Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 74 (2008) 808–822

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Fast periodic graph exploration with constant memory ☆,☆☆

Leszek Gąsieniec [a], Ralf Klasing [b,*], Russell Martin [a], Alfredo Navarra [c,1], Xiaohui Zhang [a]

[a] *Department of Computer Science, University of Liverpool, Ashton Street, Liverpool L69 3BX, UK*
[b] *LaBRI, CNRS, Université de Bordeaux 1, 351 cours de la Liberation, 33405 Talence, France*
[c] *Dipartimento di Matematica e Informatica, Universitá degli Studi di Perugia, Via Vanvitelli 1, 06123 Perugia, Italy*

Received 18 April 2007; received in revised form 21 September 2007

Available online 2 October 2007

Abstract

We consider the problem of periodic exploration of all nodes in undirected graphs by using a finite state automaton called later a *robot*. The robot, using a constant number of states (memory bits), must be able to explore any unknown anonymous graph. The nodes in the graph are neither labelled nor coloured. However, while visiting a node v the robot can distinguish between edges incident to it. The edges are ordered and labelled by consecutive integers $1, \ldots, d(v)$ called *port numbers*, where $d(v)$ is the degree of v. Periodic graph exploration requires that the automaton has to visit every node infinitely many times in a periodic manner. In this paper, we are interested in minimisation of the length of the exploration period. In other words, we want to minimise the maximum number of edge traversals performed by the robot between two consecutive visits of a generic node, in the same state and entering the node by the same port. Note that the problem is unsolvable if the local port numbers are set arbitrarily, see [L. Budach, Automata and labyrinths, Math. Nachr. 86 (1978) 195–282]. In this context, we are looking for the minimum function $\pi(n)$, such that, there exists an efficient deterministic algorithm for setting the local port numbers allowing the robot to explore all graphs of size n along a traversal route with the period $\pi(n)$. Dobrev et al. proved in [S. Dobrev, J. Jansson, K. Sadakane, W.-K. Sung, Finding short right-hand-on-the-wall walks in graphs, in: Proc. 12th Colloquium on Structural Information and Communication Complexity, SIROCCO 2005, in: Lecture Notes in Comput. Sci., vol. 3499, Springer, Berlin, 2005, pp. 127–139] that for oblivious robots $\pi(n) \leqslant 10n$. Recently Ilcinkas proposed another port labelling algorithm for robots equipped with two extra memory bits, see [D. Ilcinkas, Setting port numbers for fast graph exploration, in: Proc. 13th Colloquium on Structural Information and Communication Complexity, SIROCCO 2006, in: Lecture Notes in Comput. Sci., vol. 4056, Springer, Berlin, 2006, pp. 59–69], where the exploration period $\pi(n) \leqslant 4n - 2$. In the same paper, it is conjectured that the bound $4n - O(1)$ is tight even if the use of larger memory is allowed. In this paper, we disprove this conjecture presenting an efficient deterministic

☆ Preliminary results concerning this paper appeared in [L. Gąsieniec, R. Klasing, R. Martin, A. Navarra, X. Zhang, Fast periodic graph exploration with constant memory, in: Proc. 14th Colloquium on Structural Information and Communication Complexity, SIROCCO 2007, in: Lecture Notes in Comput. Sci., vol. 4474, Springer-Verlag, 2007, pp. 26–40. [18]].

☆☆ The research was partially funded by the project "ALPAGE" of the ANR "Masse de données: Modélisation, Simulation, Applications," the project "CEPAGE" of INRIA, the European projects COST Action 293, "Graphs and Algorithms in Communication Networks" (GRAAL), COST Action 295, "Dynamic Communication Networks" (DYNAMO), the Nuffield Foundation grant NAL/32566, "The structure and efficient utilisation of the Internet and other distributed systems," and by a visiting fellowship from LaBRI/ENSEIRB.

* Corresponding author.

E-mail addresses: leszek@csc.liv.ac.uk (L. Gąsieniec), ralf.klasing@labri.fr (R. Klasing), martin@csc.liv.ac.uk (R. Martin), navarra@dipmat.unipg.it (A. Navarra), cloud@csc.liv.ac.uk (X. Zhang).

[1] The work was done while the author was a Post-Doc at LaBRI, France.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2007.09.004

algorithm arranging the port numbers, such that, the robot equipped with a constant number of bits is able to complete the traversal period in $\pi(n) < 3.75n - 2$ steps hence decreasing the existing upper bound. This reduces the gap with the lower bound of $\pi(n) \geqslant 2n - 2$ holding for any robot.

Keywords: Finite automaton; Period; Port number; Penalty; Full degree

1. Introduction

We consider the task of graph exploration by a mobile entity equipped with small (constant number of bits) memory. The mobile entity may be, e.g., an autonomous piece of software navigating through an underlying graph of connections of a computer network. The mobile entity is expected to visit all nodes in the graph in a periodic manner. For the sake of simplicity, we call the mobile entity a *robot* and model it as a finite state automaton. The task of periodic traversal of all nodes of a network is particularly useful in network maintenance, where the status of every node has to be checked regularly.

We consider here undirected graphs that are anonymous, i.e., the nodes in the graph are neither labelled nor coloured. However, while visiting a node the robot can distinguish between edges incident to it. At each node v the incident edges are ordered and labelled by consecutive integers $1, \ldots, d(v)$ called *port numbers*, where $d(v)$ is the degree of v. We will refer to port ordering as a *local orientation*.

Following formalism from [20], we model robots as *Mealy automata*. A Mealy automaton uses only input actions, i.e., output depends on input and the current state. In our case, the Mealy automaton has a transition function f and a finite number of states governing the actions of the robot. More precisely, if the automaton enters a node v of degree $d(v)$ through port i in state s, it switches to state s' and exits the node through port i'. This corresponds to $f(s, i, d(v)) = (s', i')$.

Periodic graph exploration requires that the automaton has to visit every node infinitely many times in a periodic manner. In this paper, we are interested in minimising the length of the exploration period. In other words, we want to minimise the maximum number of edge traversals performed by the robot between two consecutive visits of a generic node, in the same state and entering the node by the same port. Budach [7] proved that no finite automaton can explore all graphs if the local orientation is given by an adversary. In this context, we want to determine the minimum function $\pi(n)$, such that, there exists an efficient deterministic algorithm for setting the local port numbers allowing the robot to explore all graphs of size n along a traversal route with the period $\pi(n)$. Dobrev et al. proved in [12] that for oblivious robots $\pi(n) \leqslant 10n$. Very recently Ilcinkas proposed another port labelling algorithm for robots equipped with two extra memory bits, see [20], with exploration period $\pi(n) \leqslant 4n - 2$. The traversal route constructed by Ilcinkas' algorithms is based on edges of an arbitrary spanning tree encoded neatly by the port numbers. The automaton proposed by Ilcinkas is not oblivious but it has only three states. Basically, such an automaton performs the spanning tree traversal by means of a refined version of the so-called *right-hand-on-the-wall rule*.[2] During a period it might traverse twice at most one edge not belonging to the chosen spanning tree for each node. It performs periodic exploration independently of its starting position and the initial state. In addition, if required, the robot is able to stop at the root of the spanning tree after finishing each period of the traversal route, and wait there, e.g., for the next wake-up message. In the same paper, Ilcinkas conjectured that the bound $4n - O(1)$ is tight even if the use of larger memory is allowed.

1.1. Our results

We present an efficient deterministic algorithm arranging port numbers in the graph, such that, the robot equipped with a constant number of bits is able to accomplish each period of the traversal route in $\pi(n) < 3.75n - 2$ steps. This

[2] The right-hand-on-the-wall rule specifies that whenever the automaton enters a node v via port number i, if $i < d(v)$ then it leaves v via port number $i + 1$, 1 otherwise.

invalidates Ilcinkas' conjecture and shows that the problem of determining the minimum function $\pi(n)$ remains wide open. In addition, our result reduces the gap with the lower bound of $\pi(n) \geqslant 2n - 2$ holding for any robot.[3]

The improvement is a consequence of the construction of the traversal route on the basis of a very specific (rooted) spanning tree, rather than an arbitrary tree used by Ilcinkas. The main idea resides in powering the robot in such a way that it can recognise, or more precisely it can expect to meet, specific subtrees in the chosen spanning tree, hence saving in the number of so-called penalties. We say the robot pays a *penalty* at the node v if, starting from v, it traverses an edge not belonging to the spanning tree.

We introduce the concept of *Extended Leaves*, *paired Extended Leaves* and *paired Leaves*. An extended leaf of the spanning tree is a path of length 1 in which one endpoint is a leaf of the tree and the other is an internal node of the tree which has no other children than the leaf. A paired extended leaf is an extended leaf rooted in a node whose children contain at least two nodes that are the endpoints of two extended leaves (see, e.g., Fig. 3). A paired leaf is a leaf rooted at a node whose children contain at least two leaves.

Intuitively, our arrangement of the port numbers at each vertex means once the robot has met an extended leaf it expects to visit a paired one, and once it meets a leaf it expects to visit a paired one. In doing so, it saves penalties at each second, third, etc., paired extended leaf as well as at each second, third, etc., paired leaf since it knows what the topology should be, and hence does not have to explore further edges beyond such a leaf or extended leaf.

Our robot requires few (constant number) states allowing it to take advantage of the specific topology of the spanning tree.

1.2. Related work

Graph exploration by robots has recently attracted growing attention. The unknown environment in which the robots operate is often modelled as a graph, assuming that the robots may only move along its edges. The graph setting is available in two different forms.

In [1,4,5,10,14], the robot explores strongly connected directed graphs and it can move only in one pre-specified direction along each edge. In [2,6,8,13,16,17,21], the explored graph is undirected and the agent can traverse edges in both directions. Also, two alternative efficiency measures are adopted in most papers devoted to graph exploration, namely, the *time* of completing the task [1,2,4–6,10,13], or the number of *memory bits* (states in the automaton) available to the agent [8,11,15–17,19].

Graph exploration scenarios considered in the literature differ in an important way: it is either assumed that nodes of the graph have unique labels which the agent can recognise, or it is assumed that nodes are anonymous. Exploration of directed graphs assuming the existence of labels was investigated in [1,10,14]. In this case, no restrictions on the agent moves were imposed, other than by directions of edges, and fast exploration algorithms were sought. Exploration of undirected labelled graphs was considered in [2,3,6,13,21]. Since in this case a simple exploration based on depth-first search can be completed in time $2m$, where m is the number of edges, investigations concentrated either on further reducing the time for an unrestricted agent, or on studying efficient exploration when moves of the agent are restricted in some way. The first approach was adopted in [21], where an exploration algorithm working in time $m + O(n)$, with n being the number of nodes, was proposed. Restricted agents were investigated in [2,3,6,13]. It was assumed that the agent is a robot with either a restricted fuel tank [2,6], forcing it to periodically return to the base for refuelling, or that it is a tethered robot, i.e., attached to the base by a "rope" or "cable" (a path from the original node) of restricted length [13]. For example, in [13] it was proved that exploration can be done in time $O(m)$ under both scenarios.

Exploration of anonymous graphs by robots with limited memory presents different types of challenges. In this case, it is impossible to explore arbitrary graphs if no marking of nodes is allowed [7]. Hence, the scenario adopted in [4,5] was to allow *pebbles* which the agent can drop on nodes to recognise already visited ones, and then remove them and drop in other places. The authors concentrated attention on the minimum number of pebbles allowing efficient exploration of arbitrary directed n-node graphs. (In the case of undirected graphs, one pebble suffices for efficient exploration.) In [5], the authors compared the exploration power of one agent with pebbles to that of two cooperating agents without pebbles. In [4], it was shown that one pebble is enough, if the agent knows an upper bound on the size of the graph, and $\Theta(\log \log n)$ pebbles are necessary and sufficient otherwise.

[3] Note that this lower bound is obtained, e.g., when the graph to be explored is a tree. And indeed, in a full periodic exploration of a tree every edge of the tree must be traversed at least twice.

In [8,11,15–17], the adopted measure of efficiency was the memory size of the agent exploring anonymous graphs. In [15,17], the agent was allowed to mark nodes by pebbles, or even by writing messages on whiteboards with which nodes are equipped. In [8], the authors studied special schemes of labelling nodes, which facilitate exploration with small memory. Another aspect of distributed graph exploration by robots with bounded memory was studied in [11,19], where the topology of graphs is restricted to trees. In [11] Diks et al. proposed a robot requiring $O(\log^2 n)$ memory bits to explore any tree with at most n nodes. They also provided the lower bound $\Omega(\log n)$ if the robot is expected to return to its original position in the tree. Very recently the gap between the upper bound and the lower bound was closed in [19] by Gąsieniec et al. who showed that $O(\log n)$ bits of memory suffice in tree exploration. However it is known, see [16], that in arbitrary graphs the number of memory bits required by any robot expected to return to the original position is $\Theta(D \log d)$, where D is the diameter and d is the maximum degree in the graph. In comparison, Reingold [22] proved recently that $SL = L$, i.e., any decision problem which can be solved by a deterministic Turing machine using logarithmic memory (space) is log-space reducible to the USTCON (st-connectivity in undirected graphs) problem. This proves the existence of a robot equipped with asymptotically optimal number of $O(\log n)$ bits being able to explore any n-node graph in the perpetual exploration model, where the return to the original position is not required. The respective lower bound $\Omega(\log n)$ is provided in [23].

In this paper, we are interested in robots characterised by very low memory utilisation. In fact, the robots are allowed to use only a constant number of memory bits. This restriction permits modelling robots as finite state automata. Budach [7] proved that no finite automaton can explore all graphs. Rollik [23] showed later that even a finite team of finite automata cannot explore all planar cubic graphs. This result is improved in [9], where Cook and Rackoff introduce a powerful tool, called the *JAG*, for Jumping Automaton for Graphs. A JAG is a finite team of finite automata that permanently cooperate and that can use *teleportation* to move from their current location to the location of any other automaton. However, even JAGs cannot explore all graphs [9].

1.3. Outline of the paper

Section 2 presents the spanning tree construction that will constitute the main route of the robot during its exploration. The same section also shows how to assign port numbers at each vertex. Section 3 includes specification of the robot by means of a finite state machine. Section 4 states the main result of the paper, that is, the analytical proof of the new upper bound of $3.75n - 2$ for the length of traversal period $\pi(n)$ required by robots equipped with constant memory. Further comments on the main themes of this paper can be found in Section 5.

2. The spanning tree construction and the port numbering

In this section, we describe the spanning tree construction and how the port labelling is performed. This will define the route allowing the robot to periodically visit all the nodes of G. During the tree construction we make use of colouring strategies. These will be useful to analyse the length, π, of the closed walk P adopted by our robot to visit G. However, the robot is not aware of such a colouring.

We construct the spanning tree starting from a special subtree, from now on called the *backbone* and denoted by $B = (V_B, E_B)$. Procedures Color() and Backbone_Construction() realise this structure (see Fig. 1). Procedure Backbone_Construction() takes graph G as an input and generates a tree, *the backbone*. Later, using procedure Color(), it colours all the nodes of G. The backbone is formed of *Red* and *Yellow* nodes, with the property that if a path connecting two *Red* nodes contains only yellow nodes, it contains exactly two of them. The remaining nodes are coloured *Green* or *Blue* if their distance from a *Red* node is 1 or 2, respectively. This implies another important property of the backbone, namely, every node outside the backbone is at distance at most two from it.

On the basis of the backbone, procedure Tree_Construction() builds the spanning tree of G, from now on called $ST(G) = (V_T, E_T)$ (see Fig. 1). The procedure also recolours some *Yellow* nodes (not having *Blue*s as neighbours) into *Orange*. Again, such a recolouring will be used only for the purpose of analysing the length of P. Having constructed the spanning tree $ST(G)$, we then need to set the port numbers.

Let v be a node of a rooted tree T. We assume that the children $c_1, \ldots, c_k$ of v are listed in a non-increasing order according to the sizes of $T(c_1), \ldots, T(c_k)$, where $T(c_i)$ is the subtree of T rooted in c_i. The main idea is that we set the port numbers so that the robot will explore the large subtrees at v first, followed by *Extended Leaves* (at v), and then (regular) *Leaves* (at v), allowing it to avoid paying penalties for most of the extended leaves and (regular) leaves.

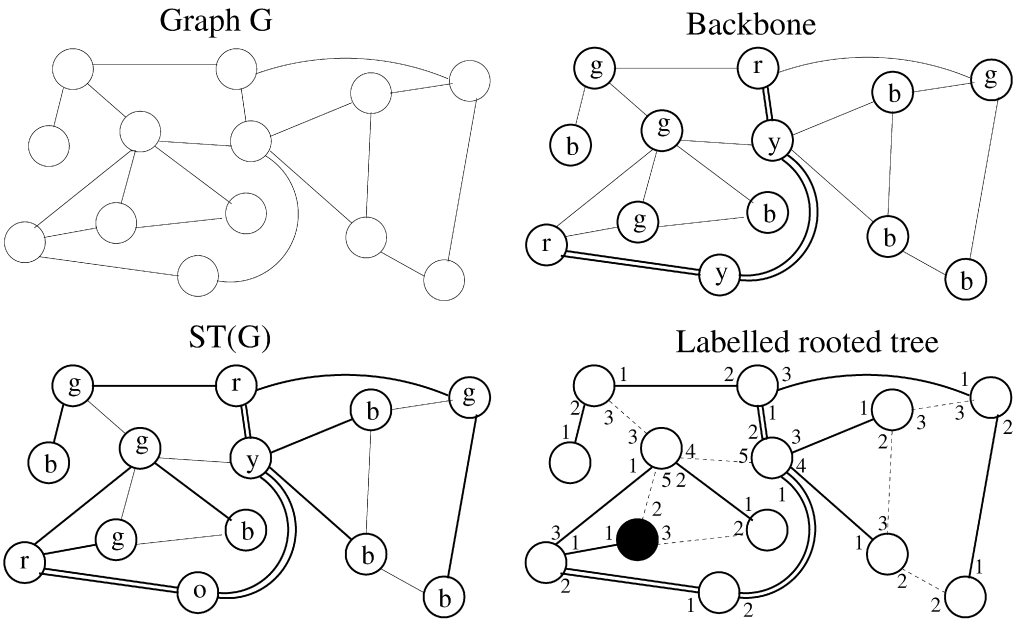

Fig. 1. The construction of $ST(G)$ and its labelling. The path characterised by double lines and joining the red nodes constitutes the backbone. The black node is the chosen root of $ST(G)$.

procedure Color(node: v)
1: Node v becomes *Red*;
2: All nodes at distance 1 from v become *Green*;
3: All not yet coloured nodes at distance 2 from v become *Blue*;

procedure Backbone_Construction(graph: G) $\rightarrow$ Tree
1: Pick an arbitrary node $v \in V$;
2: Color(v);
3: $V_B = \{v\}$;
4: $E_B = \emptyset$;
5: **while** the set of not yet coloured nodes in G is not empty **do**
6: Pick a not yet coloured node $v \in V$ at distance 1 from some *Blue* node w_1 which is itself connected via node w_2 to some *Red* node v';
7: Color(v);
8: $V_B = V_B \cup \{w_1, w_2, v\}$;
9: $E_B = E_B \cup \{(v', w_2), (w_2, w_1), (w_1, v)\}$;
10: Nodes w_1 and w_2 become *Yellow*;
11: **end while**

In what follows we use $d_G(v)$ (respectively, $d_{ST}(v)$) to denote the degree of a node v in the graph G (respectively, the spanning tree $ST(G)$). Procedures Set_Ports() and Labelling() first label ports on edges of the input tree and then provide a consistent labelling to the remaining edges in G (see Fig. 1).

3. The automaton

In this section we provide a formal definition of the automaton that governs the robot's behaviour. The automaton has eleven states, namely, I (Initial), RS (Root Search), RB (Root Backtrack), FF (Forward), N (Normal), T (Test), B (Backtrack), L (Leaf), LB (Leaf Backtrack), E (Extended Leaf), EB (Extended Leaf Backtrack). Moreover, the automaton is powered by an extra two-bits counter c. Indeed, combining c with each one of the eleven states mentioned above, we obtain 33 possible states since c ranges among values $\{0, 1, 2\}$. We also denote by d the degree of the currently visited node and by i the entering port number.

procedure Tree_Construction(graph: G, backbone of G: B) $\rightarrow$ Tree
1: $V_T = V_B$;
2: $E_T = E_B$;
3: **for** each node $v \in V \setminus V_T$ at distance 1 from a red node $v' \in V_B$ **do**
4: $V_T = V_T \cup \{v\}$;
5: $E_T = E_T \cup \{(v, v')\}$;
6: **end for**
7: $TMP = V_T$;
8: **for** each node $v \in V \setminus V_T$ at distance 1 from some node $v' \in TMP$ **do**
9: $V_T = V_T \cup \{v\}$;
10: $E_T = E_T \cup \{(v, v')\}$;
11: **end for**
12: Each *Yellow* node v not connected to any *Blue* node v' by an edge $(v, v') \in E_T$ becomes *Orange*;

procedure Set_Ports(node: v)
// under assumption $|ST(c_1)| \geqslant |ST(c_2)| \geqslant \cdots \geqslant |ST(c_{d_{ST}(v)-1})|$
1: **for** each edge $\{v, c_i\} \in E_T$ with $i = 1, \ldots, d_{ST}(v) - 1$ **do**
2: Set the port incident to v as $i + 1$;
3: Set the port incident to c_i as 1;
4: **end for**

procedure Labelling(graph: G, spanning tree of G: $ST(G)$)
1: Pick an arbitrary leaf in $ST(G)$ and set it as root r;
2: Set the port on the edge $\{r, v\}$ incident to the root r to 1 on both ends;
3: **for** each $v \in V_T \setminus \{r\}$ **do**
4: Set_Ports(v);
5: **end for**
6: Set the remaining ports arbitrarily but consistently with degrees of the nodes;

Regardless of its starting vertex, the robot begins in state I, exiting port number 1 to visit the nodes of the input graph G. Its first goal is to find the right direction of the visit, i.e., from the root down to the leaves. In the worst case this process requires a constant number of steps. For example, it can happen, that the robot starts exploration going through a special edge $e_{1,1}$, the only edge in G with ports labelled 1 at both its endpoints. In this case the robot has to figure out which endpoint is the root. This task is realised by checking which endpoint is a leaf.[4] Once the robot finds out which endpoint leads to the rest of the tree, this initialisation phase is complete and the *proper exploration* starts. Let v be the only child of the root in $ST(G)$ (this is guaranteed by code line 1 of the Labelling procedure). The robot visits the tree following the order of the ports from 2 to $d_{ST}(v)$. It first goes further in a depth-first-search manner (state T) as long as the entering port is 1. If different, it switches its state to L since a leaf has been discovered (recall that order of ports leading to subtrees of an internal node reflects the sizes of the subtrees). At this stage, the robot does not know whether such a leaf is a regular leaf or an extended leaf. It retreats through the same edge remembering (by setting the counter c to 1) that a leaf has been discovered, and it looks for another paired leaf while being in state L.

If now the robot encounters another entering port 1, it updates its leaves counter c to 2 and continues the search for more leaves. Note that, in this way the robot does not pay a penalty (it does not go beyond the leaf) at the second leaf. Moreover, the counter of leaves is not updated anymore, i.e., it is used to count at most two paired leaves. As soon as the robot arrives at some node via a port different from 1 or the whole degree of the node in which the paired leaves are rooted has been explored, it retreats further in the tree, it sets c to 0 and switches its state to N.

Alternatively, if after visiting the first leaf the encountered port number was not 1, the robot goes backwards in the tree, sets the leaves counter to 0, and switches its state to E, since an extended leaf has been discovered. Now the robot searches for extended leaves, i.e., for two consecutive entering ports set to 1.

If after the first port 1 (that is counted by c), it finds a second one, a new extended leaf has been discovered and no penalties are paid at the two visited nodes. However if after the first port 1, the second one is different, the robot goes backwards switching its state to L since a regular leaf was found. Note that, in this case, c is already set to 1. And finally, if the first entering port is not 1, the robot retreats further in the tree while being in the state N.

[4] In the case of the simple graph composed by just one edge, the first met node is considered as the root.

Table 1
The transition function f

$$f(I,i,d,c)=\begin{cases}(FF,1,0) & \text{if } i=d=1 \quad (1)\\ (RS,2,0) & \text{if } i=1 \,\&\, d\neq 1 \quad (2)\\ (T,i,0) & \text{if } i\neq 1 \quad (3)\end{cases}$$

$$f(FF,i,d,c)=\begin{cases}(RB,1,0) & \text{if } d=1 \quad (4)\\ (T,2,0) & \text{if } d\neq 1 \quad (5)\end{cases}$$

$$f(RS,i,d,c)=\begin{cases}(LB,1,1) & \text{if } i=d=1 \quad (6)\\ (T,2,0) & \text{if } i=1 \,\&\, d\neq 1 \quad (7)\\ (RB,i,0) & \text{if } i\neq 1 \quad (8)\end{cases}$$

$$f(RB,i,d,c)=(FF,1,0) \quad (9)$$

$$f(T,i,d,c)=\begin{cases}(LB,1,1) & \text{if } i=d=1 \quad (10)\\ (T,2,0) & \text{if } i=1 \,\&\, d\neq 1 \quad (11)\\ (B,i,1) & \text{if } i\neq 1 \quad (12)\end{cases}$$

$$f(B,i,d,c)=\begin{cases}(N,1,0) & \text{if } c=0 \,\|\, i\neq 2 \quad (13)\\ (LB,1,1) & \text{if } c\neq 0 \,\&\, i=2 \quad (14)\end{cases}$$

$$f(N,i,d,c)=\begin{cases}(FF,1,0) & \text{if } i=1 \quad (15)\\ (N,1,0) & \text{if } i=d\neq 1 \quad (16)\\ (T,i+1,0) & \text{if } i\neq 1 \,\&\, i\neq d \quad (17)\end{cases}$$

$$f(L,i,d,c)=\begin{cases}(LB,1,2) & \text{if } i=1 \quad (18)\\ (EB,i,1) & \text{if } i\neq 1 \,\&\, c=1 \quad (19)\\ (B,i,0) & \text{if } i\neq 1 \,\&\, c\neq 1 \quad (20)\end{cases}$$

$$f(LB,i,d,c)=\begin{cases}(EB,1,0) & \text{if } i=d=2 \quad (21)\\ (N,1,0) & \text{if } i=d\neq 2 \quad (22)\\ (L,i+1,c) & \text{if } i\neq d \quad (23)\end{cases}$$

$$f(E,i,d,c)=\begin{cases}(LB,1,1) & \text{if } i=d=1 \,\&\, c=0 \quad (24)\\ (E,2,1) & \text{if } i=1 \,\&\, d\neq 1 \,\&\, c=0 \quad (25)\\ (EB,1,2) & \text{if } i=1 \,\&\, c\neq 0 \quad (26)\\ (B,i,c) & \text{if } i\neq 1 \quad (27)\end{cases}$$

$$f(EB,i,d,c)=\begin{cases}(FF,1,0) & \text{if } i=1 \quad (28)\\ (EB,1,0) & \text{if } i\neq 1 \,\&\, c=2 \quad (29)\\ (N,1,0) & \text{if } i=d\neq 1 \,\&\, c=0 \quad (30)\\ (E,i+1,0) & \text{if } i\neq 1 \,\&\, i\neq d \,\&\, c=0 \quad (31)\\ (EB,1,0) & \text{if } i=3 \,\&\, c=1 \quad (32)\\ (N,1,0) & \text{if } i\neq 1 \,\&\, i\neq 3 \,\&\, c=1 \quad (33)\end{cases}$$

The way the robot goes backwards depends on its current state, i.e., it must take into account whether it is currently looking for the root, leaves, extended leaves, whether it has just to leave the current subtree, or finally whether it has to go back since the traversed edge does not belong to the spanning tree (i.e., the robot paid a penalty). This is the reason why the robot must be equipped with five different backward states (*RB*, *LB*, *EB*, N, and B respectively).

Formally, the transition function f is defined as follows. If the robot enters a node v of degree $d_G(v)$ through port i in state s with current value of the two-bit counter c, it switches to state s' and exits the node through port i' with the counter set to c'. This corresponds to $f(s,i,d_G(v),c)=(s',i',c')$. Please note that function f has now four arguments in contrary to the more standard definition of Mealy automata presented in Section 1. This change is necessary to incorporate the introduction of the counter c.

Table 1 shows the transition function f. The first nine rules are used in the initialisation step of the search, while the remaining ones are devoted to the proper (periodic) graph exploration. An example of the exploration performed by the automaton can be found in Appendix A.

Theorem 1 *(Correctness). Let G be a graph of size n, and let $ST(G)$ be a spanning tree of G constructed and labelled as previously described in Section* 2. *Starting at any node of G, the robot begins in the initial state I and follows exit port number* 1. *Then*

(a) *After at most* 8 *steps, the robot enters a closed walk* P *and then periodically explores* G *forever.*
(b) *Along the walk* P*, every internal node* $v \in ST(G)$ *that is not the endpoint of an extended leaf or a leaf, is entered via port number* 1 *by the robot with state* T *or* FF *while descending* $ST(G)$ *and left via port number* 1 *with state* N.
(c) *Suppose that* v *is the root of* k *extended leaves in* $ST(G)$*. The robot avoids paying possible penalties at the second, third, etc., extended leaf (so it avoids* $2(k-1)$ *penalties from these extended leaves). Similarly, if* v *is the root of* j *paired leaves, the robot only pays a single penalty for the whole set of these paired leaves, hence avoiding* $j-1$ *possible penalties.*

Proof. First we show how the robot recognises the direction of the exploration from the root towards the leaves. At the beginning the robot is in the initial state I and follows port number 1. As shown in the first equation of the transition function f, three different cases can occur:

- If the entering port is 1 and the entering node has degree 1 then such a node is considered as the root since it is a leaf connected to the edge $e_{1,1}$. The robot then exits the current node by means of rule 1 in state FF and the real exploration can start by means of rule 5. The only case in which it is unclear whether the considered node is the root is when G is composed of a single special edge $e_{1,1}$. In this case indeed, both nodes can be considered indifferently as the root since the resulting exploration is symmetric. The task is performed by switching the robot state first from I to FF (rule 1) and then repetitively from FF to RB and vice versa (rules 4 and 9).
- If the entering port is 1 and the entering node has degree larger than 1, then the robot has to verify whether such a node is a leaf or not, and this requires two steps. If a leaf is found (by means of rules 2 and 8) the robot retreats and starts proper exploration with state FF by means of rule 9. If not, then it continues in the current direction assuming that the proper exploration was done from the beginning (either by applying rules 2 and 6 or 2 and 7).
- If the entering port is different from 1 then the direction is determined since, by construction of $ST(G)$, the node connected via the exiting port on the edge is clearly a child of the node connected via the entering port. Hence, the robot has to reverse its direction (rule 3). In the worst case the robot has to perform six steps more before the proper exploration starts. It can happen, e.g., that the robot starts from a paired leaf or from a leaf of a paired extended leaf of $ST(G)$ for which in P no penalties are paid. However, since the robot has just started and does not know whether the current leaf or extended leaf is a paired one or not, it pays some extra penalties that at the successive traversals will not be paid. Namely, it needs at most two steps (sequence of rules 11, 12) to find out whether it is exploring a leaf, whereas, it needs at most six steps (sequence of rules 11, 12, 14, 23, 19, 32) to find out whether it is exploring an extended leaf.

Once this initialisation phase has been completed, the proper (periodic) graph exploration starts. Note that, at this point, states I, RS and RB are no longer accessible by the robot. Apart from cases where G coincides with $e_{1,1}$ and when the "tricks" of skipping penalties at paired leaves and paired extended leaves are used, the exploration follows the approach presented in [20]. It is, in fact, a tree-based exploration. Thus, in order to prove the theorem we have to show that while handling the above mentioned special cases the robot does not skip any nodes in $ST(G)$ and that it switches between standard and special cases safely.

We prove the claim by induction on the height of a node v in $ST(G)$, denoted from now on by $ht(v)$. By means of the following two lemmata, we first analyse the basic cases given by considering nodes at height 0 and 1 in $ST(G)$ and then we formalise the induction hypothesis for nodes at height bigger than 0 that are not the endpoints of extended leaves; see Fig. 2 for a clarification of the considered subcases.

Lemma 1. *Let node* v *be such that* $ht(v) = 0$*, i.e.,* v *is a leaf in* $ST(G)$*. As described in Table* 2*, while descending the tree, i.e., entering* v *via port number* 1*, the robot can be in state* T (*respectively* L *or* E *with* $c = 0$ *or* E *with* $c > 0$) *and it leaves* v *via port number* 1 *in state* LB *with* $c = 1$ (*respectively* LB *with* $c = 2$ *or* LB *with* $c = 1$ *or* EB *with* $c = 1$).

Proof. Let $\{w, v\}$ be the edge along which the robot enters v via port number 1 (and let w be the node from which it comes). The case in which $\{w, v\} \equiv e_{1,1}$ was already considered before, hence we can assume $\{w, v\} \not\equiv e_{1,1}$. Then,

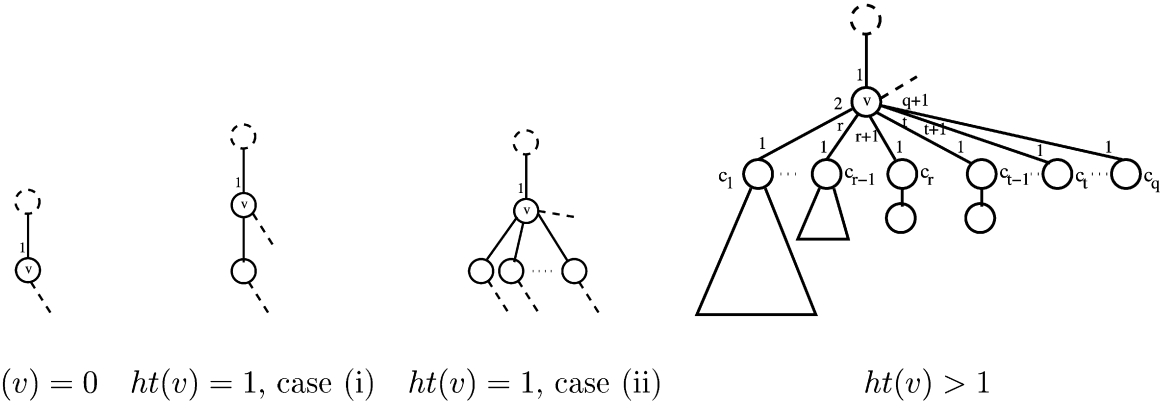

Fig. 2. Possible subtrees rooted in v of height 0, 1 or bigger than 1. Sketched lines stand for possible edges of G not belonging to $ST(G)$.

Table 2
All the possible incoming and outgoing states of the automaton when exploring a node v of height 0, 1 or bigger than 1

Height	↓ incoming state	↑ outgoing state
	T	$LB, c = 1$
$ht(v) = 0$	L	$LB, c = 2$
(*Lemma* 1)	$E, c = 0$	$LB, c = 1$
	$E, c > 0$	$EB, c = 1$
$ht(v) = 1$, case (i)	T	$EB, c = 0$
(*Lemma* 2)	$E, c = 0$	$EB, c = 0$
	FF	$EB, c = 0$
$ht(v) = 1$, case (ii)	T	N
(*Lemma* 2)	FF	N
$ht(v) > 1$	T	N
	FF	N

the other label of $\{w, v\}$ in w must be different from 1. By the definition of the transition function f, the state s with which the robot could exit w via a port number different from 1 is in the set $\{B, EB, T, L, E\}$.

If $s = B$, the only rules that could have led the robot to v from w are rules 12 and 27. If 12, then w should have been reached from v in state T via port number 1 but there is no rule allowing it. If 27, then w should have been reached from v in state E via port number 1, but again there is no rule allowing it. This implies that the robot cannot enter v in state B from w.

If $s = EB$, the only rule that could have led the robot to v from w is rule 19 but in order to apply rule 19 the robot should have come from w and the entering port should have been different from 1. This implies that the robot cannot enter v in state EB from w.

If $s = T$, then we can distinguish two cases. Either $d_G(v) = 1$, then rule 10 is applied, or $d_G(v) > 1$, then the sequence of rules 11, 12, 14 is applied since v is a leaf of $ST(G)$. In both cases the robot exits v in state $s' = LB$ and $c = 1$ via port number 1.

If $s = L$, then the robot retreats to w in state $s' = LB$ via port number 1 and $c = 2$ (rule 18).

If $s = E$, then we can distinguish two cases. Either the counter c is 0 or bigger. If $c = 0$, then either $d_G(v) = 1$ and rule 24 is applied, or $d_G(v) > 1$ and the sequence of rules 25, 27, 14 is applied since v is a leaf of $ST(G)$. In both cases the robot exits v in state $s' = LB$ with $c = 1$ via port number 1. If $c > 0$, then rule 26 is applied and the robot retreats to w in state $s' = EB$ with $c = 1$. □

Lemma 2. *Let node v be such that $ht(v) = 1$ in $ST(G)$. As described in Fig.* 2 *and Table* 2*, while descending the tree, i.e., entering v via port number* 1*, two cases can occur,*

(i) only 1 leaf is rooted in v. *The robot can be either in state* T *or in state* E *with* $c = 0$ *or in state FF, and it leaves* v *via port number* 1 *in state EB with* $c = 0$.
(ii) More than 1 leaf is rooted in v. *The robot can be either in state* T *or in state FF, and it leaves* v *via port number* 1 *in state* N.

Proof. Let $\{w, v\}$ be the edge along which the robot enters v via port number 1 (and let w be the node from which it comes). We first assume $\{w, v\} \not\equiv e_{1,1}$. Then, the other label of $\{w, v\}$ in w must be different from 1. The subtree rooted at v is made either by (i) one or (ii) more leaves (see Fig. 2).

By the definition of the transition function f, the state s with which the robot could exit w via a port number different from 1 is in the set $\{T, B, EB, L, E\}$.

If either $s = B$ or $s = EB$ is the case, then the same arguments of Lemma 1 hold.

If $s = E$ with $c > 0$ in (i) or $s = E$ in (ii) or $s = L$ is the case, then by construction it is possible to show that such cases cannot occur. This is implied by the fact that the subtree rooted in w is ordered (and labelled) according to the size of the subtrees rooted at the children of w in a non-increasing order. In order to have state $s = E$ with $c > 0$ in (i), the robot should be looking for a leaf of a paired extended leaf, i.e., there must be an extended leaf rooted at the parent of w reachable via a port number smaller than the one that leads to w. This contradicts the labelling procedure since the size of the subtree rooted in w is bigger than an extended leaf. Similarly, in order to have state $s = E$ in (ii), the robot should be looking for paired extended leaves, i.e., there must be an extended leaf rooted at w reachable from w via a port number smaller than the one that leads to v but again this contradicts the labelling procedure since the size of the tree rooted in v is bigger than an extended leaf. Finally, in order to have state $s = L$, the robot should be looking for paired leaves, i.e., there must be a leaf rooted at w reachable from w via a port number smaller than the one that leads to v but this again contradicts the labelling procedure since the size of the tree rooted in v is bigger than a leaf.

If $s = E$ with $c = 0$ in (i), then rules 25, 26 and 29 are sequentially applied, hence the robot exits v in state $s' = EB$ with $c = 0$ via port number 1.

If $s = T$ in (i), then the possible sequences of rules are either 11, 10, 21 or 11, 10, 23, 19, 32 or 11, 11, 12, 14 or 11, 11, 12, 14, 23, 19, 32. This depends on whether $d_G(v) = 2$ and the only node rooted in v in the subtree rooted at v is a leaf in G as well, or $d_G(v) = 2$ and the only node rooted in v in the subtree rooted at v is not a leaf in G, or $d_G(v) > 2$ and the only node rooted in v in the subtree rooted at v is a leaf in G, or $d_G(v) > 2$ and the only node rooted in v in the subtree rooted at v is a not a leaf in G, respectively. In all the above cases the robot exits v in state $s' = EB$ with $c = 0$ via port number 1.

If $s = T$ in (ii), then the possible sequences of rules for the first leaf rooted to v are either 11, 10 or 11, 11, 12, 14. Then, for each paired leaf the robot will apply rules 23, 18 until all the paired leaves rooted in v are visited while saving possible penalties on them. Once the robot is back in v, it may apply rule 22 or the sequence of rules 23, 20, 13 depending on whether $d_{ST}(v) = d_G(v)$ or not. In both the cases the robot exits v in state $s' = N$ via port number 1.

Let us now consider the case $\{w, v\} \equiv e_{1,1}$. w is then the root of $ST(G)$ and v is entered with state $s = FF$. No other states are possible since (apart for the initialisation phase) whenever $e_{1,1}$ is traversed toward the root, the robot backtracks in state FF (rules 15 or 28). In case (i), $ST(G)$ is just a path of length 2 and node v is the middle point with $ht(v) = 2$. P is then defined by either the sequence of rules 28, 5, 10, 21 or the sequence 28, 5, 11, 12, 14, 21 depending on whether $G \equiv ST(G)$ or G is a cycle of length 3. In both cases the robot exits v in state $s' = EB$ with $c = 0$ via port number 1. In case (ii), $ST(G)$ is a star centred in v and $d_{ST}(v) = d_G(v)$. After the first leaf rooted in v is explored by means of the sequence of rules 5, 10 or 5, 11, 11, 12, the sequence of rules 23, 18 is applied at each paired leaf, hence saving possible penalties. Once the robot is back in v from the last paired leaf, it applies rules 22 and leaves v with state $s' = N$ via port number 1. □

We are now ready to formulate the inductive hypothesis in order to prove the theorem.

Inductive Hypothesis. While visiting graph G, every node v at height $ht(v) > 0$ that is not the endpoint of an extended leaf in $ST(G)$ is entered via port number 1 in state T or FF by the robot and left via port number 1 in state N. Moreover, the robot visits every node in the subtree rooted at v while paying at most one penalty at each node but for the nodes belonging to the second, third, etc., extended leaves and for the second, third, etc., leaves rooted to v.

The base of the induction holds from case (ii) of Lemma 2 (see also Fig. 2). We assume the inductive hypothesis true for every node w with $0 < ht(w) \leqslant x$ for some $x > 0$ and we prove it for height $x + 1$. Let node v be such that $ht(v) = x + 1$, let $c_1, c_2, \ldots, c_q$ be its children ordered according to the non-increasing sizes of the corresponding rooted subtrees $T(c_1), T(c_2), \ldots, T(c_q)$, where $T(c_i)$ is the subtree rooted in c_i with $1 \leqslant i \leqslant q$ (see Fig. 2). Let r be the first index for which $T(c_r)$ is an extended leaf, and t be the first index for which $T(c_t)$ is a leaf. We will show that (c) of the theorem holds, namely if v is the root of $k = t - r$ extended paired leaves and $j = q - t + 1$ paired leaves, the robot must pay at most two penalties at $T(c_r)$ and one at $T(c_t)$ while skipping all the other $2(k - 1)$ possible penalties among paired extended leaves, and all the other $j - 1$ possible penalties among paired leaves.

By applying the same arguments of Lemmas 1 and 2, node v is entered from its parent via port number 1 in state T or FF. No other options are possible. The robot will then apply rule 11 for starting the visit of $T(c_1)$. By the inductive hypothesis, the robot visits every node in $T(c_1)$ by skipping penalty payments at every second, third, etc., extended leaf and every second, third, etc., leaf encountered. Then, it goes back from c_1 to v via port number 1 in state N. Subsequently, the robot applies rule 17 in order to start the visit of $T(c_2)$ and so forth until c_r is met. At this point the robot will start the visit of $T(c_r)$ paying the possible penalties and by Lemma 1 it will be back in v via port number 1 in state EB, with $c = 0$. The next subtrees will be visited then by sequentially applying rules 31, 25, 26 and 29 until c_t is entered hence avoiding all the possible penalties at the second, third, etc., paired extended leaves rooted in v. Now the robot will start the visit of $T(c_t)$ paying the possible penalty and by Lemma 2 it will be back in v via port number 1 in state LB. The next subtrees will be visited then by sequentially applying rules 23 and 18 until the robot is back in v from c_q hence avoiding all the possible penalties at the second, third, etc., paired leaves rooted in v. Once entered in v via port number $q + 1$ coming from c_q via port number 1 and rule 18, either $d_G(v) = d_{ST}(v) = q + 1$ and rule 22 is applied or the sequence of rules 23, 13 is applied. In both cases the robot will exit node v via port number 1 in state N and the claim holds. □

Concerning the length π of P, in the next section we provide an upper bound to the number of penalties that our robot might pay during the exploration.

4. Analysis

In this section, we show our upper bound on the length of the periodic exploration of the graph G. Our analysis uses the colouring of the nodes of G introduced in Section 2. Let *RED* be the set of *Red* nodes. We remind the reader that the robot is said to pay a *penalty* at node v if it traverses an edge, starting from v, that does not belong to $ST(G)$. The main goal is to give a bound on the number of penalties in order to prove our result.

Theorem 2. *The period length $\pi(n)$ needed by the robot described in Section 3 to visit a graph G of n nodes, according to the spanning tree $ST(G)$ with the assigned port numbering, is less than $3.75n - 2$.*

Proof. Consider $v \in RED$. By construction, by removing the edges of the backbone connected to v, the remaining subtree T' rooted at v has height at most 2. We decompose the children of v into 3 different types (see Fig. 3):

- Type L: Leaves.
- Type E: Endpoints of extended leaves.
- Type A: Neither leaves nor endpoints of extended leaves.

Let $s_L(v)$ (respectively $s_E(v)$) be 0 if the number of children of v of type L (respectively E) is 0, 1 otherwise. Let $s_A(v)$ be the number of children of v of type A. For all the children of type L, the robot might incur at most 1 penalty, since all those children (if any) are paired leaves and the robot pays only at the first one (see Section 3). For all the children of type E, the robot might incur at most 2 penalties, since all those children (if any) are the endpoints of paired extended leaves and the robot pays only at the first one. For children of type A, by construction, each subtree rooted at a node of type A consists of two or more leaves. This implies that for each subtree rooted at a node of type A the robot might incur at most 2 penalties, i.e., one for the node of type A and the other for the first paired leaf.

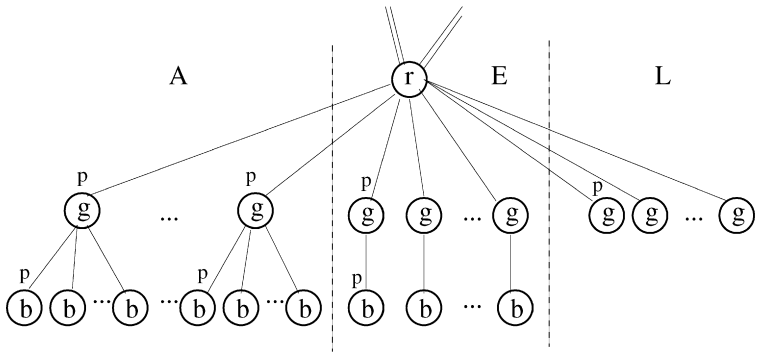

Fig. 3. The subdivision of the children of a red node. The other possible nodes connected to it can only be part of the backbone. The p associated to a node indicates where a penalty might be paid.

Concerning the backbone, let S_Y (respectively, S_O, S_R) be the number of *Yellow* (respectively, *Orange*, *Red*) nodes in the tree. Consider a *Yellow* node w, and its *Blue* children. Note that there can be at most 2 penalties in this subtree. In fact, the *Blue* children of w are paired leaves in this subtree, hence the robot does not pay penalties from the second paired leaf. Consider an *Orange* node z. Note that z has no *Blue* children in the tree, hence there can be at most 1 penalty in this subtree (made up of z itself). Concerning *Red* nodes, no penalties are paid on them, since by construction they are used with their full degree in $ST(G)$.

Let

$$S_L = \sum_{v \in RED} s_L(v), \qquad S_E = \sum_{v \in RED} s_E(v), \qquad S_A = \sum_{v \in RED} s_A(v),$$

and p be the number of nodes with penalties in the tree. As a consequence of the observations above, we have

$$p \leqslant 2S_A + 2S_E + S_L + 2S_Y + S_O,$$
$$n \geqslant 3S_A + 2S_E + S_L + 2S_Y + S_O + S_R.$$

As the backbone tree has the property that between two *Red* nodes there are exactly two nodes (each of them *Yellow* or *Orange*), we have

$$S_Y + S_O = 2S_R - 2.$$

Moreover, from the previous discussion, it also follows that

$$S_L \leqslant S_R, \qquad S_E \leqslant S_R,$$

because for each *Red* node, at most one child of type L (respectively E) is counted in S_E (respectively S_L). Hence, we can bound the number of penalties with respect to the total number of nodes as follows:

$$\begin{aligned}
\frac{p}{n} &\leqslant \frac{2S_A + 2S_E + S_L + 2S_Y + S_O}{3S_A + 2S_E + S_L + 2S_Y + S_O + S_R} \leqslant \frac{2S_A + 2S_E + S_L + 2(S_Y + S_O)}{3S_A + 2S_E + S_L + 2(S_Y + S_O) + S_R} \\
&< \frac{2S_A + 2S_E + S_L + 4S_R}{3S_A + 2S_E + S_L + 4S_R + S_R} \quad [S_Y + S_O = 2S_R - 2] \\
&\leqslant \frac{2 \cdot 0 + 2S_R + S_R + 4S_R}{3 \cdot 0 + 2S_R + S_R + 4S_R + S_R} \quad [S_A \geqslant 0, S_E, S_L \leqslant S_R] \\
&\leqslant \frac{7}{8}.
\end{aligned}$$

By construction of the automaton, whenever a penalty is paid, i.e., whenever a non-tree edge is traversed, the robot incurs another penalty because it has to traverse again the same edge backward. As the automaton traverses every tree

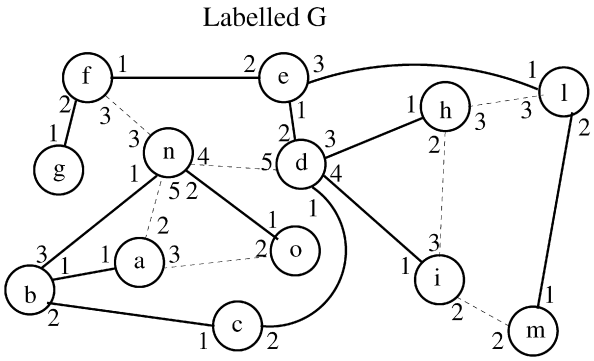

Fig. 4. Labelled $ST(G)$, node "i" is assumed to be the one from which the exploration of the robots starts.

edge twice (once in each direction) it follows that

$$\pi(n) = 2(n-1) + 2p < 2n - 2 + 2 \cdot \frac{7}{8}n = 3.75n - 2. \qquad \square$$

5. Conclusion

In this paper, we studied the problem of periodic graph exploration by means of a simple robot that uses constant memory. We disproved the conjecture given in [20] claiming that it was not possible to obtain a period less than $4n - O(1)$. The new proved upper bound is in fact $3.75n - 2$, hence the gap with the lower bound of $2n - 2$ is reduced. The improvement is obtained by means of a careful construction of the route (i.e., the spanning tree) that the robot has to follow during the graph exploration, and by designing a smart automaton (still using constant memory) able to recognise some specific subtrees.

The main open problem left is whether it is possible to further close the gap of the bounds. For the tree-based approach, a more powerful robot able to recognise more structured topologies (of a constant number of nodes) than just our paired leaves and extended leaves may be helpful in decreasing the number of penalties. Moreover, a modified tree construction and/or port numbering may still lead to further improvements. Another interesting question is if there is a better strategy than tree-based exploration.

Acknowledgments

The authors wish to thank the anonymous referees for their helpful suggestions.

Appendix A. Running example

In this section we compare the exploration algorithm defined in [20] with ours in this paper. The graph in Fig. 4 is repeated from Fig. 1, with our spanning tree $ST(G)$ indicated as the solid edges. We assume that the robot starts at node "i" in both algorithms.

Table 3 shows the steps performed by the robot in each algorithm. The first few steps are devoted to the initialisation phase in each algorithm. The robot of [20] starts with state T and exits via port number 1. The initial state of our robot is instead fixed to be I but the first move is also to exit through port number 1. Our robot requires four steps in this setting to start the real exploration while the Ilcinkas' one needs two steps. We note that on this proposed graph our robot performs a periodic walk P of eight steps shorter than the one performed by the Ilcinkas robot. This comes from the fact that $ST(G)$ contains two paired leaves (nodes "i" and "h") and two paired extended leaves (paths $\{e, l\}\{l, m\}$ and $\{e, f\}\{f, g\}$). Moreover, during the exploration phase our robot does not pay any penalty on the root "a" of $ST(G)$ since it reverses its direction as soon as edge $e_{1,1}$ is crossed. Hence, nodes "a," "i," "l" and "m" are the ones at which we can avoid penalties which account for the shorter exploration length of our method.

Table 3
The exploration performed by the robot in the graph G of Fig. 4 starting from node "i"

	Ilcinkas [20]						Our							
		f(state, i, d)			Output			f(state, i, d, c)					Output	
Step	Node	State	i	d	State	Port	Node	State	i	d	c	State	Port	c
							Initialisation phase							
0	i				T	1	i					I	1	...
1	d	T	4	5	B	4	d	I	4	5	0	T	4	0
2	i	B	1	3	N	1	i	T	1	3	0	T	2	0
3							m	T	$\neq 1$	...	0	B	i	1
4							i	B	2	3	1	LB	1	1
							Exploration phase							
1	d	N	4	5	T	5	d	LB	4	5	1	L	5	1
2	n	T	$\neq 1$	...	B	i	n	L	$\neq 1$	...	1	EB	i	1
3	d	B	5	5	N	1	d	EB	5	5	1	N	1	0
4	c	N	2	2	N	1	c	N	2	2	0	N	1	0
5	b	N	2	3	T	3	b	N	2	3	0	T	3	0
6	n	T	1	5	T	2	n	T	1	5	0	T	2	0
7	o	T	1	2	T	2	o	T	1	2	0	T	2	0
8	a	T	$\neq 1$	...	B	i	a	T	$\neq 1$	...	0	B	i	1
9	o	B	2	2	N	1	o	B	2	2	1	LB	1	1
10	n	N	2	5	T	3	n	LB	2	5	1	L	3	1
11	f	T	$\neq 1$	...	B	i	f	L	$\neq 1$	...	1	EB	i	1
12	n	B	3	5	N	1	n	EB	3	5	1	EB	1	0
13	b	N	3	3	N	1	b	EB	3	3	0	N	1	0
14	a	N	1	3	T	2	a	N	1	3	0	FF	1	0
15	n	T	$\neq 1$	...	B	i	b	FF	1	3	0	T	2	0
16	a	B	2	3	N	1	c	T	1	2	0	T	2	0
17	b	N	1	3	T	2	d	T	1	5	0	T	2	0
18	c	T	1	2	T	2	e	T	1	3	0	T	2	0
19	d	T	1	5	T	2	f	T	1	3	0	T	2	0
20	e	T	1	3	T	2	g	T	1	1	0	LB	1	1
21	f	T	1	3	T	2	f	LB	2	3	1	L	3	1
22	g	T	1	1	N	1	n	L	$\neq 1$	...	1	EB	i	1
23	f	N	2	3	T	3	f	EB	3	3	1	EB	1	0
24	n	T	$\neq 1$	...	B	i	e	EB	2	3	0	E	3	0
25	f	B	3	3	N	1	l	E	1	3	0	E	2	1
26	e	N	2	3	T	3	m	E	1	2	1	EB	1	2
27	l	T	1	3	T	2	l	EB	2	3	2	EB	1	0
28	m	T	1	2	T	2	e	EB	3	3	0	N	1	0
29	i	T	$\neq 1$	...	B	i	d	N	2	5	0	T	3	0
30	m	B	2	2	N	1	h	T	1	3	0	T	2	0
31	l	N	2	3	T	3	i	T	$\neq 1$	...	0	B	i	1
32	h	T	$\neq 1$	...	B	i	h	B	2	3	1	LB	1	1
33	l	B	3	3	N	1	d	LB	3	5	1	L	4	1
34	e	N	3	3	N	1	i	L	1	3	1	LB	1	2
35	d	N	3	5	T	3								
36	h	T	1	4	T	2								
37	i	T	$\neq 1$	...	B	i								
38	h	B	2	4	N	1								
39	d	N	3	5	T	4								
40	i	T	1	3	T	2								
41	m	T	$\neq 1$	...	B	i								
42	i	B	2	3	N	1								

Until step 13 of the exploration phase, both the robots visit the same sequence of nodes.

References

[1] S. Albers, M.R. Henzinger, Exploring unknown environments, SIAM J. Comput. 29 (2000) 1164–1188.

[2] B. Awerbuch, M. Betke, R. Rivest, M. Singh, Piecemeal graph exploration by a mobile robot, Inform. and Comput. 152 (2) (1999) 155–172.
[3] B. Awerbuch, S.G. Kobourov, Polylogarithmic-overhead piecemeal graph exploration, in: Proc. 11th Annual Conference on Computational Learning Theory, COLT 1998, pp. 280–286.
[4] M.A. Bender, A. Fernandez, D. Ron, A. Sahai, S. Vadhan, The power of a pebble: Exploring and mapping directed graphs, Inform. and Comput. 176 (1) (2002) 1–21.
[5] M.A. Bender, D. Slonim, The power of team exploration: Two robots can learn unlabeled directed graphs, in: Proc. 35th Ann. Symp. on Foundations of Computer Science, FOCS 1994, pp. 75–85.
[6] M. Betke, R. Rivest, M. Singh, Piecemeal learning of an unknown environment, Mach. Learn. 18 (1995) 231–254.
[7] L. Budach, Automata and labyrinths, Math. Nachr. 86 (1978) 195–282.
[8] R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, D. Peleg, Label-guided graph exploration by a finite automaton, in: Proc. 32nd International Colloquium on Automata, Languages and Programming, ICALP 2005, in: Lecture Notes in Comput. Sci., vol. 3580, Springer, Berlin, 2005, pp. 335–346.
[9] S. Cook, C. Rackoff, Space lower bounds for maze threadability on restricted machines, SIAM J. Comput. 9 (3) (1980) 636–652.
[10] X. Deng, C.H. Papadimitriou, Exploring an unknown graph, J. Graph Theory 32 (1999) 265–297.
[11] K. Diks, P. Fraigniaud, E. Kranakis, A. Pelc, Tree exploration with little memory, J. Algorithms 51 (2004) 38–63.
[12] S. Dobrev, J. Jansson, K. Sadakane, W.-K. Sung, Finding short right-hand-on-the-wall walks in graphs, in: Proc. 12th Colloquium on Structural Information and Communication Complexity, SIROCCO 2005, in: Lecture Notes in Comput. Sci., vol. 3499, Springer, Berlin, 2005, pp. 127–139.
[13] C.A. Duncan, S.G. Kobourov, V.S.A. Kumar, Optimal constrained graph exploration, in: Proc. 12th Ann. ACM–SIAM Symp. on Discrete Algorithms, SODA 2001, pp. 807–814.
[14] R. Fleischer, G. Trippen, Exploring an unknown graph efficiently, in: Proc. 13th Annual European Symposium on Algorithms, ESA 2005, in: Lecture Notes in Comput. Sci., vol. 3669, Springer, Berlin, 2005, pp. 11–22.
[15] P. Fraigniaud, D. Ilcinkas, Digraphs exploration with little memory, in: Proc. 21st Annual Symposium on Theoretical Aspects of Computer Science, STACS 2004, in: Lecture Notes in Comput. Sci., vol. 2996, Springer, Berlin, 2004, pp. 246–257.
[16] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, D. Peleg, Graph exploration by a finite automaton, Theoret. Comput. Sci. 345 (2005) 331–344.
[17] P. Fraigniaud, D. Ilcinkas, S. Rajsbaum, S. Tixeuil, The reduced automata technique for graph exploration space lower bounds, in: Essays in Memory of Shimon Even, Theoretical Computer Science 2006, in: Lecture Notes in Comput. Sci., vol. 3895, Springer, Berlin, 2006, pp. 1–26.
[18] L. Gąsieniec, R. Klasing, R. Martin, A. Navarra, X. Zhang, Fast periodic graph exploration with constant memory, in: Proc. 14th Colloquium on Structural Information and Communication Complexity, SIROCCO 2007, in: Lecture Notes in Comput. Sci., vol. 4474, Springer-Verlag, 2007, pp. 26–40.
[19] L. Gąsieniec, A. Pelc, T. Radzik, X. Zhang, Tree exploration with logarithmic memory, in: Proc. 18th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2007.
[20] D. Ilcinkas, Setting port numbers for fast graph exploration, in: Proc. 13th Colloquium on Structural Information and Communication Complexity, SIROCCO 2006, in: Lecture Notes in Comput. Sci., vol. 4056, Springer, Berlin, 2006, pp. 59–69.
[21] P. Panaite, A. Pelc, Exploring unknown undirected graphs, J. Algorithms 33 (1999) 281–295.
[22] O. Reingold, Undirected ST-connectivity in log-space, in: Proc. 37th ACM Symposium on Theory of Computing, STOC 2005, pp. 376–385.
[23] H. Rollik, Automaten in planaren Graphen, Acta Inform. 13 (1980) 287–298.