheless it remains polynomial.

All these results are summarized in Table I. The two mentioned labeling schemes require polynomial preprocessing time.

Label size (#bits)	Robot's memory (#bits)	Time (# edge traversals)
2	$O(1)$	$O(m)$
1	$O(\log \Delta)$	$O(\Delta^{O(1)}m)$

Table I.   Summary of main results.

We also prove several impossibility results for 1-state (i.e., oblivious) robots. The behavior of 1-state robots depends solely on the input port number, and on the degree and label of the current node. In particular, we prove that for any $d > 4$ and for any 1-state robot using at most $\lfloor \log d \rfloor - 2$ colors, there exists a simple graph of maximum degree d that cannot be explored by the robot. This lower bound on the number of colors needed for exploration can be increased exponentially to $d/2 - 1$ by allowing loops.

2. A 2-BIT EXPLORATION-LABELING SCHEME

2.1 Exploration with pre-labeling

In this section, we describe an exploration-labeling scheme using only 2-bit (actually, 3-valued) labels. More precisely, we prove the following.

THEOREM 2.1. *There exists a robot with less than* 25 *states satisfaying the property that for any m-edge graph G, it is possible to color the nodes of G with three*

colors (or alternatively, assign each node a 2-bit label) so that using the labeling, the robot can explore the graph G, starting from any given node and terminating after identifying that the entire graph has been traversed. Moreover, the total number of edge traversals by the robot is at most $20m$.

To prove Theorem 2.1, we first describe the labeling scheme $\mathcal{L}$ and then the exploration algorithm. The node labeling is in fact very simple, and uses three colors denoted WHITE, BLACK, and RED. Let D denote the diameter of the graph.

Labeling $\mathcal{L}$.. Pick an arbitrary node r to be the *root* of the labeling $\mathcal{L}$. Nodes at distance d from r, $0 \leq d \leq D$, are labeled WHITE if $d \bmod 3 = 0$, BLACK if $d \bmod 3 = 1$, and RED if $d \bmod 3 = 2$.

The neighbor set $\mathcal{N}(u)$ of each node u can be partitioned into three disjoint sets: (1) the set $\mathsf{pred}(u)$ of neighbors closer to r than u; (2) the set $\mathsf{succ}(u)$ of neighbors farther from r than u; (3) the set $\mathsf{sibling}(u)$ of neighbors at the same distance from r as u. We also identify the following two special subsets of neighbors:

—$\mathsf{parent}(u)$ is the node $v \in \mathsf{pred}(u)$ such that the edge $\{u, v\}$ has the smallest port number at u among all edges connecting it to a node in $\mathsf{pred}(u)$.

—$\mathsf{child}(u)$ is the set of nodes $v \in \mathsf{succ}(u)$ such that $\mathsf{parent}(v) = u$.

For the root, set $\mathsf{parent}(r) = \emptyset$. The exploration algorithm is partially based on the following observations.

(1) For the root r, $\mathsf{child}(r) = \mathsf{succ}(r) = \mathcal{N}(r)$.

(2) For every node u with label $\mathcal{L}(u)$, and for every neighbor $v \in \mathcal{N}(u)$, the label $\mathcal{L}(v)$ uniquely determines whether v belongs to $\mathsf{pred}(u)$, $\mathsf{succ}(u)$ or $\mathsf{sibling}(u)$.

(3) Once at node u, a robot can identify $\mathsf{parent}(u)$ by visiting its neighbors successively, starting with the neighbor connected to port 0, then port 1, and so on. Indeed, by observation 2, the nodes in $\mathsf{pred}(u)$ can be identified by their label. The order in which the robot visits the neighbors ensures that $\mathsf{parent}(u)$ is the first visited node in $\mathsf{pred}(u)$.

Observe that one of the main difficulties of graph exploration by a robot with a finite memory is that the robot entering some node u by port p, and aiming at exiting u by the same port p after having performed some local exploration around u, does not have enough memory to store the value of p.

The Robot $\mathcal{R}$.. The overall exploration performed by our algorithm is the DFS of the spanning tree defined by the parent function (and child function). However, it is not always possible to know if an edge $\{u, v\}$ belongs to this spanning tree, based solely on the colors of u and v. To solve this problem, our exploration algorithm uses a procedure called `Check_Edge`. This procedure is specified as follows. When `Check_Edge`(j) is initiated at some node u, the robot starts visiting the neighbors of u one by one, and eventually returns to u by the same edge j reporting one of three possible outcomes: "child", "parent", or "false". These values have the following interpretation:

(i). if "child" is returned, then edge j at u leads to a child of u;

(ii). if "parent" is returned, then edge j at u leads to the parent of u;

(iii). if "false" is returned, then edge j at u leads to a node in $\mathcal{N}(u) \setminus (\mathsf{parent}(u) \cup \mathsf{child}(u))$.

The implementation of Procedure `Check_Edge` will be described later. Meanwhile, let us describe how the algorithm makes use of this procedure to perform exploration. In this description, all arithmetic operations are modulo d.

Assume that the robot $\mathcal{R}$ is initially at the root r of the 3-coloring $\mathcal{L}$ of the nodes. $\mathcal{R}$ leaves r by port number 0, in state DOWN. Note that, by the above observations, the node at the other endpoint of edge 0 of r is a child of r.

Now assume that $\mathcal{R}$ enters a node u of degree d via port number i. If $\mathcal{R}$ is in state DOWN, then it aims at identifying a child of u if one exists, or to backtrack along edge i of u if none exists. If $\mathcal{R}$ is in state UP, then it aims at identifying a child of u with port number $j \in \{i+1, \ldots, p-1\}$ if one exists (where p is the port number of the edge leading to $\mathsf{parent}(u)$), or to carry on moving up to the parent of u if there is no such child. In both cases, $\mathcal{R}$ achieves its goal as follows. $\mathcal{R}$ executes Procedure `Check_Edge`(j) for every port number $j = i+1, i+2, \ldots$ until the procedure eventually returns "child" or "parent" for some port number j. $\mathcal{R}$ then sets its state to DOWN in the former case and UP in the latter, and leaves u by port j.

If the robot does not start from the root r of the labeling $\mathcal{L}$, then it first goes to r by using Procedure `Check_Edge` to identify the parent of every intermediate node, and by identifying r as the only node with $\mathsf{pred}(r) = \emptyset$.

Moreover, the robot can stop after the exploration has been completed. More precisely, this can be done by introducing a slight modification of the robot behavior when it enters a node u of degree d via port number $d-1$ in state UP. In this case, $\mathcal{R}$ first checks whether u has a parent. If yes, then it acts as previously stated ($\mathcal{R}$ does not need to store d since d is the node degree). If not, the robot terminates the exploration.

Procedure `Check_Edge`*..* We now describe the actions of the robot $\mathcal{R}$ when Procedure `Check_Edge`(j) is initiated at a node u. The objective of $\mathcal{R}$ is to set the value of the variable `edge` to one of {parent, child, false}. We denote by v the other endpoint of the edge e with port number j at u. First, $\mathcal{R}$ moves to v in state "check_edge", carrying with it the color of node u. Let i be the port number of edge e at v. There are three cases to be considered.

(a) $v \in \mathsf{sibling}(u)$*:.* Then $\mathcal{R}$ backtracks through port i and reports "`edge` = false".

(b) $v \in \mathsf{pred}(u)$*:.* Then $\mathcal{R}$ aims at checking whether v is the parent of u, that is, whether u is a child of v. For that purpose, $\mathcal{R}$ moves back to u, and proceeds as follows: $\mathcal{R}$ successively visits edges $j-1, j-2, \ldots$ of u until either the other endpoint of the edge belongs to $\mathsf{pred}(u)$, or all edges $j-1, j-2, \ldots, 0$ have been visited. $\mathcal{R}$ then sets "`edge`=false" in the former case and "`edge`=parent" in the latter. At this point, let k be the port number at u of the last edge visited by $\mathcal{R}$. Then $\mathcal{R}$ successively visits edges $k+1, k+2, \cdots$ until the other endpoint belongs to $\mathsf{pred}(u)$. Then it moves back to u and reports the value of `edge`.

(c) $v \in \mathsf{succ}(u)$*:.* Then $\mathcal{R}$ aims at checking whether u is the parent of v. For that purpose, $\mathcal{R}$ proceeds in a way similar to Case (b), i.e., it successively visits edges $i-1, i-2, \ldots$ of v until either the other endpoint of the edge belongs to $\mathsf{pred}(v)$, or

all edges $i-1, i-2, \ldots, 0$ have been visited. $\mathcal{R}$ then sets its variable edge to "false" in the former case and to "child" in the latter. At this point of the exploration, let k denote the port number of the last edge incident to v that $\mathcal{R}$ visited. Then $\mathcal{R}$ successively visits edges $k+1, k+2, \ldots$ until the other endpoint w of the edge belongs to pred(v). Then it moves to w and reports the value of edge.

This completes the description of our exploration procedure.

Proof of Theorem 2.1:

Clearly, all nodes of an m-edge graph can be labelled by $\mathcal{L}$, using two bits per label, in time linear in m. It remains to prove the correctness of the exploration algorithm.

It is easy to check that if Procedure Check_Edge satisfies its specifications, then the robot $\mathcal{R}$ essentially performs a DFS traversal of the graph using edges $\{u, v\}$ where $u = $ parent(v) or $u \in$ child(v). Thus, we focus on the correctness of Procedure Check_Edge(j) initiated at node u. Let v be the other endpoint of the edge e with port number j at u, and let i be the port number of edge e at v. We check separately the three cases considered in the description of the procedure. By the previous observations, comparing the color of the current node v with the color of u allows $\mathcal{R}$ to distinguish between these cases.

If $v \in$ sibling(u), then v is neither a parent nor a child of u, and thus reporting "false" is correct. Indeed, $\mathcal{R}$ then backtracks to u via port i, as specified in Case (a).

If $v \in$ pred(u), then $v = $ parent(u) iff for every neighbor w_k connected to u by an edge with port number $k \in \{j-1, j-2, \ldots, 0\}$, $w_k \notin$ pred(u). The robot does check this property in Case (b) of the description, by returning to u, and visiting all the w_k's while none of them is found in pred(u). Moreover, the robot ends the procedure just after returning to u by the edge j at u. Hence, Procedure Check_Edge performs correctly in this case.

Finally, if $v \in$ succ(u), then $v = $ child(u) iff for every neighbor z_l connected to v by an edge with port number $l \in \{i-1, i-2, \ldots, 0\}$, $z_l \notin$ pred(v). In Case (c), the robot does check this property by visiting all these neighbors z_l while none of them is found in pred(v). At this point, it remains for $\mathcal{R}$ to return to u (obviously, the port number leading from v to u cannot be stored in the robot's memory since it has only a constant number of states). Let k be the port number of the last edge incident to v that $\mathcal{R}$ visited before setting its variable edge to "false" or "child". We have $0 \le k \le i-1$, $z_l \notin$ pred(v) for all $l \in \{k+1, \ldots, i-1\}$, and $u \in$ pred(v). Thus u is identified as the first predecessor that is met when visiting all v's neighbors by successively traversing edges $k+1, k+2, \ldots$ of v. This is precisely what $\mathcal{R}$ does according to the description of the procedure in Case (c). Hence, Procedure Check_Edge performs correctly in this case.

In summary, Procedure Check_Edge performs correctly in all cases and so does the global exploration algorithm. It remains to compute the number of edge traversals performed by the robot during the exploration (including the calls to Check_Edge).

We use again the same notations as in the description and in the proof of correctness of Procedure Check_Edge. Let us consider the Procedure Check_Edge(j) initiated at node u. Let v be other endpoint of the edge e with port number j at u, and let i be the port number of edge e at v. First observe that during the execution
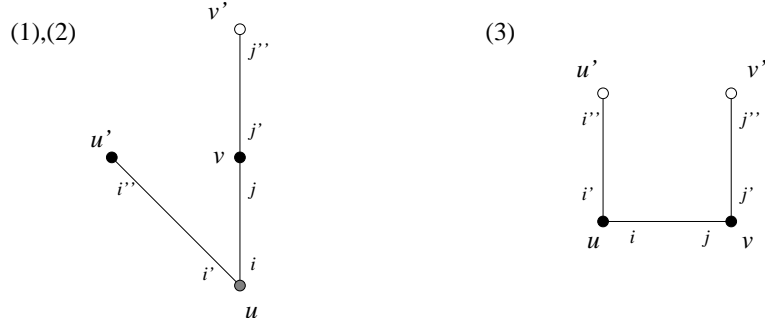
Fig. 1.   Notations for case (1), (2) and (3) of the analysis

of the Procedure Check_Edge only edges incident to u and v are traversed. More precisely:

Case (a):. $v \in \mathsf{sibling}(u)$. Then edge $e = \{u, v\}$ is traversed twice and no other edges are traversed during this execution of Procedure Check_Edge.

Case (b):. $v \in \mathsf{pred}(u)$. Then $\mathcal{R}$ traverses only edges incident to u. Let k be the greatest port number of the edges leading to a node in $\mathsf{pred}(u)$ and satisfying $k < j$. If it does not exist, set $k = 0$. $\mathcal{R}$ explores twice each edge $j, j - 1, \ldots, k + 1$ of u, then twice edge k, and finally again twice edges $k + 1, \ldots, j - 1, j$. To summarize, edge k of u is explored twice, and edges $k + 1, \ldots, j - 1, j$ of u are explored four times.

Case (c):. $v \in \mathsf{succ}(u)$. Then $\mathcal{R}$ traverses only edges incident to v. Let k be the greatest port number of the edges leading to a node in $\mathsf{pred}(v)$ and satisfying $k < i$. If it does not exist, set $k = 0$. $\mathcal{R}$ explores once edge j of u, twice each edge $i - 1, i - 2, \ldots, k + 1$ of v, twice edge k, twice again edges $k + 1, \ldots, i - 2, i - 1$, and finally once edge i of v (i.e., j of u). To summarize, edge i of u and edge k of v are explored twice and edges $k + 1, \ldots, i - 2, i - 1$ of v are explored four times.

We bound now the number of times each edge e of the graph is traversed. Edge $e = \{u, v\}$ is labeled i at u and j at v. Let us consider different cases (see Figure 1):

(1). $e = \{u, v\}$ with $v = \mathsf{parent}(u)$. The edge e is in the spanning tree, and thus is explored twice outside any execution of the Procedure Check_Edge. During Procedure Check_Edge(j) at v, edge e is explored twice. e is also explored four times during Check_Edge(i) at u, except if $i = 0$ where e is only explored twice during Check_Edge(i) at u. If there exists an edge $\{u', u\}$ labeled i' at u and i'' at u' such that $i' < i$ and $u' \in \mathsf{pred}(u)$, then edge e is explored twice during Procedure Check_Edge(i') at u and twice again during Procedure Check_Edge(i'') at u'. If there exists an edge $\{v', v\}$ labeled j' at v and j'' at v' such that $j' < j$ and $v' \in \mathsf{pred}(v)$, then edge e is explored four times during Procedure Check_Edge(j') at v and four times again during Procedure Check_Edge(j'') at v'. To summarize, edge e is explored at most 20 times during a DFS.

(2). $e = \{u, v\}$ with $v \in \mathsf{pred}(u)$ but $v \neq \mathsf{parent}(u)$. During Procedure Check_Edge(j) at v, edge e is explored twice. e is also explored four times during Check_Edge(i) at u. If there exists an edge $\{u', u\}$ labeled i' at u and i'' at u' such that $i' < i$

and $u' \in \mathsf{pred}(u)$, then edge e is explored twice during Procedure $\mathtt{Check\_Edge}(i')$ at u and twice again during Procedure $\mathtt{Check\_Edge}(i'')$ at u'. If there exists an edge $\{v', v\}$ labeled j' at v and j'' at v' such that $j' < j$ and $v' \in \mathsf{pred}(v)$, then edge e is explored four times during Procedure $\mathtt{Check\_Edge}(j')$ at v and four times again during Procedure $\mathtt{Check\_Edge}(j'')$ at v'. To summarize, edge e is explored at most 18 times during a DFS.

(3). $e = \{u, v\}$ with $v \in \mathsf{sibling}(u)$. During Procedure $\mathtt{Check\_Edge}(j)$ at v, edge e is explored twice. e is also explored twice during $\mathtt{Check\_Edge}(i)$ at u. If there exists an edge $\{u', u\}$ labeled i' at u and i'' at u' such that $i' < i$ and $u' \in \mathsf{pred}(u)$, then edge e is explored four times during Procedure $\mathtt{Check\_Edge}(i')$ at u and four times again during Procedure $\mathtt{Check\_Edge}(i'')$ at u'. If there exists an edge $\{v', v\}$ labeled j' at v and j'' at v' such that $j' < j$ and $v' \in \mathsf{pred}(v)$, then edge e is explored four times during Procedure $\mathtt{Check\_Edge}(j')$ at v and four times again during Procedure $\mathtt{Check\_Edge}(j'')$ at v'. To summarize, edge e is explored at most 20 times during a DFS.

Therefore, our exploration algorithm completes exploration in time at most $20m$ where m is the number of edges in the graph G. ∎

2.2 Exploration while labeling

The above description assumes that the labeling of the graph was conducted by an external entity prior to the exploration process by the robot. As an alternative, we present here a minor change to the exploration algorithm, allowing the robot to color the graph while conducting the exploration process.

For simplicity, we assume that all nodes begin at an uncolored "blank" state, and that the robot can identify such blank nodes. Alternatively, one can assume that blank represents a forth color, making this a full 2-bit scheme. For the coloring process, the robot needs three extra bits of memory: a flag $\mathtt{have\_colored}$ and a 2-bit variable $\mathtt{next\_color}$.

The exploration is performed in successive phases. The objective of phase $i \geq 0$ is to color all nodes at distance i from r. More precisely, letting G_i be the subgraph of the explored graph G induced by all nodes at distance at most i from the root r, the specification $P(i)$ of phase i states that the robot starts phase i from the root, traverses all the edges of G_i, colors all blank nodes by the color $\mathtt{next\_color}$ of the phase, and returns to the root.

At the beginning of the exploration (phase 0), the robot colors its current location WHITE, sets $\mathtt{next\_color}$ to BLACK, sets the flag $\mathtt{have\_colored}$ to false, and proceeds to phase 1.

For $i \geq 1$, the robot's exploration in phase i is done as described in Subsection 2.1, with the restriction that procedure $\mathtt{Check\_Edge}$ is modified as follows. Assume $\mathtt{Check\_Edge}$ is called at node u for some edge e incident to u. The procedure proceeds the same as before, except that whenever the robot traverses an edge $\{x, y\}$ from a colored node x to a blank node y, the following rules are applied.

—If the color of x is different from $\mathtt{next\_color}$, then y is colored by $\mathtt{next\_color}$, and the procedure carries on the same way as if y was already given the color $\mathtt{next\_color}$ when the robot entered it.

—Otherwise (i.e., the color of x is `next_color`), the robot returns to x and carries on just as if the edge $\{x, y\}$ did not exist.

Whenever a node is colored `next_color`, the flag `have_colored` is set to true.

For $i \geq 1$, in order to switch from phase i to phase $i + 1$, the robot does the following. Whenever the exploration should have stopped in the original algorithm, the robot now checks if the flag `have_colored` is set to true, and if so, clears it, sets `next_color` ← `succ(next_color)`, and restarts the exploration (phase $i + 1$ starts). Otherwise, it stops.

THEOREM 2.2. *By the end of the execution of the modified algorithm, the graph is fully colored and the robot has explored the entire graph, terminating at the root.*

Proof: We have to show that the algorithm meets its specification $P(i)$ for every $i \geq 0$. Define the *layer i* of G as the set of nodes at distance exactly i from r. For each $i \geq 0$ and time t, we say that $Property(i)$ holds at time t if the specification $P(i)$ holds and the following holds at time t.
1) All nodes of G_i are colored properly, i.e., according to the labeling $\mathcal{L}$ of the previous section,
2) Only nodes of G_i are colored,
3) For $i < D$, `next_color` is the color given by the labeling $\mathcal{L}$ to the nodes of layer $i + 1$.

We now prove that, at the end of phase i, $Property(i)$ holds. At the end of phase 0, $Property(0)$ holds by construction. For $i \geq 1$, assume that at the end of phase $i - 1$, $Property(i - 1)$ holds and let us prove that $Property(i)$ holds at the end of phase i.

By the induction hypothesis, during phase i, all nodes of G_{i-1} are colored properly, i.e., according to the labeling $\mathcal{L}$ of the previous section.

Whenever the robot colors a node u during phase i, it has arrived from a node v that is colored and whose color is not `next_color`. Therefore, v belongs to layer $i - 1$, and u belongs to layer i. By the induction hypothesis, u is colored by the proper color `next_color`.

By the modification of procedure `Check_Edge`, whenever a non-colored node of layer i is visited by the robot coming from another node of layer i, the robot backtracks immediately. Therefore, every time the robot reaches a node u of layer $i + 1$, it comes from a colored node v of layer i. Since v is properly colored `next_color`, u is not colored, and the robot backtracks, ignoring the edge $\{u, v\}$. Informally, this means that the behavior of the robot in G at phase i is the same as its behavior in the graph G_i at the same phase i. In fact, there is only one difference, that occurs between siblings at layer i. In this situation, the modified procedure `Check_Edge` backtracks and may ignore the edge. However, this has no impact on the exploration since anyway, even if all the nodes are colored a priori, the robot always backtracks when visiting a sibling.

By Theorem 2.1, during phase i, all the edges of G_i are traversed, and at the end of the phase, the robot returns to the root. In fact, let T be the spanning tree of the explored graph G as induced by the labeling $\mathcal{L}$ of the previous section, and for $i \geq 0$ let T_i be the subtree of T spanning all nodes at distance at most i from the root r. The exploration proceeds as a DFS traversal of T_i, thus all nodes of

layer i are visited from a node of layer $i - 1$, and hence are properly colored with `next_color`. Hence specification $P(i)$ holds.

When phase i is finished, `next_color` is changed to `succ(next_color)`. Therefore, if $i < D$, at the end of phase i, the new `next_color` is the color given by the labeling $\mathcal{L}$ to the nodes of layer $i + 1$.

Therefore, for each $i \geq 0$, $Property(i)$ holds at the end of phase i, and in particular, the specification $P(i)$ holds. It follows that after D phases (where D is the diameter of the graph), the robot has completed coloring all nodes of the graph and has explored the entire graph. Nevertheless, another phase is performed, in which the robot discovers that the exploration and the coloring are actually completed. ∎

3. A 1-BIT EXPLORATION-LABELING SCHEME FOR BOUNDED DEGREE GRAPHS

In this section, we describe an exploration labeling scheme using only 1-bit labels. This scheme requires a robot with $O(\log \Delta)$ bits of memory for the exploration of graphs of maximum degree Δ. More precisely, we prove the following.

THEOREM 3.1. *For any integer Δ, there exists a robot with the property that for any graph G of degree bounded by Δ, it is possible to color the nodes of G with two colors (or alternatively, assign each node a 1-bit label) so that using the labeling, the robot can explore the graph G, starting from any given node and terminating after identifying that the entire graph has been traversed. The robot has $O(\log \Delta)$ bits of memory, and the total number of edge traversals by the robot is $O(\Delta^{10} m)$.*

Note, that in the case where Δ is constant, Theorem 3.1 implies that there exists a constant memory robot and a coloring with two colors of any graph G with maximum degree Δ, such that the robot can explore G using $O(m)$ edge traversals.

To prove the theorem, we first describe a 1-bit labeling scheme $\mathcal{L}'$ for $G = (V, E)$, i.e., a coloring of each node in black or white. Then, we show how to perform exploration using $\mathcal{L}'$.

Labeling $\mathcal{L}'$.. As for the labeling $\mathcal{L}$ of the previous section, pick an arbitrary node $r \in V$, called the *root*. Nodes at distance d from r are labeled as a function of $d \bmod 8$. Denote the distance between two nodes v and u in G by $\mathrm{dist}_G(v, u)$. Partition the nodes into eight *classes* by letting

$$C_i = \{u \in V \mid \mathrm{dist}_G(r, u) \bmod 8 = i\}$$

for $0 \leq i \leq 7$. Node u is colored white if $u \in C_0 \cup C_2 \cup C_3 \cup C_4$, and black otherwise. Let

$$\tilde{C}_1 = \{u \mid \mathrm{dist}_G(r, u) = 1\}$$

$$\widehat{C} = \{r\} \cup \{u \in C_2 \mid \mathcal{N}(u) = \tilde{C}_1\}.$$

LEMMA 3.2. *There is a local search procedure enabling a robot of $O(\log \Delta)$ bits of memory to decide whether a node u belongs to $\widehat{C}$ and to $\tilde{C}_1$, and to identify the class C_i of every node $u \notin \widehat{C}$.*

Proof: Let **B** (resp., **W**) be the set of black (resp., white) nodes all of whose neighbors are also black (resp., white). One can easily check that the class C_1 and the classes $C_3, \ldots, C_7$ can be redefined as follows:

—$u \in C_6 \Leftrightarrow u \in \mathbf{B}$ and there is a node in $\mathbf{W}$ at distance at most 3 from u;

—$u \in C_7 \Leftrightarrow u \notin C_6$, u has a neighbor in C_6, and there is no node in $\mathbf{W}$ at distance at most 2 from u;

—$u \in C_1 \Leftrightarrow u$ is black, u has no neighbor in $\mathbf{B}$, and u has a white neighbor v that has no neighbor in $\mathbf{W}$.

—$u \in C_5 \Leftrightarrow u$ is black, and $u \notin C_1 \cup C_6 \cup C_7$;

—$u \in C_3 \Leftrightarrow u \in \mathbf{W}$, and there is a node in C_1 at distance at most 2 from u;

—$u \in C_4 \Leftrightarrow u$ has a neighbor in $\mathbf{W}$, and there is no node in C_1 at distance at most 2 from u.

Based on the above characterizations, the classes C_1 and $C_3, \ldots, C_7$ can be easily identified by a robot of $O(\log \Delta)$ bits, via performing a local search. For example, the characterization of C_6 is correct because of the following two observations. First, a node in $\mathbf{B}$ can only belong to C_6 or C_7. Second, a node in C_7 never has a node in $\mathbf{W}$ at distance at most 3 from itself. The other characterizations can be proved similarly. Using the same ideas, the sets $\tilde{C}_1$ and $\widehat{C}$ can also be characterized as follows:

—$u \in \tilde{C}_1 \Leftrightarrow u \in C_1$ and there is no node in C_7 at distance at most 2 from u;

—$u \in \widehat{C} \Leftrightarrow N(u) \subseteq \tilde{C}_1$ and every node v at distance at most 2 from u satisfies $|N(v) \cap \tilde{C}_1| \leq |N(u)|$.

Using this we can deduce:

—$u \in C_0 \setminus \widehat{C} \Leftrightarrow u \notin (\cup_{i=3}^{7} C_i) \cup C_1$ and u has a neighbor in C_7;

—$u \in C_2 \setminus \widehat{C} \Leftrightarrow u \notin \widehat{C} \cup C_1$, u has a neighbor in C_1 but no neighbor in C_7.

It follows that a robot of $O(\log \Delta)$ bits can identify the class of every node except for nodes in $\widehat{C}$.   ∎

Proof of Theorem 3.1:

The exploration algorithm for $\mathcal{L}'$ follows the same strategy as the exploration algorithm for $\mathcal{L}$. Indeed, for $u \in C_i$ we have

$$\begin{aligned} \mathsf{pred}(u) &= \mathcal{N}(u) \cap C_{i-1 \bmod 8}, \\ \mathsf{succ}(u) &= \mathcal{N}(u) \cap C_{i+1 \bmod 8}, \\ \mathsf{sibling}(u) &= \mathcal{N}(u) \cap C_i. \end{aligned}$$

Therefore, due to Lemma 3.2, all instructions of the exploration algorithm using labeling $\mathcal{L}$ can be executed using labeling $\mathcal{L}'$, but for the cases not captured in Lemma 3.2, i.e., $\widehat{C}$.

To solve the problem of identifying the root, we notice that each of the nodes in $\widehat{C}$ can be used as a root and all the others can be considered as leaves in C_2, without changing the classes of the nodes not belonging to $\widehat{C}$. Thus, when leaving the root, the robot memorizes the port p by which it should return to the root. When the robot leaves a node $u \in \tilde{C}_1$ through port p, it leaves it in the UP state and deletes the content of the variable storing p. Then, on the reached node v, the robot acts as if v would be in the class C_0 (which is the case since v is the root), i.e., it goes down through the next unexplored port, if there is one. When the robot

leaves a node $u \in \tilde{C}_1$ through a port different from p, and reaches a node in $\widehat{C}$, it leaves it in the DOWN state and then acts as if the reached node would be in the class C_2, i.e., it comes back to u in the UP state.

If the exploration begins at the root, then the above is sufficient. To handle explorations beginning at an arbitrary node, it is necessary to identify the root. Since every node in $\widehat{C}$ can be used as a root, it suffices to find one node of $\widehat{C}$ by going up, and then start the exploration from it as described above.

It remains to compute the number of edge traversals performed by the robot during the exploration. The only edge traversals performed by the robot using $\mathcal{L}'$ but not performed by the robot using $\mathcal{L}$ are the edge traversals performed by the robot during the local search procedure enabling the robot to identify the class of the current node. Thus, the total number of edge traversals performed by the robot considered in this section is at most $O(f(\Delta)m)$, where $f(Delta)$ is the maximum number of edge traversals performed during the execution of the local search procedure.

The fact that $f(Delta) = O(\Delta^{10})$ follows by observing that the local search procedure can be decided by inspecting the neighborhood of the node up to distance 10. Indeed, the membership of a node in sets $C_6, C_7, C_1, C_5, C_3, C_4, \tilde{C}_1, \widehat{C}, C_0 \setminus \widehat{C}$, and $C_2 \setminus \widehat{C}$ can be decided by inspecting the neighborhood of this node up to distance, respectively, $4, 5, 3, 5, 5, 5, 7, 10, 6$, and 10.

We note that the bound on the total number of edge traversals can possibly be reduced further, using the following observation: once the robot has computed the class of the current node v using the described local search procedure, the class of any neighbor of v can only take three identified values. In this case, the robot can use a simplified procedure to identify the class of any subsequently visited node. ∎

4. IMPOSSIBILITY RESULTS

In this section we aim to prove some bounds on exploration by robots with no memory, i.e., single state robots. We present two such bounds, one for exploring graphs with self loops (loops connecting a node to itself) and one for simple graphs, containing no self loops. We assume the robots are stateless and therefore the port traversed by the robot depends only on the color and the degree of the current node and of the label of the port by which the robot arrived at this node (unless the node is the starting node, in which case the port traversed depended only on the node's color and degree).

We aim to construct a graph (with self loops) that cannot be explored by a 1-state robot. The constructed graph will be a star-like graph with the middle node having degree d, where some of its ports lead to degree one nodes while others loop back into the middle node. We will show that an adversary, that is given the details of the robot's transition table for each of $c \le d/2 - 1$ colors, can construct such a graph which can not be explored by this robot. The proof relies on arguing that at least one node will not be explored for every possible coloring of the central node. We begin by several definitions that will help us construct such a graph.

Recall that when a 1-state robot enters a degree-d node v by port i, it will leave v by port j where j depends only on i, d and the color c of v. Thus for fixed d, each

color corresponds to a mapping from entry ports to exit ports, namely, a function from $\{0, 1, \cdots, d-1\}$ to $\{0, 1, \cdots, d-1\}$. Partition the functions corresponding to the colors of nodes of degree d into surjective functions $f_1, f_2, \cdots, f_t$ and non-surjective functions $g_1, g_2, \cdots, g_r$. We have $0 < t + r \leq d/2 - 1$. Let c_i be the color corresponding to f_i, and c_{t+i} be the color corresponding to g_i. For each g_i, choose p_i to be some port number not in the range of g_i. Since for each i, p_i is not in the range of g_i, starting from a node colored by c_{t+i} the robot will never traverse port p_i.

To show that the central node can not be colored by any of the colors c_i, $1 \leq i \leq t$ (the colors with a surjective transition function) a graph should be constructed in which coloring the central node by any of these colors will result in at least one of the leaves not being explored. To do that we construct a family $\{G_0, G_1, \cdots, G_t\}$ of graphs such that, for every $k \in \{0, 1, \cdots, t\}$:

(1) G_k has exactly one degree-d vertex v (possibly with loops);
(2) all edges are either loops incident to v, or edges leading from v to some degree-1 node;
(3) edges labeled $p_1, p_2, \cdots, p_r$ at v (if any, i.e., if $r > 0$) are not loops (and thus lead to degree-1 nodes);
(4) the edge labeled p_0 leads to some distinguished degree-1 node, denoted by u_0;
(5) there exists a set $X_k \subseteq \{0, 1, \cdots, d-1\}$ such that $\{p_0, p_1, \cdots, p_r\} \subseteq X_k$ and $d - |X_k| > 2(t - k)$, and for which, in G_k, edges with port number not in X_k lead to degree-1 vertices.

For a given graph G_{k-1}, we define the following: For a port $i \in \{0, 1, \cdots, d-1\}$, set $\mathsf{twin}(i) = j$ if there exists a port j and a loop labeled by i and j in G_{k-1}; Set $\mathsf{twin}(i) = i$ otherwise. This definition of twin corresponds to the port by which the robot reenters the central node after leaving it using port i; if port i leads to a leaf then the robot returns from the leaf to the central node using the same port, and if port i leads to a loop returning through port j, the robot immediately reenters the central node through port j.

We prove the following property for any $k = 0, \cdots, t$:

Property P_k.

In G_k, if the color of v is in $\{c_1, \cdots, c_k\}$, then the robot, starting at $u_0 \in V(G_k)$, cannot explore G_k. More precisely any vertex attached to v by a port $\notin X_k$ is not visited by the robot.

The proof relies on starting with a star graph, and for each color, k, replacing two nodes by a loop (if needed). The loop is formed between appropriate ports, chosen such that the robot will return to previously visited ports without completing the exploration of all ports.

LEMMA 4.1. *For every $k = 0, \ldots, t$ there exists a graph G_k, having property P_k.*

Proof: We prove the lemma by induction on k. First, note that Property P_0 is trivially true for every graph. Still, to ease the induction proof, we would like to define the graph G_0 and the set X_0. Let G_0 be the star composed of one degree-d vertex v and d leaf vertices. Let $p_0 \in \{0, 1, \cdots, d-1\} \setminus \{p_1, p_2, \cdots, p_r\}$ (it is possible because $d - r \geq 1$), and let $X_0 = \{p_0, p_1, p_2, \cdots, p_r\}$. Recall that
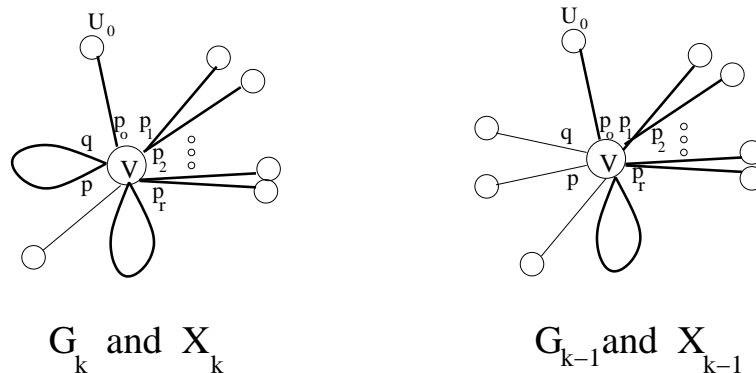
$$G_k \text{ and } X_k \qquad\qquad G_{k-1} \text{ and } X_{k-1}$$

Fig. 2. In the right figure, the thick lines correspond the ports in X_{k-1}. Port p is the first port not in X_{k-1} that is visited by the robot at v, assuming that the starting node in u_0 and that v is colored with color c_k. Port q is that port i_h, where h is the smallest such that $i_h \notin X_{k-1}$. In G_k, Ports p and q (if they are different) are connected, constructing a loop, as depicted in the left figure. X_k then becomes $X_k = X_{k-1} \cup \{p, q\}$.

$t + r \le d/2 - 1$. Thus, $t \le d/2 - 1$ and hence $2t + r + 1 \le d - 1$. Therefore, we have $d - |X_0| = d - (r + 1) > 2t$.

Let $k > 0$, and let G_{k-1} and X_{k-1} be respectively a graph and a set satisfying the induction property for $k - 1$. Assume first that v is colored by color c_k and that the robot starts its traversal at u_0. If the robot never visits vertices attached to v by ports not in X_{k-1} then the graph G_{k-1} and the set X_{k-1} satisfy P_k. I.e., $G_k = G_{k-1}$ and $X_k = X_{k-1}$. Otherwise, let p be the first port not in X_{k-1} that is visited by the robot at v, when starting at u_0.

Define a sequence of ports $(i_l)_{l \ge 1}$ as follows. Let i_1 be the port in X_{k-1} such that $f_k(i_1) = p$. For all $l \ge 2$, let i_l be the port such that $f_k(i_l) = \mathsf{twin}(i_{l-1})$. This sequence is well defined because f_k is surjective. The sequence represents (in reverse order) the ports spanned by the robot on its exploration process towards port p.

Observe that there exists some l such that $i_l \notin X_{k-1}$. Indeed, suppose, for the purpose of contradiction, that $i_l \in X_{k-1}$ for all l. Since X_{k-1} is finite, there exists some $i_l = i_{l+m}$ for $m \ge 1$. Let i_l be the first port repeated twice in this process. If $l > 1$, then we have $f_k(i_l) = \mathsf{twin}(i_{l-1})$ and $f_k(i_{l+m}) = \mathsf{twin}(i_{l+m-1})$. Therefore $\mathsf{twin}(i_{l-1}) = \mathsf{twin}(i_{l+m-1})$, yielding $i_{l-1} = i_{l+m-1}$ by bijectivity of f_k, which contradicts the minimality of l. If $l = 1$, then we have $i_1 = i_{1+m}$, therefore $i_m = p$, contradicting $i_j \in X_{k-1}$ for all j.

From the above, let h be the smallest index such that $i_h \notin X_{k-1}$. Let $q = i_h$. If $q = p$, then set $G_k = G_{k-1}$ and $X_k = X_{k-1} \cup \{p\}$. If $q \ne p$, then connect ports p and q to create a loop, denote the new graph G_k and let $X_k = X_{k-1} \cup \{p, q\}$. See Figure 2.

In G_k, if v is colored by color c_k, then by the choice of p, starting at u_0, the robot enters and exits v through ports in X_{k-1} until it eventually exits v through port p. After that, the robot goes back to v by port q. Port q was chosen so that it causes the robot to continue entering v on ports $i_{h-1}, i_{h-2}, \cdots i_1$, after which the robot exits v through port p, locking the robot in an infinite loop. Since the ports

of v occurring in this cycle are all from X_k, the robot does not visit any of the
ports outside X_k, as claimed. By induction, we have $d - |X_{k-1}| > 2(t - (k-1))$.
By the construction of X_k from X_{k-1}, we have $|X_k| \leq |X_{k-1}| + 2$. Therefore
$d - |X_k| > 2(t - k)$, which completes the correctness of G_k and X_k.

If the color of v in G_k is in $\{c_1, \cdots, c_{k-1}\}$ then the robot is doomed to fail in
exploring G_k. Indeed, since starting at u_0 in G_{k-1} the robot does not traverse any
of the vertices corresponding to ports not in X_{k-1}, then in G_k too, the robot does
not traverse any of the vertices corresponding to ports not in $X_k \supseteq X_{k-1}$, and thus
fails to explore G_k because $d - |X_k| \geq 1$. This completes the proof of P_k and thus
the induction.   ∎

THEOREM 4.2. *For any $d > 4$, and for any 1-state robot using at most $d/2 - 1$
colors, there exists a graph (with loops) with maximum degree d and at most $d+1$
vertices that cannot be explored by the robot.*

Proof: Fix $d > 4$, and assume for contradiction that there exists a 1-state robot
exploring all graphs of degree d colored with at most $d/2 - 1$ colors. By Lemma
4.1, G_t has property P_t and thus at least one of its vertices is not explored by the
robot if the node v is colored with a color in $c_1, c_2, \cdots, c_t$. If v is colored c_{t+i} with
$1 \leq i \leq r$, then assume that the robot starts the traversal at vertex u_0. Since the
edge labeled p_i leads to a degree-1 vertex in G_t, this vertex will never be visited by
the robot, by definition of p_i. Therefore the graph G_t cannot be explored by the
robot.   ∎

The theorem above makes use of graphs with loops. For graphs without loops we
use the same construction, replacing loops with simple subgraphs without loops.
However, in a loop, a robot entering the loop through one port will immediately
return to its last visited node through the other port. If the loop is replaced by a
more complex structure we need to guarantee that the robot will necessarily return
to the node from which it entered this structure using the second port connected
to this structure.

Consider a node w of degree d, accessible from the rest of the graph through two
ports, p and q, and all its other $d - 2$ ports lead to leaves. We aim to show that
there exists a labeling of the two ports, such that entering the node through one of
them, will guarantee leaving through the other, regardless of the color used for the
node. As before we define the set $\{p_1, \cdots, p_r\}$ of ports excluded from non-surjective
colors and have each of them lead to a leaf. We also note, that by looking at each
function f_i as a permutation on d elements, if for some f_i, the cycle decomposition
of this permutation consists of more than two cycles, then w cannot be colored
by f_i because whatever ports p and q are chosen, the robot can visit only ports
corresponding to the cycles that p and q belong to. Therefore we may assume that
the cycle decomposition of each of the relevant f_i consists of at most two cycles.
We will term such a color (whose function is surjective and contains at most two
cycles) as *useful*.

Using the next two lemmas we show that for high enough degree d, we can always
choose two ports $p, q \notin \{p_1, \cdots, p_r\}$ such that for every *useful* color they belong to
the same cycle.

LEMMA 4.3. *Given the functions $f_1, \cdots, f_k$ of k useful colors, there exist at least $d/2^k$ ports that are in the same cycle for each of the k functions.*

Proof: We prove by induction on k. For $k = 1$, f_1 contains at most two cycles permuting the d ports, therefore one of these cycles contains at least $d/2$ ports.

Assume that the induction hypothesis is true for $k = n$, i.e., there are $s \geq d/2^n$ ports that are in the same cycle for each of $f_1, \cdots, f_n$. Now, f_{n+1} contains at most two cycles, so one of them contains at least half of the s ports. Therefore, there are at least $s/2 \geq d/2^{n+1}$ ports that belong to one cycle in each of the $n + 1$ functions, completing the induction. ∎

LEMMA 4.4. *For any $d > 4$ and for $c \leq \lfloor \log d \rfloor - 2$ colors, there exists at least two ports $p, q \in \{0, 1, \cdots, d - 1\} \setminus \{p_1, p_2, \cdots, p_r\}$ that belong to the same cycle for all useful colors.*

Proof: Assume that there are $k \leq t$ *useful* colors, by Lemma 4.3 there are at least $d/2^k$ ports belonging to the same cycle for all *useful* colors. Now $d/2^k \geq d/2^t \geq 2^{2+c-t} = 2^{2+r} = 4 \cdot 2^r$. Therefore, there are at least $4 \cdot 2^r$ ports belonging to one cycle for all useful colors, and $4 \cdot 2^r \geq 2 + r$ for all $r \geq 0$. Thus, p and q can always be selected to be other than $p_1, p_2, \cdots, p_r$. ∎

We now turn to the main theorem, building on the construction with loops of Theorem 4.2 presented above, and replacing the loops by paths in the graph.

THEOREM 4.5. *For any $d > 4$ and for any 1-state robot using at most $\lfloor \log d \rfloor - 2$ colors, there exists a graph of maximum degree d, without loops, that cannot be explored by the robot.*

Proof: As in the proof of Theorem 4.2, denote the functions corresponding to the colors by $f_1, f_2, \cdots, f_t, g_1, g_2, \cdots, g_r$ where the f_i's are surjective functions and the g_i's are non-surjective ones. For each g_i, choose p_i to be some port number not in the range of g_i. We use the same proof as for Theorem 4.2, except that we replace each loop L of the construction in there by a loopless graph. For a loop L of vertex v, with port numbers i and j, let G_L be the following graph. (See Figure 3.) Connect v by edges to two new vertices w and w', add the edge (w, w'), and connect w (respectively, w') to $d - 2$ new vertices $n_1, n_2, \cdots, n_{d-2}$ (resp., $n'_1, n'_2, \cdots, n'_{d-2}$). We now set the port numbering for G_L. Assign the port numbers i and j for the edges (v, w) and (v, w'). At w (respectively, w') use the port numbers $p_1, p_2, \cdots, p_r$ for the edges $(w, n_1), (w, n_2), \cdots, (w, n_r)$ (resp., $(w', n'_1), (w', n'_2), \cdots, (w', n'_r)$). The port numbers p and q that w uses for the edges (w, v) and (w, w') are the same as the port numbers that w' uses for the edges (w', v) and (w', w). They are chosen to be in the same cycle for all useful colors, as shown in Lemma 4.4.

We immediately get that w and w' cannot be colored by a color whose corresponding function is in the set $g_1, g_2, \cdots, g_r$ since these functions are not surjective and ports $\{p_1, p_2, \cdots, p_r\}$ lead to leaves of w and w'. Also, w and w' cannot be colored by a non-*useful* color, since it contains at least three cycles, and since w and w' can only be entered through two ports (p and q) all ports belonging to cycles not containing p and q will not be visited.

Therefore, w and w' must be colored by a *useful* color. However, for all useful colors, p and q belong to the same cycle, and thus a tour entering through port p
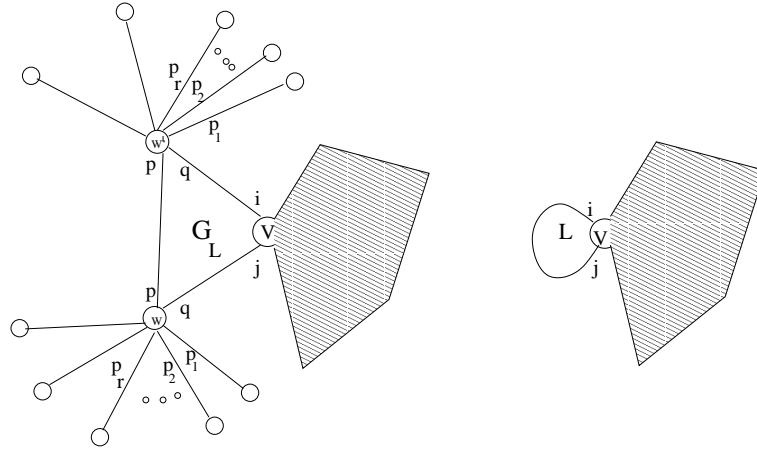
Fig. 3. The loop L (with ports i and j) in the right figure, is replaced by the subgraph G_L, as depicted in the left figure. Ports p and q are chosen such that $\{p,q\} \cap \{p_1, p_2, \cdots p_r\} = \emptyset$ and such that they belong to the same cycle, for every useful color. It follows that if the robot exits v through port i, it must return to v via port j, and vice versa.

must leave through q and vice versa. It follows that whenever the robot enters G_L from v in port i it must return to v in port j (and vice versa). Therefore, the loop L in the proof of Theorem 4.2 can be replaced by G_L. ∎

5. DISCUSSION

It was known that there is no 0-bit exploration-labeling scheme, even for bounded degree graphs. We proved that there is a 2-bit exploration-labeling scheme for arbitrary graphs, and that there is a 1-bit exploration-labeling scheme for bounded degree graphs. It remains open whether or not there exists a 1-bit exploration-labeling scheme for arbitrary graphs.

REFERENCES

BLUM, M. AND KOZEN, D. 1978. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Symposium on Foundations of Computer Science (FOCS)*, 132–142.

BUDACH, L. 1978. Automata and labyrinths. *Math. Nachrichten*, 195–282.

DIKS, K., FRAIGNIAUD, P., KRANAKIS, E., AND PELC, A. 2004. Tree Exploration with Little Memory. *Journal of Algorithms 51 (1)*, 38–63.

FRAIGNIAUD, P., ILCINKAS, D., PEER, G, PELC, A. AND PELEG, D. 2005. Graph Exploration by a Finite Automaton. *Theoretical Computer Science 345 (2-3)*, 331–344.

FRAIGNIAUD, P., ILCINKAS, D., AND PELC, A. 2006. Oracle size: a new measure of difficulty for communication tasks. In *Proc. the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*.

FRAIGNIAUD, P., ILCINKAS, D., RAJSBAUM, S., AND TIXEUIL, S. 2005. Space lower bounds for graph exploration via reduced automata. In *Proc. 12th Int. Colloq. on Structural Information and Communication Complexity (SIROCCO)*, 140–154.

FRAIGNIAUD, P., KORMAN, A., AND LEBHAR, E. 2007. Local MST Computation with Short Advice. In *Proc. 19th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*.

GASIENIEC, L, PELC, A, RADZIK, T, AND ZHANG, X. 2007. Tree exploration with logarithmic memory. In *proc. 18th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 585–594.

KORMAN, A., KUTTEN, S., AND PELEG, D. 2005. Proof Labeling Schemes. In *Proc. the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*.

KOZEN, D. 1979. Automata and planar graphs. *Fundamentals of Computation Theory (FCT)*, 243–254.

RABIN, M.O. 1967. Maze threading automata. *In a Seminar talk presented at the University of California at Berkeley*.

ROLLIK, H.A. 1980. Automaten in planaren Graphen. *Acta Informatica 13*, 287–298.

SHANNON, C.E. 1951. Presentation of a maze-solving machine. In *8th Conf. of the Josiah Macy Jr. Found. (Cybernetics)*, 173–180.