Label-Guided Graph Exploration by a Finite Automaton

Reuven Cohen[1] *, Pierre Fraigniaud[2] **, David Ilcinkas[2] **, Amos Korman[1], and David Peleg[1]

[1] Dept. of Computer Science, Weizmann Institute, Israel
{r.cohen,amos.korman,david.peleg}@weizmann.ac.il
[2] CNRS, LRI, Université Paris-Sud, France
{pierre,ilcinkas}@lri.fr

Abstract. 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 $(k+1)$-node 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).

1 Introduction

Let $\mathcal{R}$ be a finite automaton, simply referred to in this context as a *robot*, moving in an unknown graph $G = (V, E)$. The robot has no a priori information about the topology of G and its size. To allow the robot $\mathcal{R}$, visiting a node u, to distinguish between its edges, the $d = \deg(u)$ edges incident to u are associated to 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,

* Supported by the Pacific Theaters Foundation.
** Supported by the project "PairAPair" of the ACI Masses de Données, the project "Fragile" of the ACI Sécurité et Informatique, and by the project "Grand Large" of INRIA.

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 transition function f, and a finite number of states. 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 first known algorithm designed for graph exploration was introduced by Shannon [8]. Since then, several papers have been dedicated to the feasibility of graph exploration by a finite automaton. Rabin [6] conjectured that no finite automaton with a finite number of 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 [2], for a robot without pebbles. Blum and Kozen [1] improved Budach's result by proving that a robot with three pebbles cannot perform exploration of all graphs. Kozen [5] proved that a robot with four pebbles cannot explore all graphs. Finally, Rollik [7] 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 [4] 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 DFS using the transition function $f(i, d) = i + 1 \bmod d$ (see, e.g., [3]).

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 of 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. This paper considers the effects of allowing the system designer to assign labels to the nodes in a preprocessing stage, and using these labels to guide the exploration by the robot. The transition function f is augmented to utilize labels as follows. If $\mathcal{R}$ 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.

In this paper, we consider settings where it is expected that the graph will be visited by many exploring robots, and consequently, the system designer would like to preprocess the graph by leaving (preferably small) road-signs, or *labels*, that will aid the robots in their exploration task. As possible scenarios one may consider a network system where finite automata are used for traversing the system and distributing information in a sequential manner.

More formally, 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

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 1. Summary of main results.

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.

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 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.

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 1. The two mentioned labeling schemes require polynomial preprocessing time.

We also prove several impossibility results for 1-state robots, i.e., robots that are oblivious. 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

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

Theorem 1. *There exists a robot with the property that for any 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 entire* *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 $\leq 20m$.*

To prove Theorem 1, we first describe the labeling scheme $\mathcal{L}$ and then the exploration algorithm. The node labeling is in fact very simple; it uses three labels, called colors, and denoted WHITE, BLACK, and RED. Let D be the diameter of the graph.

Labeling $\mathcal{L}$. Pick an arbitrary node r. Node r is called 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 leading 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 $\mathsf{pred}(u)$.

Remark. The difficulty 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, has not enough memory to store the value of p.

Exploration algorithm. 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 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.

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.

Assume that $\mathcal{R}$ enters a node u via port number i, in state DOWN. Assume u is of degree d; all arithmetic operations in the following description are modulo d. $\mathcal{R}$ aims at identifying a child of u if one exists, or to backtrack along edge i of u if none exists. To do so it 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.

Assume that $\mathcal{R}$ enters a node u via port number i, in state UP. Assume u is of degree d; all arithmetic operations in the following description are modulo d. $\mathcal{R}$ 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. To do so, $\mathcal{R}$ 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 in state UP. In this case, $\mathcal{R}$ first check 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 visit edge $k+1, k+2, \cdots$ 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 denotes 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 $\mathsf{pred}(v)$. Then it moves to w, and reports the value of `edge`.

This completes the description of our exploration procedure.

Proof of Theorem 1. Clearly, labeling all nodes by $\mathcal{L}$ can be done in time linear in m, the number of edges of the graph. Obviously, two bits are enough to encode the label of each node. More specifically, using two bits for a color that is present on at most one third of the nodes, and one bit for the two other colors, we obtain a labeling with average label size $4/3$. 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 $u = \mathsf{parent}(v)$ or $u \in \mathsf{child}(v)$. Thus, we focus on the correctness of 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. We check separately the three cases considered in the description of the procedure. By the previous observations, comparing the color of the current node with the color of u allows $\mathcal{R}$ to distinguish between these cases.

If $v \in \mathsf{sibling}(u)$, 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 $v \in \mathsf{pred}(u)$, then $v = \mathsf{parent}(u)$ iff for every neighbor w_k connected to u by an edge with port number $k \in \{j-1, j-2, \ldots, 0\}$, $w_k \notin \mathsf{pred}(u)$. The robot does check this property in Case (b) of the description, by returning to u, and visiting all the w_k's. Hence, Procedure `Check_Edge` performs correctly in this case.

Finally, if $v \in \mathsf{succ}(u)$, then $v = \mathsf{child}(u)$ iff for every neighbor z_l connected to v by an edge with port number $l \in \{i-1, i-2, \ldots, 0\}$, $z_l \notin \mathsf{pred}(v)$. In case (c), the robot does check this property by visiting all the z_l's. 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 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 $0 \leq k \leq i-1$, $z_l \notin \mathsf{pred}(v)$ for all $l \in \{k+1, \dots, i-1\}$, and $u \in \mathsf{pred}(v)$. Thus u is identified as the first neighbor that is met when visiting all v's neighbors by successively traversing edges $k+1, k+2, \dots$ 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.

Hence 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 several calls to `Check_Edge`).

We use again the same notations as in the description and the proof 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 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 $j, j-1, \dots, k+1$ of u, then twice edge k, and finally again twice edges $k+1, \dots, j-1, j$. To summarize, edge k of u is explored twice, and edges $k+1, \dots, j-1, j$ 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 $i-1, i-2, \dots, k+1$ of v, twice edge k, twice again edges $k+1, \dots, i-2, i-1$, 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 $k+1, \dots, i-2, i-1$ 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:

(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 $\texttt{Check\_Edge}(i'')$ 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 $\texttt{Check\_Edge}(j')$ at v and four times again during Procedure $\texttt{Check\_Edge}(j'')$ 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 $\texttt{Check\_Edge}(j)$ at v, edge e is explored twice. e is also explored four times during $\texttt{Check\_Edge}(i)$ 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 $\texttt{Check\_Edge}(i')$ at u and twice again during Procedure $\texttt{Check\_Edge}(i'')$ 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 $\texttt{Check\_Edge}(j')$ at v and four times again during Procedure $\texttt{Check\_Edge}(j'')$ 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 $\texttt{Check\_Edge}(j)$ at v, edge e is explored twice. e is also explored twice during $\texttt{Check\_Edge}(i)$ 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 $\texttt{Check\_Edge}(i')$ at u and four times again during Procedure $\texttt{Check\_Edge}(i'')$ 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 $\texttt{Check\_Edge}(j')$ at v and four times again during Procedure $\texttt{Check\_Edge}(j'')$ at v'. To summarize, edge e is explored at most 20 times during a DFS.

Therefore, our exploration algorithm completes exploration in time $\leq 20|E|$ where $|E|$ is the number of edges in the graph G. □

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 2. *There exists a robot with the property that for any graph G of degree bounded by a constant 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 entire 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 $O(\Delta^{(O(1)}m)$.*

To prove Theorem 2, 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 will show how to perform exploration using $\mathcal{L}'$.

Labeling $\mathcal{L}'$. As for $\mathcal{L}$, pick an arbitrary node $r \in V$, called the *root*. Nodes at distance d from r are labeled as a function of $d \bmod 8$. 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

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

$$\widehat{C} = \{r\} \cup \{u \in C_2 \mid \mathrm{dist}_G(r,u) = 2 \text{ and } \mathcal{N}(u) = \tilde{C}_1\}.$$

Lemma 1. *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 $\mathbf{B}$ (resp., $\mathbf{W}$) be the set of black (resp., white) nodes which have all their neighbors 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 ≤ 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 ≤ 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 ≤ 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 ≤ 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. Moreover, 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 u has no node in C_7 at distance ≤ 2;
- $u \in \widehat{C} \Leftrightarrow N(u) \subseteq \tilde{C}_1$ and every node v at distance ≤ 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;
- $u \in C_2 \setminus \widehat{C} \Leftrightarrow u \notin \widehat{C}$, has a neighbor in C_1, but has 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 $\widehat{C}$. $\square$

Proof of Theorem 2. 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

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

Therefore, due to Lemma 1, 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 1, 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. Thus, when leaving the root, the robot should memorize the port P by which it should return to the root. When the robot arrives at a node $u \in \tilde{C}_1$ through a tree edge and is in the UP state, it leaves immediately through port P and deletes the contents of P, then it goes down through the next unexplored port if one is left. When the robot is in a node $u \in \tilde{C}_1$ and in the DOWN state, it will skip the port P.

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. □

4 Impossibility results

Theorem 3. *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. Recall that when a 1-state robot enters a node v by port i, it will leave v by port j where j is depending 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. Let $p_0 \in \{0, 1, \cdots, d-1\} \setminus \{p_1, p_2, \cdots, p_r\}$ (it is possible because $d - r \geq 1$).

We will 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. the other vertices of G_k are degree-1 neighbors of v;
3. all edges are either loops incident to v, or edges leading from v to some degree-1 node;

4. 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);
5. the edge labeled p_0 leads to some degree-1 node, denoted by u_0;
6. 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.

We will 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 $\notin X$ is not visited by the robot.

We prove P_k by induction on k. Let G_0 be the star composed of one degree-d vertex v and d leaf vertices. Let $X_0 = \{p_0, p_1, p_2, \cdots, p_r\}$. Recall that $t + r \leq d/2 - 1$. Thus, $t \leq d/2 - 1$ and hence $2t + r + 1 \leq d - 1$. Therefore, we have $d - |X_0| = d - (r+1) > 2t$. P_0 is trivially true.

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. 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. Define a sequence of ports $(i_l)_{l \geq 1}$ as follows. Let i_1 be the port in X_{k-1} such that $f_k(i_1) = p$. For all $l \geq 2$, 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.

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 $m \geq 1$. 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 $i_h \notin X$. Let $q = i_h$. If $q = p$, then set $G_k = G_{k-1}$ and $X_k = X_{k-1} \cup \{p\}$. If $q \neq p$, then connect ports p and q to create a loop, denote the new graph G_k and let $X_k = X_{k-1} \cup \{p, q\}$.

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 a cycle. 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.

In particular, G_t 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. $\square$

The theorem above makes use of graphs with loops. For graphs without loops we have the following theorem.

Theorem 4. *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.*

5 Further Investigations

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

1. M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In 19th Symposium on Foundations of Computer Science (FOCS), pages 132-142, 1978.
2. L. Budach. Automata and labyrinths. Math. Nachrichten, pages 195-282, 1978.
3. K. Diks, P. Fraigniaud, E. Kranakis, and A. Pelc. Tree Exploration with Little Memory. In 13th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA), pages 588-597, 2002.
4. P. Fraigniaud, D. Ilcinkas, A. Pelc, G. Peer and D. Peleg. Graph Exploration by a Finite Automaton. In Proc. 29th Int. Symp. on Mathematical Foundations of Computer Science (MFCS), LNCS 3153, 451-462, 2004.
5. D. Kozen. Automata and planar graphs. In Fund. Computat. Theory (FCT), 243-254, 1979. Fundamentals of Computation Theory (FCT), pages 243-254, 1979.
6. M.O. Rabin, Maze threading automata. Seminar talk presented at the University of California at Berkeley, October 1967.
7. H.A. Rollik. Automaten in planaren Graphen. Acta Informatica 13:287-298, 1980 (also in LNCS 67, pages 266-275, 1979).
8. C. E. Shannon. Presentation of a maze-solving machine. In 8th Conf. of the Josiah Macy Jr. Found. (Cybernetics), pages 173-180, 1951.