Robustness of the rotor-router mechanism

Evangelos Bampas[1], Leszek Gąsieniec[2,⋆], Ralf Klasing[3], Adrian Kosowski[4], and Tomasz Radzik[5]

[1] School of Elec. & Comp. Eng., National Technical University of Athens, Greece
`ebamp@cs.ntua.gr`
[2] Dept of Computer Science, Univ. of Liverpool, UK
`L.A.Gasieniec@liverpool.ac.uk`
[3] LaBRI, CNRS / INRIA / Univ. of Bordeaux, France[⋆⋆]
`ralf.klasing@labri.fr`
[4] Dept of Algorithms and System Modeling, Gdańsk Univ. of Technology, Poland
`adrian@kaims.pl`
[5] Dept of Computer Science, King's College London, UK
`tomasz.radzik@kcl.ac.uk`

Abstract. We consider the model of exploration of an undirected graph G by a single *agent* which is called the *rotor-router mechanism* or the *Propp machine* (among other names). Let π_v indicate the edge adjacent to a node v which the agent took on its last exit from v. The next time when the agent enters node v, first a "rotor" at node v advances pointer π_v to the edge $next(\pi_v)$ which is next after the edge π_v in a fixed cyclic order of the edges adjacent to v. Then the agent is directed onto edge π_v to move to the next node. It was shown before that after initial $O(mD)$ steps, the agent periodically follows one established Eulerian cycle, that is, in each period of $2m$ consecutive steps the agent traverses each edge exactly twice, once in each direction. The parameters m and D are the number of edges in G and the diameter of G. We investigate robustness of such exploration in presence of faults in the pointers π_v or dynamic changes in the graph. We show that after the exploration establishes an Eulerian cycle,

(i) if at some step the values of k pointers π_v are arbitrarily changed, then a new Eulerian cycle is established within $O(km)$ steps;
(ii) if at some step k edges are added to the graph, then a new Eulerian cycle is established within $O(km)$ steps;
(iii) if at some step an edge is deleted from the graph, then a new Eulerian cycle is established within $O(\gamma m)$ steps, where γ is the smallest number of edges in a cycle in graph G containing the deleted edge.

Our proofs are based on the relation between Eulerian cycles and spanning trees known as the "BEST" Theorem (after de **B**ruijn, van Aardenne-**E**hrenfest, **S**mith and **T**utte).

Key words: Graph exploration, Rotor-router mechanism, Propp machine, Network faults, Dynamic graphs.

⋆ Partially funded by the Royal Society International Joint Project, IJP - 2007/R1.
⋆⋆ Additional support by the ANR projects ALADDIN and IDEA and the INRIA project CEPAGE.

1 Introduction

We investigate robustness of the single-agent exploration of an undirected connected graph G based on the *rotor-router mechanism.* In this model of graph exploration the agent has no operational memory and the whole routing mechanism is provided within the environment. The edges adjacent to each node v are arranged in a fixed cyclic order, which does not change during the exploration. Each node v maintains a *port pointer* π_v which indicates the edge traversed by the agents on its *last* exit from v. If the agent has not visited node v yet, then π_v points to the initial arbitrary edge adjacent to v. The next time when the agent enters node v, first the port pointer π_v is advanced to the edge *next*(π_v) which is next after the edge π_v in the cyclic order of the edges adjacent to v, and then the agent is directed onto edge π_v to move to the next node. This is *one step* of the exploration. We can think about the process of advancing the port pointer π_v as if there was a "rotor" at node v moving pointer π_v around the cyclic order of the edges adjacent to v, hence the name the *rotor-router mechanism.* This model was introduced by Priezzhev *et al.* [11], was further studied and popularised by James Propp, and now is also referred to as the *Propp machine.*

Wagner *et al.* [14, 15] showed that in this model, starting from an arbitrary configuration (arbitrary cyclic orders of edges, arbitrary initial values of the port pointers and an arbitrary starting node) the agent covers all edges of the graph within $O(nm)$ steps, where n and m are the number of nodes and the number of edges in the graph. Bhatt *et al.* [2] showed later that within $O(nm)$ steps the agent not only covers all edges but actually *enters (establishes) an Eulerian cycle.* More precisely, after the initial *stabilisation period* of $O(nm)$ steps, the agent keeps repeating the same Eulerian cycle of the directed graph $\boldsymbol{G}$ which is the directed symmetric version of graph G. Graph $\boldsymbol{G}$ contains two opposite arcs (v, u) and (u, v) for each edge $\{v, u\}$ in G. Subsequently Yanovski *et al.* [16] showed that the agent enters an Eulerian cycle within $2mD$ steps, where D is the diameter of the graph.

It has been frequently mentioned in the previous work that a useful property of graph exploration based on the rotor-router mechanism is its robustness. In case of link failures or other dynamic changes in the graph, after some additional stabilisation period the agent goes back into the regime of repeatedly traversing the graph along a (new) Eulerian cycle. We know that whatever the changes in the graph are, the length of that additional stabilisation period is $O(mD)$ (as shown in [16], that much time is sufficient for establishing an Eulerian cycle from *any* initial configuration) but no better bounds have been shown before.

Our results. In this paper we develop bounds on the length of that additional stabilisation period. These bounds depend on the extent of the failures or changes in the graph. Thus we assume that an Eulerian cycle has been already established and show the following.

(i) **Faults in port pointers.** If at some step the values of k pointers π_v are changed to arbitrary edges (that is, the value of π_v is changed to an arbitrary edge adjacent to node v), then a new Eulerian cycle is established within $O(m \min\{k, D\})$ steps.

(ii) **Addition of new edges.** If at some step k edges are added to the graph, then a new Eulerian cycle is established within $O(m \min\{k, D\})$ steps.

(iii) **Deletion of an edge.** If at some step an edge is deleted from the graph but the graph remains connected, then a new Eulerian cycle is established within $O(\gamma m)$ steps, where γ is the smallest number of edges in a cycle in graph G containing the deleted edge.

A faulty change of the value of the port pointer π_v at a node v might occur when something unexpected makes the node believe that π_v should be re-set to some default value. We assume that when a new edge $\{u, v\}$ is added, it is inserted in arbitrary places in the existing cyclic orders of edges adjacent to nodes u and v, but otherwise those cyclic orders remain as they were before. Similarly, when an edge $\{u, v\}$ is deleted, the cyclic orders of the remaining edges adjacent to nodes u and v remain as they were. On both addition and deletion of an edge $\{v, u\}$, we allow arbitrary changes of the values of the port pointers at nodes v and u. A concrete system would specify some default updates for the port pointers on insertion or deletion of an edge, but for our results we do not need to make any assumptions about those defaults.

Regarding our $O(\gamma m)$ bound for the case of deleting an edge, we note that there are non-trivial classes of graphs (e.g., random graphs) in which each edge belongs to a short cycle. For such graphs parameter γ is small and our bound implies that the additional stabilisation period is short.

Previous work. The previous work which is most directly relevant to our paper are Bhatt *et al.* [2] and Yanovski *et al.* [16], both already mentioned above. Bhatt *et al.* [2] considers also mechanisms enabling the agent to stop after exploring the whole graph. Yanovski *et al.* [16], in addition to proving the $2mD$ bound on the length of the stabilisation period, show also that this bound is asymptotically optimal in the worst-case, and study the case when there are $k \geq 2$ agents. Regarding the terminology, we note that the graph exploration model based on the rotor-router mechanism which we consider in this paper is called the *Edge Ant Walk algorithm* in [14–16], while the same model is described in [2] in terms of traversing a maze and marking edges with pebbles.

The rotor-router mechanism is the strategy of leaving a node v along the edge for which the most time has elapsed since its last traversal *in the direction from* v. Cooper *et al.* [4] consider an *undirected* variant of this *oldest-first* strategy which chooses the edge for which the most time has elapsed since its last traversal *in any direction*. They show that this undirected oldest-first strategy leads in the worst case to exponential cover time.

The rotor-router mechanism has been often studied as a deterministic analogue of the *random walk* on a graph, with the main objective of discovering similarities and differences between these two processes. In the context of balancing the workload in a network, the single agent is replaced with a number of agents, referred to as *tokens*. Cooper and Spencer [5] study d-dimensional grid graphs and show a constant bound on the difference between the number of tokens at a given node v in the rotor-router model and the expected number of tokens at v in the random-walk model. Subsequently Doerr and Friedrich [7] analyse in more detail the distribution of tokens in the rotor-router mechanism on the 2-dimensional grid.

The research area of graph exploration with simple agents (robots) is rich in models and approaches. Exploration with robots with bounded memory has been considered for example in [8, 9, 12]. Models which allow placement of some identifiers or markers on nodes or edges of the graph have been considered for example in [1, 6]. Some graph exploration techniques are surveyed in [10].

Our analysis of the rotor-router mechanism is based on the relationship between the Eulerian cycles in the directed graph $\boldsymbol{G}$ and the spanning trees in the undirected graph G which underlies the following theorem. This theorem, sometimes referred to as the "BEST" theorem, was discovered by de **B**ruijn and van Aardenne-**E**hrenfest [3] on the basis of earlier work by **S**mith and **T**utte [13].

Theorem 1 (Bruijn, van Aardenne-Ehrenfest, Smith, Tutte). *The number of Eulerian cycles in the directed, symmetric version of an undirected connected graph $G = (V, E)$ is equal to $\prod_{v \in V}(d(v) - 1)!$ times the number of spanning trees of G, where $d(v)$ is the degree of node v in G.*

In Section 2 we establish the terminology and notation used in this paper and give the basic properties of exploration based on the rotor-router mechanism. In Section 3 we describe the connection between the Eulerian cycles and spanning trees in the context of the rotor-router mechanism. In passing we show how Theorem 1 follows from our analysis of the rotor-router mechanism. In Section 4 we investigate in more detail the stabilisation period of exploration with the rotor-router mechanism. The analysis developed in Sections 3 and 4 culminates in Theorem 3, which can be viewed as a quantitative description of the progress of stabilising the exploration, that is, the progress towards establishing an Eulerian cycle. In Section 5 we give our bounds on the length of the additional stabilisation period after some failures or changes in the graph have occurred. All these results are simple consequences of Theorem 3. We point out that all of the obtained bounds are asymptotically tight in the worst case.

2 The Rotor-router model

Let $G = (V, E)$ be an undirected connected graph with n nodes, m edges and diameter D. The directed graph $\boldsymbol{G} = (V, \boldsymbol{E})$ is the directed symmetric version of G, where the set of arcs $\boldsymbol{E} = \{(v, u), (u, v) : \{v, u\} \in E\}$. We will refer to the undirected links in graph G as *edges* and to the directed links in graph $\boldsymbol{G}$ as *arcs*. We will also keep using boldface symbols, as $\boldsymbol{G}$ and $\boldsymbol{E}$, to stress that we refer to directed graphs and arcs. For a node $v \in V$, $d(v)$ denotes the degree of v in G.

We consider the rotor-router model (on graph G) with a single agent. The agent moves in discrete steps from node to node along the arcs of graph $\boldsymbol{G}$. A *configuration* at the current step is a triple

$$((\rho_v)_{v\in V},\,(\pi_v)_{v\in V},\,r),$$

where ρ_v is a cyclic order of the arcs (in graph $\boldsymbol{G}$) outgoing from node v, π_v is an arc outgoing from node v, which is referred to as *the (current) port pointer at node* v, and r is *the current node* – the node where the agent is at the current step. For each node $v \in V$, the cyclic order ρ_v of the arcs outgoing from v is fixed at the beginning of exploration and does not change in any way from step to step (unless an edge is dynamically added or deleted as discussed in the previous section). For an arc (v, u), let $\mathit{next}(v, u)$ denote the arc next after arc (v, u) in the cyclic order ρ_v.

During the current step, first the port pointer π_r at the current node r is advanced to the next arc outgoing from r (that is, π_r becomes $\mathit{next}(\pi_r)$), and then the agent moves from node r traversing the arc π_r. The exploration starts from some initial configuration and then keeps running without ever terminating. We consider in this paper the rotor-router model as a mechanism for exploration of a graph, so the most interesting questions for us are how quickly the agent explores the whole graph, and how evenly it keeps traversing the edges of the graph. The following two simple lemmas will be used in later analysis.

Lemma 1. *The agent visits each node infinitely many times (thus traverses each arc infinitely many times).*

Proof. If a node v is visited only finitely many times, then each node in the neighbourhood of v is visited only finitely many times. Thus, by induction, each node in the graph is visited only finitely many times, contradicting the assumption that the agent does not terminate. □

Lemma 2. *If in the current step i the agent leaves the current node r along an arc (r, y), then the first arc traversed for the second time during the period $i, i+1, \ldots,$ is this arc (r, y).*

Proof. Let v be the first node which the agent exits $d(v) + 1$ times during the period $i, i+1, \ldots$. That is, the agent exits node v for the first time at some step $j' \geq i$ along an arc (v, u), then it exits v once along all remaining arcs outgoing from v, and then it exits v again along arc (v, u) at some step $j'' > j'$. During the period $i, i+1, \ldots j'' - 1$, for every node $z \in V$, the agent exits z at most $d(z)$ times. Thus the arcs traversed during the period $i, i+1, \ldots, j'' - 1$ are all distinct, and arc (v, u) is the first arc traversed for the second time. If $j' \geq i+1$, then during the period $i, i+1, \ldots j'' - 1$ the agent must enter node v $d(v)+1$ times, because during the period $i+1, i+2, \ldots j''$ the agent leaves node v $d(v) + 1$ times. Hence if $j' \geq i+1$, then there is an arc incoming to node v which is traversed twice during the period $i, i+1, \ldots j'' - 1$. This contradiction implies that $j' = i$, so $(v, u) = (r, y)$. □

3 Trees and Eulerian cycles

If T is a tree in graph G (not necessarily spanning all nodes of G), then $\boldsymbol{T}$ obtained from T by directing all edges towards a selected node v in T is called an *in-bound tree* in $\boldsymbol{G}$, and node v is the root of $\boldsymbol{T}$. A subset of arcs $\boldsymbol{H}$ in $\boldsymbol{G}$ is an *in-bound tree with a root cycle*, if it is an in-bound tree with one additional arc outgoing from the root. That additional arc creates a (directed) cycle, which we call a root cycle. We can view $\boldsymbol{H}$ as consisting of one cycle (the root cycle) and a number of in-bound node-disjoint trees rooted at nodes of this cycle (only the roots of these trees belong to the root cycle).

Let $\boldsymbol{F} = \{\pi_v : v \in V\}$ be the set of the current port pointers. For the current node r, we are interested in the structure of $\boldsymbol{F}_r = \boldsymbol{F} \setminus \{\pi_r\}$, since, as we show later, the structure of $\boldsymbol{F}_r$ is a good indicator of how far the agent is from entering an Eulerian cycle. The component of $\boldsymbol{F}_r$ containing the current node r is an *in-bound tree* rooted at r, which we call *the leading tree.* Each component $\boldsymbol{H}$ of $\boldsymbol{F}_r$ other than the leading tree is an *in-bound tree with a root cycle.*

The following Lemmas 3 and 4 show that the condition that the agent follows an Eulerian cycle and the condition that the leading tree spans all nodes of the graph are equivalent.

Lemma 3. *Assume that the current leading tree $\boldsymbol{T}$ spans all nodes of the graph. Then during the next $2m$ steps the agent traverses an Eulerian cycle in $\boldsymbol{G}$. Moreover, the leading tree after these $2m$ steps is again the same tree $\boldsymbol{T}$.*

Proof. Let r be the current node (and the root of the current leading tree $\boldsymbol{T}$) and let (r, y) be the arc which the agent traverses in the current step. Let $\boldsymbol{\Gamma}$ be the cycle (the sequence of arcs) which the agent follows starting from this traversal of arc (r, y) and ending right before the second traversal of this arc. We show that $\boldsymbol{\Gamma}$ is an Eulerian cycle in $\boldsymbol{G}$. For $u \neq r$, let $p(u)$ be the parent of u in tree $\boldsymbol{T}$, that is, $(u, p(u)) = \pi_u$. From Lemma 2, all arcs on $\boldsymbol{\Gamma}$ are distinct, so it remains to show that $\boldsymbol{\Gamma}$ contains all arcs.

Cycle $\boldsymbol{\Gamma}$ contains all $d(r)$ arcs outgoing from node r: after following cycle $\boldsymbol{\Gamma}$, the agent is about to traverse arc (r, y) again, so it must have already traversed all arcs outgoing from r. This means that $\boldsymbol{\Gamma}$ must also contain all $d(r)$ arcs incoming to r (no arc occurs twice on $\boldsymbol{\Gamma}$), including all arcs $\pi_u = (u, p(u))$ with $p(u) = r$. When cycle $\boldsymbol{\Gamma}$ passes through such an arc $(u, p(u))$, then it must have already passed through all arcs outgoing from node u, since arc $(u, p(u)) = \pi_u$ is the last arc outgoing from node u to be taken by the agent. This further implies that $\boldsymbol{\Gamma}$ contains all arcs incoming to u, including all arcs $(w, p(w))$ with $p(w) = u$. By induction on the distance to node r in tree $\boldsymbol{T}$, for each node v, cycle $\boldsymbol{\Gamma}$ contains all arcs outgoing from v.

Since the agent has traversed an Eulerian cycle, all port pointers are back to what they were before traversing $\boldsymbol{\Gamma}$. Thus the leading tree after traversing $\boldsymbol{\Gamma}$ is the same as it was before traversing $\boldsymbol{\Gamma}$. □

Fig. 1 illustrates Lemma 3. The diagram on the left shows a graph and the current leading tree (arcs in bold) which spans all nodes. The current node

r is the root of this tree. The diagram on the right shows the Eulerian cycle followed by the agent. We assume in this figure that the cyclic order of the arcs outgoing from a node is the anti-clockwise order, and that arc (r, x) is the current value of the port pointer π_r. Thus the first arc followed by the agent is arc $(r, y) = next(r, x)$.

Lemma 4. *Assume that at the current step i the leading tree $\boldsymbol{T}$ does not span all nodes of the graph. Then the route $\boldsymbol{\Gamma}$ traversed by the agent during the next $2m$ steps is not an Eulerian cycle.*

Proof. Consider the (non-empty) set $\boldsymbol{A}$ of the arcs incoming to tree $\boldsymbol{T}$, that is, the arcs with the start nodes outside $\boldsymbol{T}$ and the end nodes in $\boldsymbol{T}$. If after the next $2m$ steps the agent is not back in the starting node r, then $\boldsymbol{\Gamma}$ is not a cycle. Therefore assume that the agent comes back to node r after $2m$ steps. If during these $2m$ steps the agent traverses an arc in $\boldsymbol{A}$ more than once or does not traverse it at all, then obviously $\boldsymbol{\Gamma}$ is not an Eulerian cycle. Therefore assume now that each arc in $\boldsymbol{A}$ is traversed exactly once, and let (v, u) be the arc in $\boldsymbol{A}$ traversed last. Consider the order of the arcs outgoing from node v ending with the arc which is the value of the port pointer π_v at node v at step i: $(v, x_1), (v, x_2), \ldots, (v, x_{d(v)}) = \pi_v$. Arc (v, u) is the last arc in this order which belongs to $\boldsymbol{\Gamma}$, but (v, u) is not the arc π_v: node u is in $\boldsymbol{T}$ but arc π_v does not lead to $\boldsymbol{T}$ (or otherwise node v would belong to the leading tree $\boldsymbol{T}$). Thus $\boldsymbol{\Gamma}$ does not contain arc π_v, so it is not an Eulerian cycle. □

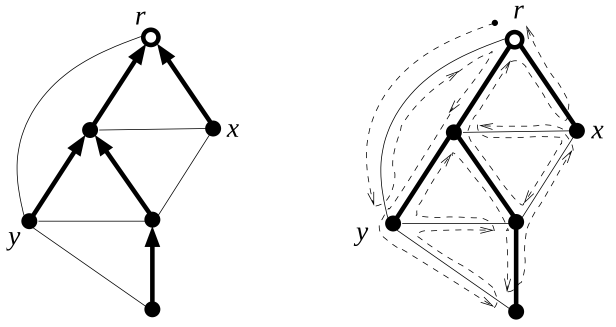

Fig. 1. Left: the leading tree spanning all nodes of the graph (arcs in bold). Right: the corresponding Eulerian cycle, assuming the anti-clockwise order of arcs outgoing from a node (other cycles are obtained for other cyclic orders of arcs).

For the initial configuration $((\rho_v)_{v\in V}, (\pi_v^{init})_{v\in V}, r_{init})$, let

$$\tau = \tau((\rho_v)_{v\in V}, (\pi_v^{init})_{v\in V}, r_{init}) \leq \infty,$$

denote the first step when the leading tree spans all nodes of $\boldsymbol{G}$. We use this parameter τ as our formal definition of the *stabilisation time*, and call these initial τ steps the *stabilisation period*. Lemma 3 implies immediately the following corollary.

Corollary 1. *After the stabilisation period, the agent keeps traversing the same Eulerian cycle.*

We refer to the Eulerian cycle which the agent keeps repeating after the stabilisation period as the *established Eulerian cycle*. Yanovski *et al.* [16] defined t_0 as the first step when all arcs of the graph are traversed and showed the following result.

Theorem 2. [16] *For any graph G, any cyclic order ρ_v of the arcs outgoing from each node $v \in V$, and any initial values of the port pointers π_v, $v \in V$, we have $t_0 \le 2mD$, and from step t_0+1 the agent keeps repeating the same Eulerian cycle of graph $\boldsymbol{G}$.*

The definitions of steps τ and t_0 are somewhat different, but these two steps cannot be far apart. Lemma 3 implies that $t_0 \le \tau+2m$. On the other hand, since from step t_0 the agent follows an Eulerian cycle (Theorem 2), then Lemma 4 implies that $\tau \le t_0$.

We conclude this section by showing the connection between the rotor-router mechanism and Theorem 1. We fix a node r as the current node and an arc (r, x) as the current value of the port pointer π_r (the agent will follow in the current step arc *next*(r, x)). Let $\boldsymbol{T}$ denote an in-bound spanning tree of $\boldsymbol{G}$ rooted at node r, and let ρ_v denote a cyclic order of the arcs in $\boldsymbol{G}$ outgoing from v. Consider the assignment of Eulerian cycles of $\boldsymbol{G}$ to pairs $(\boldsymbol{T}, (\rho_v)_{v\in V})$ given by Lemma 3. More precisely, assigns to a pair $(\boldsymbol{T}, (\rho_v)_{v\in V})$ the Eulerian cycle of $\boldsymbol{G}$ which is followed by the agent starting from the configuration $((\rho_v)_{v\in V}, \boldsymbol{T}\cup\{\pi_r\}, r)$, that is, starting with $\boldsymbol{T}$ as the leading tree. It is easy to verify that the cycles $\boldsymbol{\Gamma}'$ and $\boldsymbol{\Gamma}''$ assigned to two distinct pairs $(\boldsymbol{T}', (\rho'_v)_{v\in V})$ and $(\boldsymbol{T}'', (\rho''_v)_{v\in V})$ are distinct.

We now show that each Eulerian cycle of graph $\boldsymbol{G}$ corresponds to some pair $(\boldsymbol{T}, (\rho_v)_{v\in V})$. For an arbitrary Eulerian cycle $\boldsymbol{\Gamma}$ of $\boldsymbol{G}$, for each $v \in V$, let ρ_v be the cyclic order of the arcs outgoing from a node v defined by the order of these arcs along $\boldsymbol{\Gamma}$. Pick the beginning of cycle $\boldsymbol{\Gamma}$ such that edge (r, x) is the last edge on $\boldsymbol{\Gamma}$. For each node $v \ne r$, let π_v be the last arc on $\boldsymbol{\Gamma}$ outgoing from v. The set of arcs $\boldsymbol{T} = \{\pi_v : v \in V \setminus \{r\}\}$ does not contain a cycle, so it is an in-bound spanning tree of $\boldsymbol{G}$ rooted at node r. The agent starting from the configuration $((\rho_v)_{v\in V}, \boldsymbol{T}\cup\{\pi_r\}, r)$ follows cycle $\boldsymbol{\Gamma}$, so cycle $\boldsymbol{\Gamma}$ is assigned to the pair $(\boldsymbol{T}, (\rho_v)_{v\in V})$.

The above one-to-one correspondence between the Eulerian cycles in $\boldsymbol{G}$ and the pairs $(\boldsymbol{T}, (\rho_v)_{v\in V})$, where $\boldsymbol{T}$ is an in-bound spanning tree of $\boldsymbol{G}$ rooted at node r and ρ_v is a cyclic order of the arcs in $\boldsymbol{G}$ outgoing from v, gives a one-to-one correspondence between the Eulerian cycles in $\boldsymbol{G}$ and the pairs $(T, (\rho_v)_{v\in V})$, where T is a spanning tree in G: identify an in-bound spanning tree of $\boldsymbol{G}$ rooted at node r with the spanning tree of G obtained from $\boldsymbol{T}$ by disregarding the directions of arcs. The existence of a one-to-one correspondence between the Eulerian cycles in $\boldsymbol{G}$ and the pairs $(T, (\rho_v)_{v\in V})$ implies Theorem 1. Indeed, for a node $v \in V$, there are $(d(v)-1)!$ distinct cyclic orders ρ_v. Thus the number of Eulerian cycles in $\boldsymbol{G}$ is equal to $\prod_{v\in V}(d(v)-1)!$ times the number of spanning trees of G, as Theorem 1 states.

4 Evolution of the leading tree

With respect to the set of port pointers $\boldsymbol{F}_r = \boldsymbol{F} \setminus \{\pi_r\}$, where r is the current node, a node v is an *ancestor* of a node u if, and only if, the path in $\boldsymbol{F}_r$ starting from v passes through u. Each node is its own ancestor. If a node v belongs to the leading tree $\boldsymbol{T}$, then the ancestors of v are all nodes on the path in $\boldsymbol{T}$ from v to the root r, including both v and r. If a node v does not belong to the leading tree $\boldsymbol{T}$, then it belongs to a component $\boldsymbol{H}$ of $\boldsymbol{F}_r$ which is an in-bound tree with a root cycle. In this case, the ancestors of v are all nodes on the path in $\boldsymbol{H}$ from v to the cycle and all nodes on the cycle.

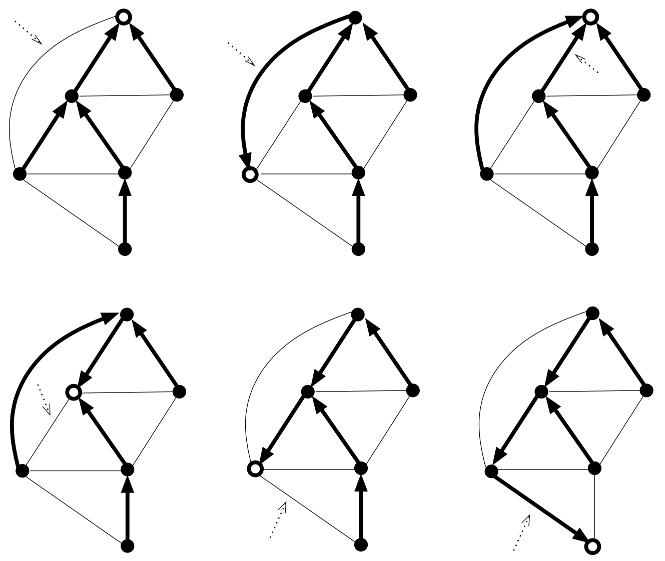

Fig. 2. The changing leading tree when the agent does not go outside the tree. The white node is the current node (and the root of the tree). The dotted arrow indicates the edge to be taken from the current node.

The following Lemmas 5 and 6, which describe changes of the leading tree, can be easily verified.

Lemma 5. *If a node belongs to the current leading tree, then it remains in the leading tree in all subsequent steps.*

Lemma 6. *Let v be a node which is not in the current leading tree. Node v enters the leading tree at the first step when the agent visits an ancestor of v.*

Fig. 2 shows an example how the leading tree changes when the agent does not go outside of the tree. Note that this figure shows only the leading tree, not the whole graph. Fig. 3 illustrates Lemma 6.

Lemma 7 below can be viewed as a generalisation of Lemma 3 to the case when the leading tree does not span all nodes. The neighbourhood of the leading tree $\boldsymbol{T}$ consists of the nodes which are not in $\boldsymbol{T}$ but are adjacent to the nodes in $\boldsymbol{T}$.

Lemma 7. *Each node which is in the current step in the neighbourhood of the leading tree* $\boldsymbol{T}$ *is visited within the next* $2m$ *steps.*

Proof. Let (r, s) be the arc traversed in the current step i, and then traversed again for the second time in a future step $j > i$. Lemma 2 implies that the arcs traversed in steps $i, i+1, \ldots, j-1$ are all distinct, so $j \leq i + 2m$. We can show, similarly as in the proof of Lemma 3, that each arc outgoing from each node in tree $\boldsymbol{T}$ is traversed during the period $i, i+1, \ldots, j-1$. Thus each node v in the neighbourhood of tree $\boldsymbol{T}$ is visited at least once during this period. □

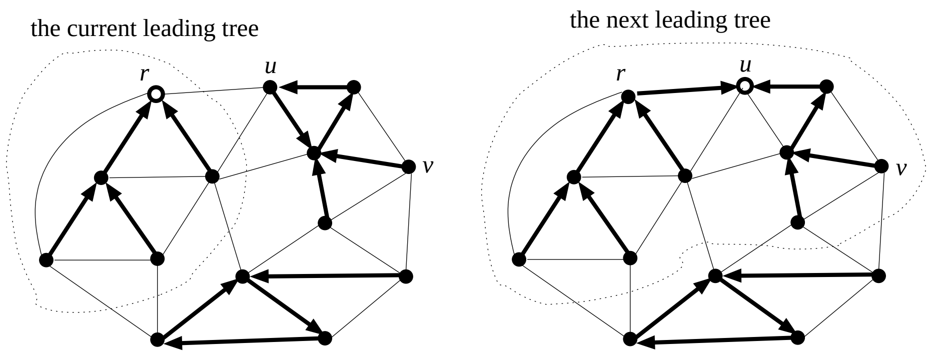

Fig. 3. The port pointers π_x, $x \neq r$, are shown as boldface arrows. Left: the current step, when node v is outside the leading tree. Right: the next step, when the agent visits an ancestor u of v and v enters the leading tree.

Lemmas 6 and 7 imply that a node v which is outside of the leading tree but has an ancestor u in the neighbourhood of the leading tree will enter the leading tree within $2m$ steps. If a node v which is outside of the leading tree has an ancestor u in the neighbourhood of a node w which has an ancestor y in the neighbourhood of the leading tree, then v will enter the leading tree within $4m$ steps (node w will enter the leading tree within $2m$ steps and then node v will enter the leading tree within additional $2m$ steps), and so on. To formalise this, we define the *length of an arc* (v, u) as equal to 0, if $(v, u) \in \boldsymbol{F}_r$, and equal to 1 otherwise. The *distance* from a node v to a node x is the minimum total length of a path from v to x in $\boldsymbol{G}$. Note that the length of an arc and the distance from a node to another node are relative to the current step (and the current values of the port pointers). The distance from a node v to a node x is 0 if, and only if, x is an ancestor of v. Observe also that the distance from one node to any other node is never greater than the diameter D of the graph.

Theorem 3. *If the distance from a node v to the current node r is equal to k, then node v enters the leading tree within $2km$ steps.*

Proof. The proof is by induction on k. If the distance from a node v to the current node r is 0, then there is a path from v to r consisting of port pointers (arcs with length 0), so node v is in the leading tree already in the current step.

If the distance from a node v to the current node r is $k \geq 1$, then a shortest path from v to r (with respect to the current lengths of the arcs) follows first port pointers from v to an ancestor u of v (zero or more arcs of length 0), and then follows an arc (u, w) to a neighbour w of u which is not an ancestor of v (an arc of length 1). The distance from node w to the current node r is $k-1$, so by the inductive hypothesis, node w enters the leading tree within $2(k-1)m$ steps. Thus node u is in the neighbourhood of the leading tree within $2(k-1)m$ steps, so Lemma 7 implies that node u is visited within $2km$ steps. This and Lemma 6 imply that node v enters the leading tree within $2km$ steps. □

We note that Theorem 3 gives an alternative proof of the $O(mD)$ bound shown in [16] on the number of steps required in the rotor-router model to enter an Eulerian cycle. Recall that for any configuration of the rotor-router mechanism and any node v, the distance from v to the current node (w.r.t. the lengths of the arcs) is at most D. Using Theorem 3 we conclude that all nodes enter the leading tree within $2mD$ steps.

5 Faulty port pointers and dynamic changes of the graph

In this section we give bounds on the number of steps needed to establish a new Eulerian cycle when some changes in the graph have occurred. All these bounds follow from Theorem 3. A spontaneous (faulty) change of the value of the port pointer π_v is a change to an arbitrary arc outgoing from node v.

After the stabilisation period, inserting or deleting an edge $\{v, u\}$ may be harmless if this operation does not change the port pointers at nodes v and u. If these port pointers remain as they were, then the leading tree does not change, so it continues spanning all nodes and a new Eulerian cycle is established immediately. However, recall from Section 1 that we assume that insertion or deletion of an edge $\{v, u\}$ may cause arbitrary changes of the values of the port pointers at nodes v and u. Recall also that we assume that the cyclic orders of the edges adjacent to nodes v and u excluding edge $\{v, u\}$ are the same after the insertion/deletion as they were before.

Theorem 4. *In the rotor-router model, after the stabilisation period, if k port pointers spontaneously change their values at some step, then a new Eulerian cycle is established within $2m \min\{k, D\}$ steps.*

Proof. Consider the leading tree right before those k changes of the port pointers. The stabilisation period has passed, so the leading tree spans all nodes. For each node $x \in V$, the length of the path P in the leading tree from x to the current node is equal to 0. When k port pointers change their values, then at most k arcs on path P change length from 0 to 1. This means that the new length of P is at most k, so the distance from x to the current node is at most k, and this distance is never greater than D. Thus Theorem 3 implies that all nodes in the graph will be back in the leading tree within $2m \min\{k, D\}$ steps. □

Theorem 5. *In the rotor-router model, after the stabilisation period, if k new edges are added to the graph at some step, then an Eulerian cycle in the expanded graph is established within $2m \min\{2k, D\}$ steps.*

Proof. Adding k edges may result in changes of the values of up to $2k$ port pointers, so Theorem 4 implies that an Eulerian cycle is established in the expanded graph within $2m \min\{2k, D\}$ steps. □

Theorem 6. *In the rotor-router model, after the stabilisation period, if at some step an edge $\{v, u\}$ is removed from the graph but without disconnecting it, then an Eulerian cycle in the new graph is established within $2\gamma m$ steps, where γ is the smallest number of edges on a cycle in G containing edge $\{v, u\}$.*

Proof. The removal of an edge $\{v, u\}$ from the graph may change the port pointers at nodes v and u. Similarly as in the proof of Theorem 4, consider the leading tree right before this edge removal. For each node $x \in V$, the length of the path P in the leading tree from x to the current node is equal to 0. When edge $\{v, u\}$ is removed, then two arcs on path P may change their length from 0 to 1, and if arc (v, u) or arc (u, v) belongs to P, then we replace this arc with the $\gamma - 1$ arcs from a shortest cycle in $\boldsymbol{G}$ containing this arc. The length of the new path from x to the current node is at most γ (at most γ arcs have length 1), so the distance from x to the current node is at most γ. Thus Theorem 3 implies that all nodes in the graph are back in the leading tree within $2\gamma m$ steps. □

The bounds which appear in Theorems 4, 5, and 6 are all asymptotically tight in the worst case. Indeed, for some values of parameters s and d, consider the lollipop graph $G_{s,d}$ obtained by merging a vertex r of the clique K_s with an end-vertex of the path P_d (cf. Fig. 4a). Let the agent be located at vertex r after the stabilization of the rotor-router. When k port pointers are altered at internal nodes of path P_d ($k < d$), the rotor-router will only stabilize to a new Eulerian cycle after visiting each of the edges of the clique at least k times. Hence, for any feasible set of parameters n, m, k, D there exists a graph with $\Theta(n)$ nodes, $\Theta(m)$ edges and diameter $\Theta(D)$, such that restoring the stable state of the rotor-router after modification of k port pointers requires $\Omega(m \min\{k, D\})$ steps. Thus, the bound in Theorem 4 is asymptotically tight in the worst case.

Likewise, by the construction shown in Fig. 4b, we obtain a worst-case lower bound of $\Omega(m \min\{k, D\})$ steps for the stabilization period after adding k new edges to the graph, asymptotically matching the bound in Theorem 5. Note that the addition of edges to the graph may by assumption result in modifications to pointer arrangements at the endpoints of added edges. Finally, Fig. 4c provides an example of a scenario in which removing a single edge leads to a stabilization period of $\Omega(\gamma m)$, asymptotically matching the bound in Theorem 6.

6 Conclusions

In this paper we have presented a quantitative evaluation of the robustness of the graph exploration based on the rotor-router mechanism. Our bounds on the length of the additional stabilisation period, required after some faults or changes have occurred in the graph, are asymptotically tight in the worst-case.

Our analysis can be applied to other possible models of faults and dynamic changes in the graph. For example, one may observe that our analysis implies that the rotor-router mechanism tolerates spontaneous changes of the cyclic orders of the edges. More precisely, if at some step after the stabilisation period the cyclic orders of edges change in any way but the port pointers remain the same

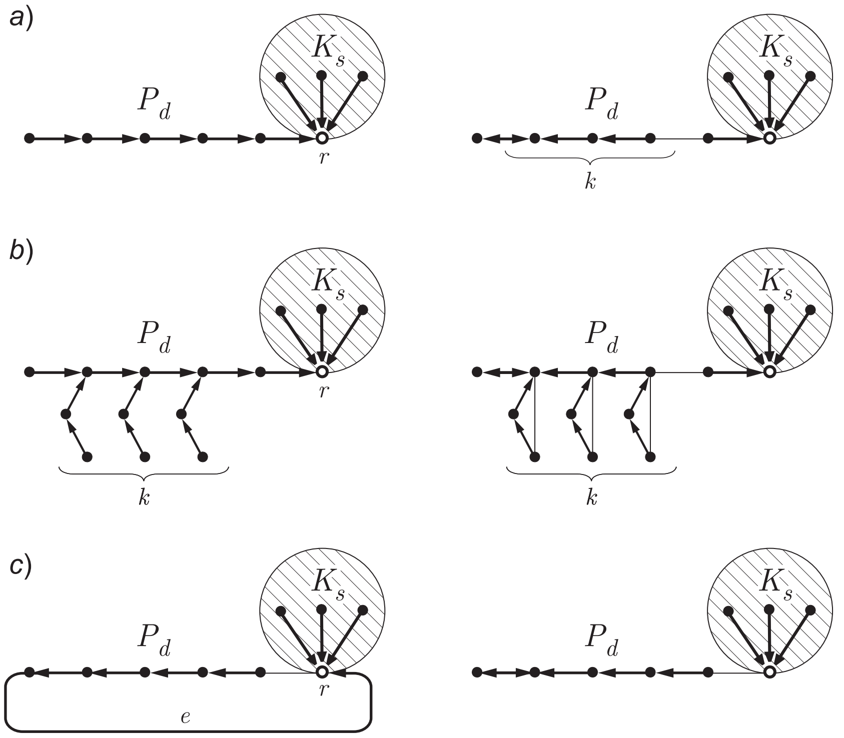

Fig. 4. Worst-case examples for the stabilization period of the rotor-router after changes to the graph: (a) modification of k port pointers, (b) addition of k edges, (c) removal of a single edge e.

(that is, they point to the same edges as before), then the agent immediately enters a new Eulerian cycle (no need for any additional stabilisation period).

Challenging questions arise with introduction of multiple agents to the rotor-router model. If we have many agents, then there are still interesting open questions left regarding the stabilisation and periodicity of exploration even in the static case (no faults, no dynamic changes of the graph).

References

1. Bender, M.A., Fernández, A., Ron, D., Sahai, A., Vadhan, S.P.: The power of a pebble: Exploring and mapping directed graphs. Inf. Comput. **176**(1) (2002) 1–21
2. Bhatt, S.N., Even, S., Greenberg, D.S., Tayar, R.: Traversing directed Eulerian mazes. J. Graph Algorithms Appl. **6**(2) (2002) 157–173
3. de Bruijn, N.G., Aardenne-Ehrenfest, T.: Circuits and trees in oriented linear graphs. Simon Stevin (Bull. Belgian Math. Soc.) **28** (1951) 203–217
4. Cooper, C., Ilcinkas, D., Klasing, R., Kosowski, A.: Derandomizing random walks in undirected graphs using locally fair exploration strategies. In: Automata, Languages and Programming, 36th Internatilonal Collogquium, ICALP 2009. Volume 5556 of Springer LNCS. (2009) 411–422
5. Cooper, J.N., Spencer, J.: Simulating a random walk with constant error. Combinatorics, Probability & Computing **15**(6) (2006) 815–822
6. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. Journal of Graph Theory **32**(3) (1999) 265–297
7. Doerr, B., Friedrich, T.: Deterministic random walks on the two-dimensional grid. Combinatorics, Probability & Computing **18**(1-2) (2009) 123–144
8. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theoretical Computer Science **345**(2-3) (2005) 331–344
9. Gąsieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. In: Proceedings 19th ACM-SIAM Symposium on Discrete Algorithms, SODA 2007. (2007) 585–594
10. Gąsieniec, L., Radzik, T.: Memory efficient anonymous graph exploration. In: Graph-Theoretic Concepts in Computer Science, 34th International Workshop, WG 2008. Volume 5344 of Springer LNCS. (2008) 14–29
11. Priezzhev, V., Dhar, D., Dhar, A., Krishnamurthy, S.: Eulerian walkers as a model of self-organized criticality. Phys. Rev. Lett. **77**(25) (Dec 1996) 5079–5082
12. Reingold, O.: Undirected connectivity in log-space. J. ACM **55**(4) (2008)
13. Tutte, W.T., Smith, C.A.B.: On unicursal paths in a network of degree 4. The American Mathematical Monthly **48**(4) (1941) 233–237
14. Wagner, I.A., Lindenbaum, M., Bruckstein, A.M.: Smell as a computational resource - a lesson we can learn from the ant. In: Proc. 4th Israel Symposium on Theory of Computing and Systems, ISTCS 1996. (1996) 219–230
15. Wagner, I.A., Lindenbaum, M., Bruckstein, A.M.: Distributed covering by ant-robots using evaporating traces. IEEE Transactions on Robotics and Automation **15** (1999) 918–933
16. Yanovski, V., Wagner, I.A., Bruckstein, A.M.: A distributed ant algorithm for efficiently patrolling a network. Algorithmica **37**(3) (2003) 165–186