Robustness of the Rotor–Router Mechanism

Evangelos Bampas, Leszek Gąsieniec, Nicolas Hanusse, David Ilcinkas, Ralf Klasing, Adrian Kosowski, Tomasz Radzik

Robustness of the Rotor-Router Mechanism*

Evangelos Bampas[1], Leszek Gąsieniec[2], Nicolas Hanusse[1], David Ilcinkas[1], Ralf Klasing[1], Adrian Kosowski[3], and Tomasz Radzik[4]

[1] LaBRI, CNRS and University of Bordeaux, France
{evangelos.bampas,nicolas.hanusse,david.ilcinkas,ralf.klasing}@labri.fr
[2] Dept. of Computer Science, University of Liverpool, UK
L.A.Gasieniec@liverpool.ac.uk
[3] Inria Paris and LIAFA, CNRS and Paris Diderot University, France
adrian.kosowski@inria.fr
[4] Dept. of Computer Science, King's College London, UK
tomasz.radzik@kcl.ac.uk

Abstract. The *rotor-router model*, also called the *Propp machine*, was first considered as a deterministic alternative to the random walk. The edges adjacent to each node v (or equivalently, the exit ports at 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 exit port to be adopted by an agent on the conclusion of the next visit to this node (the "next exit port"). The rotor-router mechanism guarantees that after each consecutive visit at the same node, the pointer at this node is moved to the next port in the cyclic order. It is known that, in an undirected graph G with m edges, the route adopted by an agent controlled by the rotor-router mechanism forms eventually an Euler tour based on arcs obtained via replacing each edge in G by two arcs with opposite direction. The process of ushering the agent to an Euler tour is referred to as the *lock-in problem*. In [Yanovski et al., Algorithmica 37(3), 165–186 (2003)], it was proved that, independently of the initial configuration of the rotor-router mechanism in G, the agent locks-in in time bounded by $2mD$, where D is the diameter of G.

In this paper we examine the dependence of the lock-in time on the initial configuration of the rotor-router mechanism. Our analysis is performed in the form of a game between a player $\mathcal{P}$ intending to lock-in the agent in an Euler tour as quickly as possible and its adversary $\mathcal{A}$ with the counter objective. We consider all cases of who decides the initial cyclic orders and the initial values π_v. We show, for example, that if $\mathcal{A}$ provides its own port numbering after the initial setup of pointers by $\mathcal{P}$, the complexity of the lock-in problem is $\mathcal{O}(m \cdot \min\{\log m, D\})$.

We also investigate the robustness of the rotor-router graph exploration in presence of faults in the pointers π_v or dynamic changes in the graph. We show, for example, that after the exploration establishes an Eulerian cycle, if k edges are added to the graph, then a new Eulerian cycle is established within $\mathcal{O}(km)$ steps.

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

1 Introduction

A graph is a fundamental combinatorial concept used for modeling complex systems in various application domains including communication, transportation and computer networks, manufacturing, scheduling, molecular biology, and peer-to-peer networks. Certain models of computation based on graphs, often classified as *alternative models of computation*, rely on the use of mobile entities called *agents*. An agent can be, e.g., a robot servicing a hazardous environment or a software process navigating the Internet in search of information.

* Parts of this work appear in preliminary form in the Proceedings of the 23rd International Symposium on Distributed Computing, LNCS vol. 5805, pp. 423–435, Springer, 2009 and in the Proceedings of the 13th International Conference on Principles of Distributed Systems, LNCS vol. 5923, pp. 345–358, Springer, 2009. Research partially supported by the ANR project DISPLEXITY (ANR-11-BS02-014). This study has been carried out in the frame of the "Investments for the future" Programme IdEx Bordeaux - CPU (ANR-10-IDEX-03-02).

The family of *anonymous graphs* provides foundations for a model that has found its application in network communication, graph exploration and stabilization of distributed processes. The nodes of an anonymous graph do not have identifiers. For each node v, each of the $d(v)$ edges adjacent to v is mapped to a unique *(exit) port number* selected from $\{1, 2, \ldots, d(v)\}$. In a synchronous system, at every step of the execution, an agent which is currently at a node v decides which port (that is, which adjacent edge) to take to move to a neighboring node. The details of a particular model would specify what type of information (if any) is maintained at the nodes and what data (if any) the agents can carry along when they move in the graph from node to node. In principle, due to minimalistic assumptions of anonymous graphs, a solution provided in this model would normally be a valid solution in other broader graph-based computation models. Another important rationale for the use of anonymous graphs is the intention to study border cases (limits of computation) in the field of distributed computing. For example, what information maintained at the nodes of the graph is sufficient to enable a single agent with no operational memory to explore the whole graph, that is, to visit each node of the graph?

We investigate the robustness of the single-agent exploration of an undirected connected graph $G = (V, E)$ 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 and in particular at the nodes of the graph. The edges adjacent to each node v (or, equivalently, the exit ports at v) are arranged in a fixed local cyclic order, which does not change during the exploration. Each node v maintains a *port pointer* π_v which indicates the exit port to be followed by an agent upon conclusion of the next visit to this node (the "next exit port"). The rotor-router mechanism guarantees that after each consecutive visit at the same node, the pointer at this node is moved to the next port in the local cyclic order. More specifically, whenever an agent exits a node v, it is directed onto edge π_v to move to the next node, and then the pointer is advanced to the edge $next(\pi_v)$ which is next after the edge π_v in the cyclic order of the edges adjacent to v. 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. This model was introduced by Priezzhev et al. [20] and then further studied and popularized by James Propp, hence it is also referred to as the *Propp machine*.

The rotor-router mechanism was introduced as a deterministic alternative to random walks on graphs and was studied in the context of a wide range of network problems, including load balancing problems [13,10], graph exploration [14,1,16], and stabilization of distributed processes [20,7,25]. Due to a limited number of configurations (positions of the node pointers and the location of the agent), it should be clear that a walk of the agent controlled by the rotor-router mechanism must eventually be locked-in in a loop. Rather surprisingly, however, Priezzhev et al. [20] proved that an agent traversing a finite graph gets locked-in in an Euler tour of the arcs (directed edges) obtained by replacing each (undirected) edge $\{u, v\}$ in G with two arcs (u, v) and (v, u). This "lock-in" in an Euler tour (which we also refer to as "entering" or "establishing" an Euler tour) means that after some initial *stabilization period* the agent will be repeatedly following the same Euler tour, without any deviations. Wagner et al. [24] showed that for arbitrary cyclic orders of edges at the nodes, arbitrary initial values of the port pointers and an arbitrary starting location of the agent, the agent covers all edges of the graph within $\mathcal{O}(nm)$ steps, where n and m are the number of nodes and the number of edges in the graph. Bhatt et al. [7] proved later that the lock-in time (the number of steps needed for the agent to get locked-in in an Euler tour) has the same $\mathcal{O}(nm)$ bound. Yanovski et al. [25] further improved the bound on the lock-in time to $2mD$, where D is the diameter of G. Related models of traversal in undirected graphs were studied in [9].

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. Links (edges) failures or some other dynamic changes in the graph occurring after the initial stabilization period knock

the agent out of the established Euler tour, but after some additional stabilization 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 stabilization period is $\mathcal{O}(mD)$ (as shown in [25], that much time is sufficient for establishing an Eulerian cycle from *any* initial configuration). One would expect that the additional stabilization time depends on the nature and extent of the failures/changes, but this has not been studied before.

In this paper we further investigate two aspects of the rotor-router mechanism. Firstly, we analyze in more detail the lock-in time, looking at different scenarios of choosing the initial configuration. We consider a number of questions here, for example, does the general $\mathcal{O}(mD)$ bound on the worst-case lock-in time decrease, if we can choose which edges are the initial values of the node pointers? Secondly, we provide bounds on the length of the additional stabilization period required after failures or dynamic changes in the graph.

1.1 Our contribution

We examine the influence of the initial configuration of pointers and port numbers on the time needed to lock-in the agent in an Euler tour. The case study is performed in the form of a competition between a *player* $\mathcal{P}$ intending to lock-in the agent in an Euler tour as quickly as possible and its *adversary* $\mathcal{A}$ having the counter objective. We assume that both the player $\mathcal{P}$ and its adversary $\mathcal{A}$ have unlimited computational power, i.e., we do not take into account the cost of computation of the initial configuration of ports and pointers to be adopted by $\mathcal{P}$ and $\mathcal{A}$. The results of our studies are asymptotically tight in terms of the worst-case choice of the graph topology and the initial location of the agent.

We start our analysis with border cases. In the case $\mathcal{P}$-*all*, where the player $\mathcal{P}$ is in charge of the initial arrangement of port numbers and pointers, we observe that the lock-in in an Euler tour can be obtained in $\mathcal{O}(m)$ steps. Also, the case $\mathcal{A}(\circlearrowleft)\mathcal{P}(\pi)$, where first the port numbers are assigned by $\mathcal{A}$ and then $\mathcal{P}$ sets the pointers, reduces to the border case $\mathcal{P}$-*all*, where $\mathcal{P}$ is solely in charge of the initial configuration. On the other hand, in the case $\mathcal{A}$-*all*, where the adversary solely decides about the initial configuration, we show that, in any graph with m edges and diameter D, the adversary is able to enforce the lower bound $\Omega(mD)$ for the lock-in time, matching the upper bound from [25].

Furthermore, we show that, for the case $\mathcal{P}(\pi)\mathcal{A}(\circlearrowleft)$, where $\mathcal{P}$ first provides the initial setup of pointers and then $\mathcal{A}$ decides its own port numbering, the complexity of the lock-in problem is bounded by $\mathcal{O}(m \cdot \min\{\log m, D\})$. We also construct a class of graphs in which the lock-in requires time $\Omega(m \cdot \min\{\log m, D\})$. At the same time, we point out that, e.g., in Hamiltonian graphs the lock-in is obtained in time $\mathcal{O}(m)$.

We conclude our analysis of the lock-in time with the proof that, in the remaining two cases $\mathcal{A}(\pi)\mathcal{P}(\circlearrowleft)$ and $\mathcal{P}(\circlearrowleft)\mathcal{A}(\pi)$, the lock-in requires time $\Omega(m \cdot D)$ in graphs with the worst-case topology. More precisely, we first show that, in the case $\mathcal{A}(\pi)\mathcal{P}(\circlearrowleft)$, where $\mathcal{P}$ responds with a port assignment to the initial setup of pointers by $\mathcal{A}$, there exist graphs for which the lock-in requires time $\Omega(m \cdot D)$. At the same time, we present a non-trivial class of graphs with an arbitrarily large diameter in which an appropriate choice of port numbers leads to the lock-in in time $\mathcal{O}(m)$. Finally, in the case $\mathcal{P}(\circlearrowleft)\mathcal{A}(\pi)$, where $\mathcal{A}$ sets the pointers after the assignment of ports is revealed by $\mathcal{P}$, the lower bound $\Omega(m \cdot D)$ argument for the lock-in follows directly from the previous case. Also, here we propose a non-trivial class of graphs, this time with an arbitrary diameter $D \leq \sqrt{n}$, in which the lock-in is feasible in time $\mathcal{O}(m)$.

Our results are summarized in Table 1. The worst-case bounds hold for the worst choice (for the player $\mathcal{P}$) of a graph with m edges and diameter D and an arbitrary choice of the starting node. The best-case bounds for all scenarios except the last $\mathcal{A}$-*all* scenario hold for the best choice of a graph and an arbitrary choice of the starting node. The $\Theta(m + D^2)$ best-case bound for the $\mathcal{A}$-*all* scenario holds for the best choice of a graph and the best choice of a starting node.

Table 1. The worst and the best case lock-in times in considered scenarios for graphs with m edges and diameter D. The asymptotic results hold for any choice of the starting node s in the graph, unless otherwise stated.

Scenario	Lock-in time for worst-case graph	Lock-in time for best-case graph
$\mathcal{P}$-*all*	$\Theta(m)$	$\Theta(m)$
$\mathcal{A}(\circlearrowleft)\mathcal{P}(\pi)$	$\Theta(m)$	$\Theta(m)$
$\mathcal{P}(\pi)\mathcal{A}(\circlearrowleft)$	$\Theta(m \cdot \min\{\log m, D\})$, Thm. 4 & 5	$\Theta(m)$, Rem. 1
$\mathcal{A}(\pi)\mathcal{P}(\circlearrowleft)$	$\Theta(m \cdot D)$, Thm. 6	$\Theta(m)$, Thm. 7
$\mathcal{P}(\circlearrowleft)\mathcal{A}(\pi)$	$\Theta(m \cdot D)$, Cor. 1	$\Theta(m)$, if $D \leq m^{1/3}$, Thm. 8
$\mathcal{A}$-*all*	$\Theta(m \cdot D)$ [25]	$\Theta(m + D^2)$, [best starting node] Thm. 2
		$\Theta(m \cdot D)$, [worst starting node] Thm. 3

The $\Theta(m \cdot D)$ best-case bound for the $\mathcal{A}$-*all* scenario holds for the best choice of a graph and the worst choice of a starting node.

Regarding the additional stabilization period needed in case of failures or changes in the graph, we develop bounds which depend on the number of such failures or changes. Here, 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 $\Theta(m \cdot \min\{k, D\})$ 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 $\Theta(m \cdot \min\{k, D\})$ 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 $\mathcal{O}(\gamma m)$ steps, where γ is the length of the smallest cycle in graph G containing the deleted edge in the original graph.

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.

Our analysis of the additional stabilization period in the rotor-router mechanism is based on the relationship between Eulerian cycles and spanning trees in a graph, which underlies the following classical theorem.

BEST Theorem (de Bruijn, van Aardenne-Ehrenfest, Smith, Tutte [23]).
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.

The relationship between spanning trees and Eulerian cycles was used by Holroyd et al. [17] in their analysis of periodicity of the rotor-router mechanism. To analyze the additional stabilization period, we generalize this relationship to forests and emerging Eulerian cycles.

1.2 Previous work

The previous work which is most directly relevant to our paper is Bhatt et al. [7] and Yanovski et al. [25], both already mentioned above. Bhatt et al. [7] consider also mechanisms enabling the

agent to stop after exploring the whole graph. Yanovski et al. [25], in addition to proving the $2mD$ bound on the length of the stabilization period, show also that this bound is asymptotically optimal in the worst case, and initiate a study of the case when there are $k \geq 2$ agents. A characterization of the long-term behavior of the rotor-router for $k \geq 2$ agents has been provided more recently by Chalopin et al. [8]. 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 [24,25], while the same model is described in [7] 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. [9] 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 [10] 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 [13] analyse in more detail the distribution of tokens in the rotor-router mechanism on the 2-dimensional grid. Further results, which make use of the rotor-router as a load-balancing strategy for uniform processors in a regular network, are presented in [2,6].

While the model considered in [10] and [13] refers to the case when the number of tokens is large in comparison with the number of explored nodes, the recent papers by Klasing et al. [18], Dereniowski et al. [12], and Kosowski and Pajak [19] present results for graph exploration using the multiple-agent rotor-router mechanism, which cover also the case of relatively small number of agents. For example, it is shown in [12] that k agents cover all nodes of any graph in the number of steps which is $\mathcal{O}(mD/\log k)$ and $\Omega(mD/k)$, and there are examples of graphs for both ends of this range.

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 [15,4,21]. Models which allow placement of some identifiers or markers on nodes or edges of the graph have been considered for example in [5,11]. Some graph exploration techniques are surveyed in [16].

2 Preliminaries

In this section, we provide basic definitions and recall known facts in relation to the operation and performance of the rotor-router mechanism in anonymous graphs.

Let $G = (V, E)$ be an undirected connected graph with n nodes and m edges. The directed graph $\vec{G} = (V, \vec{E})$ is the directed symmetric version of G, with arc set $\vec{E} = \{(v, u), (u, v) : \{v, u\} \in E\}$. Observe that $\vec{G}$ is always Eulerian. We will refer to the undirected links in graph G as *edges* and to the directed links in graph $\vec{G}$ as *arcs*. We will also be using an arrow on the top of a symbol, as in $\vec{G}$ and $\vec{E}$, to stress the fact that we refer to directed graphs and arcs. For a node $v \in V$, $d(v)$ denotes the degree of v in G, $E_G(v)$ denotes the set of edges that are incident to v in G, and $\vec{E}_G(v)$ denotes the set of arcs outgoing from v. If $H = (X, C)$ is a subgraph of G induced by some subset of edges $C \subseteq E$, then $N_G(H)$ denotes the subgraph of G (and a supergraph of H) induced by the set of edges $\bigcup_{v \in X} E_G(v)$. Finally, D denotes the diameter of G.

We consider the rotor-router model (on graph G) with a single agent. For each node $v \in V$, a cyclic order ρ_v of the arcs outgoing from v is associated with v. We denote by $next(v, u)$ the arc

next after arc (v, u) in this cyclic order. Equivalently, node v can be viewed as having $d(v)$ "ports" numbered $1, 2, \ldots, d(v)$, where the outgoing arcs are connected, and ρ_v as an assignment of port numbers to the outgoing arcs. The cyclic orders ρ_v are fixed at the beginning of exploration and do not change in any way from step to step (except in the case of dynamic changes to the graph, which we discuss in Section 4). Additionally, for each node $v \in V$, a *port pointer* $\pi_v \in \vec{E}_G(v)$ is maintained at v. The initial values of these pointers are set at the beginning of exploration, and the port pointer π_v at a node v changes whenever the agent visits v as described below. The *configuration* of the rotor-router at the current step is described by the pair

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

where r is the *current node*, that is, the node where the agent is at the current step.

The agent moves in discrete steps from node to node along the arcs of graph $\vec{G}$. During the current step, first the agent moves from the current node r traversing the arc $\pi_r = (r, r')$ to a neighboring node r', and then the port pointer π_r is advanced to the next arc outgoing from r (that is, π_r becomes $next(\pi_r)$ immediately after the agent leaves r). Thus, for all nodes v, the pointer π_v always points to the outgoing arc from v which the agent will take on the next visit to r.

The exploration starts from some initial configuration and then keeps running without ever terminating. In this paper, we consider the rotor-router model as a mechanism for exploring a graph, so the most interesting questions for us are how quickly the agent explores the whole graph, and how evenly, in the long run, it keeps traversing the arcs of the graph. As discussed in the previous section, it is well-known that, whatever the initial configuration is, the agent eventually locks in an Eulerian cycle of $\vec{G}$. That is, from some step on, the agent keeps repeatedly tracing the same Eulerian cycle of $\vec{G}$. In our analysis we need to look in detail at how this eventual state of following an Eulerian cycle is reached. The following two lemmas and related definitions provide the basis for our analysis.

Lemma 1 ([20]). *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).*

Definition 1. *During a rotor-router exploration, a node becomes* saturated *when all its incident edges have been traversed in both directions for the first time.*

Definition 2. *The rotor-router exploration is viewed as a sequence of* phases $\{P_i\}_{i \geq 0}$. *The trivial phase P_0 (introduced for convenience) is the initial event of placing the agent at the start node s before the first step. Each subsequent phase starts when the agent leaves node s via the initial port π_s and continues until the agent traverses all edges incident to s in both directions (and the pointer π_s returns to its initial value).*

Note that when a node v becomes saturated, its pointer π_v returns to the initial position for the first time. Note also that Lemma 1 implies that every edge is traversed at most once in each direction during each phase.

Lemma 2 ([7]). *Each phase P_i, $i \geq 1$, of the rotor-router exploration of graph G has the following properties.*

- *While the agent is visiting nodes saturated in an earlier phase or earlier in this phase, it retraces the route of the previous phase P_{i-1}.*
- *If the agent encounters a node u that has been visited in an earlier phase but is not saturated yet, it suspends the retracing of the tour of phase P_{i-1}. A new (sub-)tour starts at u, traverses only new arcs (that is, the arcs not traversed earlier) and ends at node u when u becomes saturated. The tour of phase P_{i-1} is then resumed (via port π_u, which was the initial port at u).*

Lemma 2 has the following consequences. If phases P_j, for $j = 0, 1, \ldots, i-1$, have not saturated all nodes, then the next phase P_i is longer than the previous one and saturates at least one additional node. Eventually, all nodes in G become saturated by the end of some phase. This phase follows an Eulerian cycle of $\vec{G}$ (traverses each arc exactly once) and each subsequent phase is exactly the same.

Definition 3. *The* stabilization time t_0 *of the rotor-router is the number of steps taken by the agent until it completes for the first time an Euler tour of* $\vec{G}$.

Observe that this definition implies that the stabilization time is always at least $2m$. To get a general upper bound on the stabilization time, conclude from Lemma 2 that for each $i \geq 1$, the connected subgraph G_i of G defined by the edges traversed (in both direction) during phase P_i includes all edges in G that are incident to nodes in G_{i-1}. That is, $N_G(G_{i-1})$ is a subgraph of G_i, where G_0 contains just the starting node s. Thus all nodes at distance at most i are in G_i and the subgraph of G induced by these nodes is a subgraph of G_{i+1}. Since the length of each phase is bounded by $2m$ (each arc is traversed at most once in each phase), the next theorem follows.

Theorem 1 ([25]). *For any graph G, any starting node, and any cyclic orders ρ_v and initial pointer assignments π_v, we have the stabilization time $2m \leq t_0 \leq 2m(D+1)$. From step $t_0 + 1$ the agent keeps repeating the same Eulerian cycle of graph $\vec{G}$.*

Definition 4. *For any m and D, $D \leq m$, let $\mathcal{G}_{m,D}$ denote the class of graphs with diameter between D and $4D$ and a number of edges between m and $4m$.*

3 Case Study of the Lock-in Problem

In this section we study the game between $\mathcal{P}$ and $\mathcal{A}$ in detail. Recall that the goal of $\mathcal{P}$ is to minimize the stabilization time, whereas the goal of $\mathcal{A}$ is to maximize it.

We first discuss the cases when the fast $\Theta(m)$ lock-in time is possible for any graph. Consider first the case $\mathcal{P}$-*all* where the player $\mathcal{P}$ is solely responsible for the initial setup of port numbers and pointers. Clearly, for any graph G and any starting node s, the player can choose a configuration that locks the agent in an Euler tour right from the beginning: Choose an arbitrary Eulerian cycle $\vec{C}$ of $\vec{G}$, take one of the occurrences of s on $\vec{C}$ as the start point and for each node v, set the cyclic order ρ_v according to the order of the arcs along $\vec{C}$ and set the π_v pointer to the first arc on $\vec{C}$ with the tail at v.

Similarly, also in the case $\mathcal{A}(\circlearrowleft)\mathcal{P}(\pi)$, for any input graph G and any starting node s, after the adversary $\mathcal{A}$ sets port numbers, the player $\mathcal{P}$ can respond with an appropriate assignment of pointers that instantly leads to an Euler tour. An obvious way for the player to calculate the initial assignment of pointers is to start with arbitrary pointers and simulate the rotor-router until the first phase when all arcs are traversed. Then the player takes the pointers at the end of this phase as the initial pointers. (A faster way of setting the pointers will be shown in Section 4.) Thus in the cases $\mathcal{P}$-*all* and $\mathcal{A}(\circlearrowleft)\mathcal{P}(\pi)$ the agent gets locked in an Euler tour in time $\Theta(m)$.

3.1 Long lock-in times – case $\mathcal{A}$-*all*

At the other end of the spectrum, in the case $\mathcal{A}$-*all* where the adversary $\mathcal{A}$ is solely responsible for the initial configuration of port numbers and pointers, the worst-case bound of $\Theta(m \cdot D)$ was established in [25], so we focus on deriving tight best-case bounds. Here the main question is to obtain lower bounds which apply to all graphs. We show two such lower bounds: an $\Omega(m + D^2)$ bound for any graph and any starting node (Theorem 2), and an $\Omega(mD)$ bound for any graph and the worst choice of the starting node (Theorem 3). We will need the following two lemmas.

Lemma 3. *Let $G = (V, E)$ be an input graph with a starting node $s \in V$. For any subset $C \subseteq E$ such that C contains at least $E_G(s)$ and also induces a connected subgraph $H = (X, C)$ of G, there exists an assignment of ports and pointers such that the first phase of the exploration of G traverses all edges in C in both directions, and only these edges.*

Proof. Let $\mathcal{C}$ be an Euler cycle in $\vec{H}$. Fix the corresponding sequence of edge traversals $e_1, \ldots, e_{2|C|}$, starting with an edge e_1 incident to s. Each undirected edge in C is traversed exactly twice by $\mathcal{C}$, once in each direction. For each node $v \in V$, we now define a port assignment and an assignment of pointers π_v.

Let $e_{v_1}, \ldots, e_{v_k}$ be the order in which its incident edges are traversed in $\mathcal{C}$, going *out* of v. It can happen that $k < d(v)$ if v has incident edges in $E \setminus C$, or even $k = 0$ if $v \notin X$. Define the port assignment for the node v so that for any $i \leq k$, edge e_{v_i} is the port with number i. If $k < \deg_G(v)$, extend this port assignment so that edges in $E \setminus C$ receive higher port numbers than edges in C. Finally, define π_v to be the edge e_{v_1}, if $k \geq 1$; otherwise π_v can be an arbitrary edge in $E_G(v)$.

Now, let $\mathcal{E}$ be the sequence of edges traversed by the agent in the first phase of the exploration of G starting from s. For every node $v \in V$ and every i, consider the i-th time that $\mathcal{E}$ visits v. The edge followed then by $\mathcal{E}$ is e_{v_i}, which coincides with the edge that $\mathcal{C}$ followed during the i-th visit at v. It follows that $\mathcal{E}$ coincides with $\mathcal{C}$ and therefore $\mathcal{E}$ traverses all edges of C in both directions, and only these edges. □

Lemma 4. *Let $G = (V, E)$ be an undirected graph with a starting node $s \in V$, a given port assignment, and a pointer assignment π_v for each node of G. Let $\mathcal{E}$ be the sequence of edges traversed by the agent in the first i phases of exploration, for some $i \geq 1$. Let $H = (X, C)$ be the subgraph of G induced by the edges traversed in $\mathcal{E}$ (not necessarily in both directions). The ports and pointers of nodes in $V \setminus X$ can be modified so that, during phase $i + 1$ of the exploration, the agent traverses all edges of $N_G(H)$ in both directions, and does not traverse any other edges in G.*

Proof. Let $N_G(H) = (Y, D)$. Clearly, $Y \supseteq X$ and $D \supseteq C$. Phase $i + 1$ of the exploration will saturate all nodes that were visited during the first i phases. This implies that all edges incident to nodes in X are traversed in both directions during phase $i + 1$. Therefore all edges in D will be traversed in both directions (see Lemma 2).

To ensure that no other edges will be traversed, we modify the port assignment of nodes in $Y \setminus X$ as follows. For each $v \in Y \setminus X$, all edges connecting v to nodes in X receive smaller port numbers than all edges connecting v to nodes in $V \setminus X$. Furthermore, we set π_v to be the edge with port number 1, for all $v \in Y \setminus X$.

To prove the claim, assume for the sake of contradiction that during phase $i + 1$, the agent traverses some edges in $E \setminus D$. Let e be the first such edge. The edge e must have been traversed on the way out from some node $v \in Y \setminus X$. But, due to the port numbering scheme defined above, the cyclic distance between the port number of e and the first pointer at node v is greater than the number of edges that connect v to nodes in X, which implies that at least one of these edges was traversed at least twice in the same direction (towards v) during phase $i + 1$. This leads to a contradiction, since an edge is never traversed twice in the same direction in one phase. □

We will use now Lemma 4 to show lower bounds on the lock-in time for the case $\mathcal{A}$-*all*.

Theorem 2. *For any graph $G = (V, E)$ and any starting node $s \in V$, there exist port and pointer assignments in G, such that $t_0 = \Omega(m + D^2)$. For any $1 \leq D \leq m$, there exists a graph $G = (V, E) \in \mathcal{G}_{m,D}$ and a starting node $s \in V$ such that for any port and pointer assignments in G, $t_0 = \mathcal{O}(m + D^2)$.*

Proof. The first part of this theorem follows immediately from Lemma 4. The port numbers and pointers can be chosen so that there are at least $D/2$ phases (during the stabilization period) and the length of phase i is always at least $2i$.

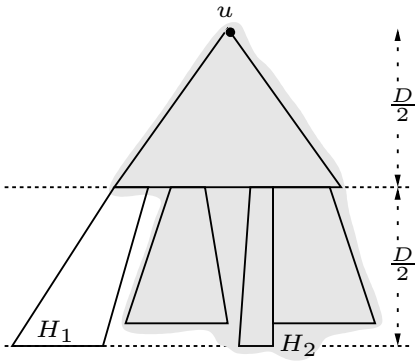

Fig. 1. The partition of G into subgraphs that is described in the proof of Theorem 3. Either subgraph H_1 or subgraph H_2 contains at least half of the edges of G.

For the second part of the theorem, take a lollipop graph obtained by connecting with an edge a complete graph $K = K_{\lceil\sqrt{m}\rceil}$ to one end of a path with $D-1$ nodes. Set the other end of the path as the starting node. Whatever port and pointer assignments the adversary chooses, there will be $q \leq D$ phases, and the lengths l_i of these phases will be such that $1 \leq l_1 < l_2 < \cdots l_{q-2} \leq D$, $l_{q-1} < 2m$ and $l_q = 2m$. Thus the length of the whole stabilization period is $\mathcal{O}(m + D^2)$. □

Theorem 3. *For any graph $G = (V, E)$, there exist a starting node $s \in V$ and port and pointer assignments in G, such that $t_0 \geq \frac{1}{2} \cdot mD$.*

Proof. Assume that $D \geq 4$, let T be a BFS tree of G of height D, and let $u \in V$ denote the root of T. Finally, let H be the subgraph of G induced by the nodes whose distance from u in T is at least $\frac{D}{2}$, and let $H_1 = (X, C)$ be a connected component of H that contains at least one node whose distance from u in T is D; see Figure 1.

If H_1 contains at least $\frac{m}{2}$ edges, then pick an arbitrary starting node s in H_1 and set the ports and pointers so that the first phase of the exploration starting from s explores exactly $G_1 = H_1$. This is feasible due to Lemma 3. Furthermore, arrange ports and pointers so that for any $i \geq 2$, $G_i = N_G(G_{i-1})$, where G_i denotes the graph induced by the edges traversed during phase P_i of the exploration. This is feasible by repeated applications of Lemma 4. In this case, the exploration from s will require at least $2 \cdot \frac{m}{2} \cdot \frac{D}{2} \geq \frac{1}{2} \cdot mD$ edge traversals before visiting all nodes.

Otherwise, the subgraph H_2 induced on G by the edge set $E \setminus C$ must contain at least $\frac{m}{2}$ edges. Pick a starting node s in H_2 and set the ports and pointers so that $G_1 = H_2$ and for any $i \geq 2$, $G_i = N_G(G_{i-1})$. The exploration will again require at least $\frac{1}{2} \cdot mD$ edge traversals before visiting all nodes. □

To summarize, in the $\mathcal{A}$-*all* case, the lock-in time is $\Theta(m + D^2)$ for the best graph and the best choice of the starting node, and $\Theta(mD)$ for the best graph (hence also for any graph because of the general $\mathcal{O}(mD)$ bound) and the worst choice of the starting node.

3.2 Almost linear lock-in – case $\mathcal{P}(\pi)\mathcal{A}(\circlearrowleft)$

In this section we discuss the case where the player $\mathcal{P}$ chooses pointers first and the adversary $\mathcal{A}$ responds with the worst-case assignment of ports.

Theorem 4. *For any graph $G = (V, E)$ and any starting point $s \in V$, there exists a pointer assignment such that for any port assignment $t_0 = \mathcal{O}(m \cdot \min\{\log m, D\})$.*

Proof. We show that the player can find an assignment of pointers such that the lock-in is obtained in phase P_i, for some $i \leq \min\{\log m, D\}$.

Take an arbitrary BFS tree T in G rooted at s. For every node in T compute a *rank* according to the following rules. Each leaf in T acquires rank 0. For each internal node v (including the root s) we look at the rank of its children. If the highest rank r belongs to only one child, the node v adopts r as its own rank. Otherwise, i.e., when the highest rank is shared by at least two children, the node v adopts the rank $r+1$. One can prove that the rank r_s of the root s is the largest and it does not exceed $\log m$. It does not exceed D either, because we chose a BFS tree. The rank of the root is known as the *Strahler number*, a numerical measure of branching complexity of the tree T [22]. Note that the nodes with the same rank form a collection of *downward chains* in T.

After the ranks are introduced to T, the pointer at each node in T is assigned to the port leading towards a child with the largest rank. This is to ensure that G_i contains all nodes in T with ranks $\leq r_s - i + 1$. And indeed, G_1 contains all nodes with rank r_s, since, as soon as the traversal process is initiated, the agent is forced to visit all nodes with the highest rank (and possibly some others). Assume now inductively that all nodes with ranks $\leq r_s - i + 1$ belong to G_i. These include the nodes that are connected to downward chains with rank $r_s - i$ with nodes still not present in G_i. But note that, due to Lemma 2, all edges incident to nodes in G_i are present in G_{i+1} which means that each downward chain with rank $r_s - i$ will be accessed and all of their nodes will be traversed when G_{i+1} is formed. This proves that G_{r_s} contains all nodes from G and G_{r_s+1} contains all nodes and edges. Since $r_s \leq \min\{\log m, D\}$ and the number of edges in each G_i is bounded by m, the lock-in time is $\mathcal{O}(m \cdot \min\{\log m, D\})$. □

Remark 1. Note finally, that if G is Hamiltonian the player $\mathcal{P}$ can arrange pointers so that they form a Hamiltonian tour. This ensures that G_2 contains all edges in G and that the stabilization time in such graphs is $\mathcal{O}(m)$ (see Lemma 2).

We now show that there exist graphs for which the lock-in upper bound of Theorem 4 is asymptotically matched from below.

Theorem 5. *For any m and $D \leq m$, there exists a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ such that for any starting node s and any initial pointer assignment, there exists a port assignment for which $t_0 = \Omega(m \cdot \min\{\log m, D\})$.*

Proof. We first show a graph which satisfies the requirements of this theorem for one designated starting node. Consider a graph G formed of a complete graph $K = K_{\lceil\sqrt{m}\rceil}$ with $\Theta(m)$ edges and $\Theta(\sqrt{m})$ nodes, connected by a path of length $\lceil\max\{1, D - \log m\}\rceil$ with a complete binary tree B of height $\lceil\min\{\log m, D\}\rceil$. The starting node s is the node in K which is connected to the path. This is to ensure that G_2 contains all the edges from K. Consider now the arrangement of pointers in each node of B. We show that, independently of the assignment of the initial pointer at an internal node v, if i is the smallest integer such that v is visited in phase P_i, then one of its children is not present in G_i.

And indeed, assume that G_i is the first graph in which v is visited by the agent. There are three ports associated with v. One port leads to its parent and two towards its children. If the player $\mathcal{P}$ decides to assign the pointer to the port leading towards the parent of v, then after the agent arrives at v (forming a part of G_i) it immediately returns back to the parent of v. Since each edge in G_i is visited exactly once in each direction (cf. Lemma 2), the next visit at v must occur in G_{i+1}. Thus, none of its children can be present in G_i.

Now, assume that the pointer is assigned to a port k leading to one of the children c_1 of v. Since the port numbers available at v are $\{1, 2, 3\}$, the adversary $\mathcal{A}$ assigns number $(k \bmod 3)+1$ (that follows k in the cyclic order) to the port leading to the parent of v. This ensures that after the agent comes back from c_1, it immediately returns to the parent of v. Since each edge in G_i is visited exactly once in each direction (cf. Lemma 2), the next visit at v must occur in G_{i+1}. This proves that the other child of v does not belong to G_i. Thus, there is a path from the root of B to some leaf on which neither of any two consecutive nodes belong to the same G_i.

Finally, since the height of B is $\Theta(\min\{\log m, D\})$ and each G_i, for $i \geq 2$, contains at least $\Omega(m)$ edges, the lock-in requires time at least $\Omega(m \cdot \min\{\log m, D\})$.

To get a graph which has the required stabilization time for each node as the starting node, take two copies G' and G'' of the above graph G and join them into one graph by identifying their starting nodes $s_1 = s_2 = s$. If the starting node of the new combined graph is any node in part G', then $\Omega(m \cdot \min\{\log m, D\})$ steps will be required to cover all edges of part G''. The reason for this is that if we disregard the steps taken in part G', then the remaining steps are traversing part G'' in exactly the same way as in the exploration of graph G above. Analogously, if the starting node of the new combined graph is in part G'', then $\Omega(m \cdot \min\{\log m, D\})$ steps will be required to cover all edges of part G'. □

3.3 The two remaining cases: $\mathcal{A}(\pi)\mathcal{P}(\circlearrowleft)$ and $\mathcal{P}(\circlearrowleft)\mathcal{A}(\pi)$

In the last part of this section, we discuss two cases with the worst-case complexity $\Omega(m \cdot D)$. We show, however, that here, in contrast to the border case $\mathcal{A}$-*all*, there exist non-trivial classes of graphs with a stabilization time of $\mathcal{O}(m)$.

Case $\mathcal{A}(\pi)\mathcal{P}(\circlearrowleft)$. In the case where the player responds by a port assignment to the adversary's initial pointer assignment, we demonstrate a family of graphs in which stabilization requires time $\Omega(mD)$, matching the general worst-case upper bound from Theorem 1. We also demonstrate a non-trivial family of graphs in which, for any choice of starting point, the stabilization time is at most $\mathcal{O}(m)$.

Theorem 6. *For any m and $D \leq m$, there exists a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ such that for any starting node s, there is an initial pointer assignment, such that for any port assignment, $t_0 = \Omega(mD)$.*

Proof. We only show a graph G which satisfies the requirements of this theorem for one designated starting node. A graph satisfying the requirements for any starting node can be obtained from two copies of graph G as in the proof of Theorem 5.

Let G be a lollipop graph obtained by connecting a complete graph $K = K_{\lceil\sqrt{m}\rceil}$ to a path P_D with D nodes via a bridging edge. Let s be a node of K different from the node connecting K to P_D, and let the pointers within K point towards s (the pointer of s itself can initially be on an arbitrary port). Finally, set the pointers at each node of P_D to point towards K.

It is clear that, no matter which port assignment is chosen by the player, during the first phase of the exploration initiated in s, the agent traverses the edges connecting s to its neighbors in the clique in both directions, thus visiting all nodes in K. During the second phase, the agent will traverse all $\Theta(m)$ edges of K in both directions, and it will return to K by the first pointer of P_D. During subsequent phases of exploration the agent will progress along the path at a rate of one edge per phase, until the last node of the path is reached. Therefore, D phases are required, each of which retraces at least the $\Theta(m)$ edges in K; the lower bound of $\Omega(m \cdot D)$ for the lock-in time follows. □

Theorem 7. *For any m and $D \leq m$, there exists a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ such that for any starting node s and for any initial pointer assignment, there exists a port assignment for which $t_0 \leq 24m$.*

Proof. Let $G = (V, E)$ be a graph consisting of two chains of $D + 1$ nodes, where additionally the i-th node of each chain is connected to the $(i-1)$-st and $(i+1)$-st nodes of the other chain (except for the first and $(D+1)$-st nodes, which are connected only to the second and D-th nodes of the other chain, respectively). The number of edges in G so far is equal to $4D \leq 4m$. In the case where $4D < m$, append to G a star consisting of $m - 4D + 2$ edges, as illustrated in

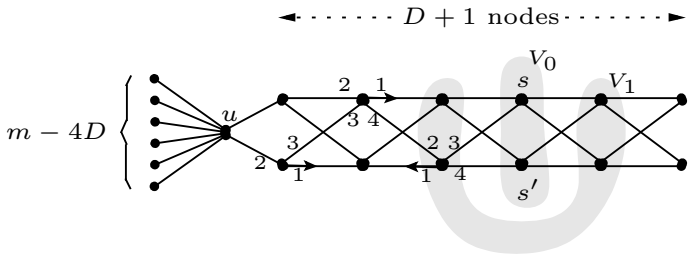

Fig. 2. The construction described in the proof of Theorem 7 for the values of parameters $m = 26$ and $D = 5$.

Figure 2. In both cases, the diameter of the final graph G is either D or $D+2$ and the number of edges is between m and $4m$, thus $G \in \mathcal{G}_{m,D}$. Let $s \in V$ be the starting node in G, and $(\pi_v)_{v \in V}$ be the pointer assignment supplied by the adversary $\mathcal{A}$. Denote the set of nodes of the two chains of G by X, and the central node of the appended star by u (if it exists).

For the time being, assume also that $s \in X$ and s' is the node on the other chain that has exactly the same neighbors as s. Furthermore, let V_i denote the subset of X that contains nodes at distance i from s ($i \geq 0$), with the exception that s' belongs not to V_2 but to V_0. We adopt a port assignment for the nodes of G as follows (refer to Figure 2 for illustration):

- For the special case of s and s', arrange the port numbers arbitrarily.
- For the node u, assign port 1 to π_u. If π_u connects u to one of the chains, assign port number $d(u)$ to the edge that connects u to the other chain, and the rest of the ports arbitrarily. Otherwise, assign port numbers $d(u) - 1$ and $d(u)$ to the two edges connecting u to the chains, and set the rest of the port numbers arbitrarily.
- For a node $v \in X$ at the endpoints of the chains, if v is not connected to u then set ports arbitrarily. If v is connected to u, then assign port 1 to π_v and assign the smallest possible port to the edge connecting v to u (if it is not π_v).
- For any other $v \in X$, port 1 is always assigned to π_v. Let $i \geq 1$ be the distance of v from s, thus $v \in V_i$. If π_v connects v to V_{i-1}, assign ports 2 and 3 to the edges connecting v to V_{i+1}, and port 4 to the remaining edge that connects v to V_{i-1}. Otherwise, assign port 2 to the remaining edge connecting v to V_{i+1} and ports 3 and 4 to the edges connecting v to V_{i-1}.

We claim that during the second phase of exploration the agent traverses all edges in both directions. In order to prove this claim, we first observe that during the first phase of exploration the agent must visit all the nodes in V_1. Therefore, during the second phase all the edges connecting V_0 to V_1 are traversed in both directions. Now, for some $i \geq 1$, assume that during the second phase of exploration the agent traverses all edges connecting V_i to V_{i-1} in both directions. According to the port assignment scheme defined above, for any $v \in V_i$ there is an incident edge e with port number 4 that connects it to some node in V_{i-1}. By assumption, e is traversed in both directions during the second phase. But before the tour of the second phase can use edge e on the way out of v, it is forced to use all other edges incident to v also on the way out of v, and in particular the edges that connect v to V_{i+1}. Since this property holds for all $v \in V_i$, and since the edges connecting V_i to V_{i+1} constitute a cut that disconnects s from V_{i+1}, it follows that these edges must be traversed in both directions during the second phase. It follows by induction that all edges of the chain are traversed in both directions during the second phase. Furthermore, consider any node $v \in X$ that is connected to u. The edge with the highest port number at v is traversed in both directions, therefore by the same argument all edges incident to v are traversed in both directions. Applying the same argument one more time for the node u concludes the proof of the claim.

Now, if $s \notin X$, we set the ports of nodes in X analogously, pretending that s is one of the endpoints of the chain that are connected to u. After at most two phases of exploration the agent traverses all edges of the star centered at u, and thus it visits the two endpoints of the

chain connected to u. Then, by a similar argument, during the third phase the agent traverses all edges in G in both directions.

We have proved that for any starting point and any pointer assignment, after at most three phases of exploration the agent traverses all edges in G in both directions. Since during each phase at most $4m$ edges are traversed in each direction, the upper bound of $24m$ for the lock-in time follows. □

Case $\mathcal{P}(\circlearrowleft)\mathcal{A}(\pi)$. In the case where the adversary $\mathcal{A}$ responds by a pointer assignment to player $\mathcal{P}$'s initial port assignment, the existence of a family of graphs in which the stabilization time is at least $\Omega(mD)$, matching the general worst-case upper bound from Theorem 1, is an immediate corollary of Theorem 6.

Corollary 1. *For any m and $D \leq m$, there exists a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ with a starting node $s \in V$, such that for any port assignment there exists a pointer assignment under which $t_0 = \Omega(mD)$.*

We show, however, that there is also a non-trivial class of graphs with diameter $\mathcal{O}(m^{1/3})$ in which the stabilization time is $\mathcal{O}(m)$ in this case.

Theorem 8. *For any m and $D \leq m^{1/3}$, there exists a graph $G = (V, E)$ with m edges and diameter at most D such that, for any starting node s, there exists a port assignment, such that for any initial pointer assignment $t_0 \leq 8m$.*

Proof. For any $a \geq 2$, let $G_a = (V, E)$ be the graph consisting of a chains of $a + 1$ nodes, where additionally the i-th node of each chain is connected to the $(i - 1)$-st and $(i + 1)$-st nodes of all the other chains (except for the first and $(a + 1)$-st nodes, which are connected only to the second and a-th nodes of the other chains, respectively).

Let $s \in V$ be a starting node with eccentricity ϵ. Let V_0 be the subset of nodes with the same neighbors as s (including s), and let V_i, $1 \leq i \leq \epsilon$, be the subset of $V \setminus V_0$ that contains all nodes at distance i from s. Moreover, let E_i denote the set of edges connecting V_i to V_{i+1} $(0 \leq i \leq \epsilon - 1)$.

Consider an arbitrary node $v \in V_i$, for some i, such that $d(v) = 2a$. Exactly a edges in E_i connect v to nodes in the set V_{i+1}; call these the *outward* edges of v. Moreover, exactly a of these edges connect v to nodes in the set V_{i-1}; call these the *inward* edges of v. We define a port assignment as follows. For any node v, its outward edges receive the odd port numbers $1, 3, \ldots, 2a - 1$, and its inward edges receive the even port numbers $2, 4, \ldots, 2a$. The ports of nodes in V_0 and of nodes with degree a are assigned arbitrarily.

Regardless of the adversary's initial pointer assignment π_v, during the first phase of the exploration the agent visits at least all neighbors of s, i.e., at least all nodes in V_1. Therefore, during the second phase of exploration all nodes in V_1 become saturated which implies that all edges in E_0 and in E_1 are traversed in both directions.

Without loss of generality, we assume that V_ϵ contains at least the $(a + 1)$-st nodes of all chains. For the remaining part of the proof we consider only nodes and edges on the side of V_0 that contains the $(a + 1)$-st nodes of all chains. The proof for the other side is analogous. We claim that during the second phase of exploration the agent will visit at least one node in V_ϵ. To see why, first observe that for any node and during any phase of exploration, if x of the node's inward edges are traversed on the way out of v then, due to the alternating port assignment we adopted, at least $x - 1$ outward edges will be traversed also on the way out of v. Now, for any $i \geq 1$ let y_i be the number of edges in E_i that are traversed in the direction $(V_i \to V_{i+1})$ during the second phase of exploration. Since E_i separates s from the nodes in V_{i+1}, y_i edges in E_i must be also traversed in the direction $(V_{i+1} \to V_i)$ during the second phase. By the previous observation, at least $y_i - a$ edges of E_{i+1} will be traversed in the direction $(V_{i+1} \to V_{i+2})$.

Therefore, $y_{i+1} \geq y_i - a$. We have already established that $y_1 = a^2$. This recurrence boils down to $y_i \geq a^2 - (i-1)a$, which implies that for $i \leq \epsilon \leq a$ we have $y_i \geq a$. Thus, during the second phase of exploration the agent visits at least one of the $(a+1)$-st nodes of the chains.

It follows that every node in the graph is at distance at most 1 from some node visited during the second phase of exploration. Therefore, during the third phase the agent visits all nodes in the graph, and in the fourth phase it traverses all edges of the graph in both directions, achieving the Euler tour lock-in. Since during each phase the agent traverses at most m edges, each at most once in each direction, the upper bound of $8m$ for the lock-in time follows. □

4 Robustness under Dynamic Faults

In this section, we provide bounds on the additional stabilization period needed in case of dynamic faults or changes to the graph. We start with an analysis of the rotor-router assuming no faults or changes, which concludes with Theorem 9 in Section 4.1. We then employ Theorem 9 in Section 4.2, in order to derive bounds on the stabilization period after faults or changes to the system.

If T is a tree in graph G (not necessarily spanning all nodes of G), then $\vec{T}$ obtained from T by directing all edges towards a selected node v in T is called an *in-bound tree* in $\vec{G}$, and node v is the root of $\vec{T}$. A subset of arcs $\vec{H}$ in $\vec{G}$ 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 $\vec{H}$ 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).

In addition to the actual pointer π_v, we associate with each node a virtual *shadow pointer* σ_v, which is bound to always point to the port immediately before π_v in the cyclic order ρ_v. Note that, under regular operation of the rotor-router (i.e., in the absence of dynamic faults), σ_v always points to the port through which the agent exited on its last visit to node v. Let $\vec{F} = \{\sigma_v : v \in V\}$ be the set of the current shadow pointers. For the current node r, we are interested in the structure of $\vec{F}_r = \vec{F} \setminus \{\sigma_r\}$, since, as we show later, the structure of $\vec{F}_r$ is a good indicator of how far the agent is from entering an Eulerian cycle. The component of $\vec{F}_r$ containing the current node r is an *in-bound tree* rooted at r, which we call *the leading tree*. Each component $\vec{H}$ of $\vec{F}_r$ other than the leading tree is an *in-bound tree with a root cycle*.

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

4.1 Evolution of the leading tree

The following Propositions 1 and 2, which describe changes of the leading tree, can be easily verified.

Proposition 1. *If a node belongs to the current leading tree, then it remains in the leading tree in all subsequent steps.*

Proposition 2. *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.*

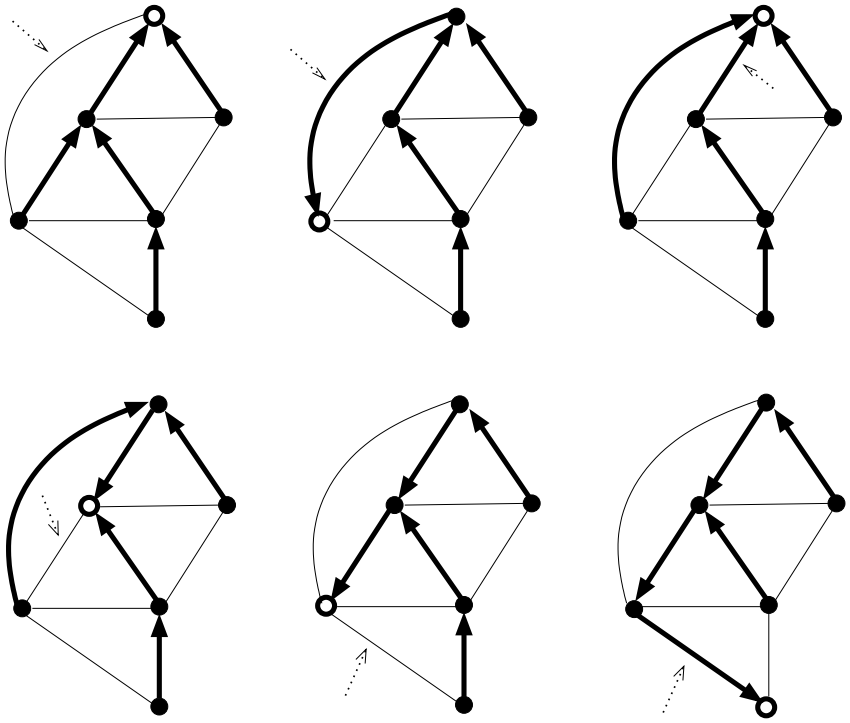

Fig. 3. The changing leading tree when the agent does not go outside the tree. The white node is the current node (and the root of the tree). The dotted arrow indicates the edge to be taken from the current node.

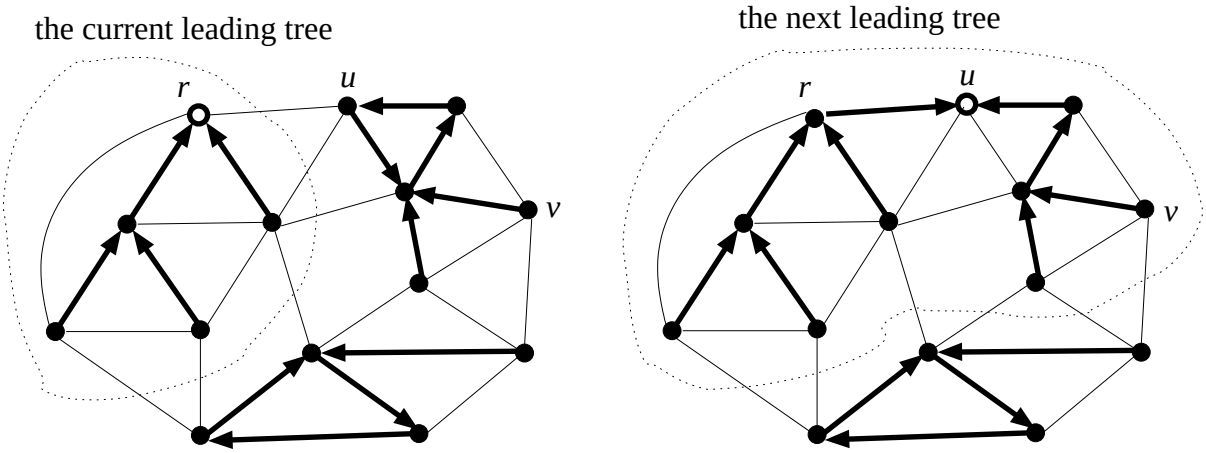

Fig. 4. The shadow pointers σ_x, $x \neq r$, are shown as boldface arrows. Left: the current step, when node v is outside the leading tree. Right: the next step, when the agent visits an ancestor u of v and v enters the leading tree.

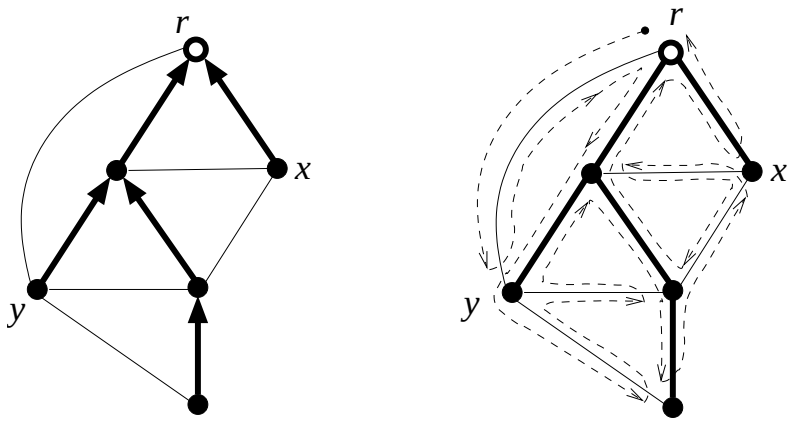

Fig. 5. Left: the leading tree spanning all nodes of the graph (arcs in bold). Right: the corresponding Eulerian cycle, assuming the anti-clockwise order of arcs outgoing from a node (different cycles are obtained for different cyclic orders of arcs).

Fig. 3 illustrates 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. 4 illustrates Proposition 2.

It has been shown in [17] that if the current leading tree spans all nodes of the graph, then the agent follows an Eulerian cycle in $\vec{G}$ during the next $2m$ steps.

Lemma 5 ([17, Lemma 4.9]). *Assume that the current leading tree $\vec{T}$ spans all nodes of the graph. Then during the next $2m$ steps the agent traverses an Eulerian cycle in $\vec{G}$. Moreover, the leading tree after these $2m$ steps is again the same tree $\vec{T}$.*

Fig. 5 illustrates Lemma 5. 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 shadow pointer σ_r. Thus the first arc followed by the agent is arc $(r, y) = \mathit{next}(r, x)$.

In fact, we can show that the condition that the agent follows an Eulerian cycle during the next $2m$ steps and the condition that the leading tree spans all nodes of the graph are equivalent:

Lemma 6. *Assume that at the current step i the leading tree $\vec{T}$ does not span all nodes of the graph. Then the route $\vec{\Gamma}$ traversed by the agent during the next $2m$ steps is not an Eulerian cycle.*

Proof. Let $\vec{A}$ be the set of arcs incoming to the leading tree $\vec{T}$ at the current step i, i.e., $\vec{A}$ contains arcs (v, u) with u in $\vec{T}$ and v outside of $\vec{T}$. Since, by assumption, $\vec{T}$ does not span all nodes of the graph, $\vec{A}$ is nonempty.

For a contradiction, suppose that $\vec{\Gamma}$ is an Euler cycle, i.e., each arc in $\vec{G}$ is traversed exactly once by $\vec{\Gamma}$. After the agent concludes $\vec{\Gamma}$, it is at the same node as at step i and the pointers and shadow pointers of all nodes are at the same positions as at step i. In particular, the leading tree is the same as at step i and, in view of Proposition 1, the node set of the leading tree has remained constant during the execution of $\vec{\Gamma}$. Let (v, u) be the last arc in $\vec{A}$ that is traversed during $\vec{\Gamma}$. Since the agent does not visit any nodes outside of the leading tree after traversing (v, u), this arc is the last outgoing arc from v that is traversed during $\vec{\Gamma}$. Therefore, if σ_v is the value of the shadow pointer of v at step i (and consequently also upon conclusion of $\vec{\Gamma}$), we must have $\sigma_v = (v, u)$. This implies that v is in the leading tree and thus contradicts our assumption that $(v, u) \in \vec{A}$. □

Definition 5 (Expansion time). *The* leading tree expansion time τ *is the number of steps taken by the rotor-router until the leading tree spans all nodes of* G *for the first time.*

By Lemma 5, we have the following corollary:

Corollary 2. *After* τ *steps, the agent keeps traversing the same Eulerian cycle.*

We refer to the Eulerian cycle which the agent keeps repeating after the leading tree expansion time as the *established Eulerian cycle*. Recall that, according to Definition 3, the stabilization time t_0 of the rotor-router is the number of steps until the agent has completed an Euler tour of $\vec{G}$ for the first time. The definitions of steps τ and t_0 are somewhat different, but these two times cannot be far apart. Lemma 5 implies that $t_0 \leq \tau + 2m$. On the other hand, since from step t_0 the agent follows an Eulerian cycle (Theorem 1), then Lemma 6 implies that $\tau \leq t_0$.

Corollary 3. $\tau \leq t_0 \leq \tau + 2m$.

Lemma 7 below can be viewed as a generalization of Lemma 5 to the case when the leading tree does not span all nodes. The neighborhood of the leading tree $\vec{T}$ consists of the nodes which are not in $\vec{T}$ but are adjacent to the nodes in $\vec{T}$.

Lemma 7. *Each node which is in the current step in the neighborhood of the leading tree* $\vec{T}$ *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 $j > i$. Let $\vec{\Gamma}$ denote the route traversed by the agent from step i up to step $j-1$. By Lemma 1, $\vec{\Gamma}$ does not use the same arc twice, so the length of $\vec{\Gamma}$ is at most $2m$.

For a node v in $\vec{T}$ (the leading tree at step i), let ℓ_v denote its distance from r in $\vec{T}$. We prove, by induction on ℓ_v, that each arc outgoing from each node in $\vec{T}$ is traversed during $\vec{\Gamma}$. For the root r of $\vec{T}$, the pointer π_r makes one full turn between steps i and j, so clearly all outgoing arcs from r are traversed during that period. Assuming that the claim holds for all nodes v of $\vec{T}$ with $\ell_v \leq k$, consider a node u with $\ell_u = k + 1$. If $\sigma_u = (u, v)$ at step i, then v is in $\vec{T}$ and $\ell_v \leq k$. By the inductive hypothesis, all outgoing arcs from v are traversed during $\vec{\Gamma}$. Combined with the fact that $\vec{\Gamma}$ does not use the same arc twice, this implies that all incoming arcs to v are also traversed during $\vec{\Gamma}$. In particular, let j' $(i < j' < j)$ be the step at which (u, v) is traversed and let $i' < i$ be the last step at which (u, v) was traversed. The pointer π_u makes one full turn between steps i' and j', so all outgoing arcs from u are traversed during that period. However, note that $\sigma_u = (u, v)$ at step i, so between i' and i no other outgoing arc from u is traversed. We conclude that all outgoing arcs from u are traversed between steps i and j'. □

Proposition 2 and Lemma 7 imply that a node v which is outside of the leading tree but has an ancestor u in the neighborhood 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 neighborhood of a node w which has an ancestor y in the neighborhood 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 formalize this, we define the *length of an arc* (v, u) as equal to 0, if $(v, u) \in \vec{F}_r$, 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 $\vec{G}$. 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.

Theorem 9. *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 neighbor 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 neighborhood of the leading tree within $2(k-1)m$ steps, so Lemma 7 implies that node u is visited within $2km$ steps. This and Proposition 2 imply that node v enters the leading tree within $2km$ steps. □

Observe also that the distance from one node to any other node is never greater than the diameter D of the graph. Therefore, Theorem 9 gives an alternative proof of the $\mathcal{O}(mD)$ bound shown in [25] on the number of steps required in the rotor-router model to enter an Eulerian cycle.

4.2 Faulty port pointers and dynamic changes of the graph

We are now ready to 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 9. 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 leading tree expansion period, inserting or deleting an edge $\{v, u\}$ may be harmless if this operation does not change the shadow pointers at nodes v and u. If the shadow pointers remain as they were, then the leading tree does not change, so it continues spanning all nodes and a new Eulerian cycle will be concluded in the next $2m$ steps (Lemma 5). 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.

In the following theorems, we bound the time that elapses between the step at which the fault or change appears in the graph and the step at which the leading tree spans all nodes. In view of Corollary 3, this yields immediately bounds on the respective re-stabilization times.

Theorem 10. *If k port pointers spontaneously change their values at some time after τ, then the leading tree will span all nodes within $2m \min\{k, D\}$ steps.*

Proof. Consider the leading tree right before those k changes of the port pointers. The leading tree expansion 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 9 implies that all nodes in the graph will be back in the leading tree within $2m \min\{k, D\}$ steps. □

Theorem 11. *If k new edges are added to the graph at some time after τ, then the leading tree in the new graph will span all nodes 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 10 implies that an Eulerian cycle is established in the new graph within $2m \min\{2k, D\}$ steps. □

Theorem 12. *If an edge $\{v, u\}$ is removed from the graph without disconnecting it at some time after τ, then the leading tree in the new graph will span all nodes within $2\gamma m$ steps, where γ is the smallest number of edges on a cycle containing edge $\{v, u\}$ in the original graph.*

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 10, 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 $\vec{G}$ 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 9 implies that all nodes in the graph are back in the leading tree within $2\gamma m$ steps. □

The bounds which appear in Theorems 10, 11, and 12 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. 6a). 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 10 is asymptotically tight in the worst case.

Likewise, by the construction shown in Fig. 6b, 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 11. 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. 6c 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 12.

5 Concluding Remarks

In this paper we have examined the dependence of the lock-in time on the initial configuration of the rotor-router mechanism. We have shown that it is advantageous to be in charge of pointer assignment in the rotor-router model. In all cases where the player $\mathcal{P}$ is responsible for pointer assignment the complexity of the lock-in problem is either linear or close to linear. In contrast, in all remaining cases where the adversary $\mathcal{A}$ controls assignment of pointers the worst-case complexity of the lock-in problem is always $\Omega(m \cdot D)$, i.e., the worst possible in view of Theorem 1.

In view of results from Subsection 3.3 a detailed study on the lock-in problem in more specific classes of graphs such as 2D-grids, planar or random graphs would be highly appreciated. This could be accompanied by a comparative study with the random walk procedure. Indeed, the lock-in time of a Propp machine is, in all the studied scenarios, equal up to constant factors to the time required to visit all the edges of the graph (its edge cover time). For example, in the $\mathcal{A}$-*all* scenario, the edge cover time using the Propp machine is precisely $\Theta(mD)$. This compares interestingly to the expected edge cover time of a graph when using random walk, which can be bounded as $\mathcal{O}(mD \log m)$. Whereas our bound for the Propp machine is tight for any graph, the bound for random walks is not; indeed, for a 2D-grid on $k \times k$ nodes we have a worst-case edge cover time of $\Theta(k^3)$ using the Propp machine, and an expected edge cover time of $\Theta(k^2 \log^2 k)$ using random walk [3].

We have also 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 stabilization

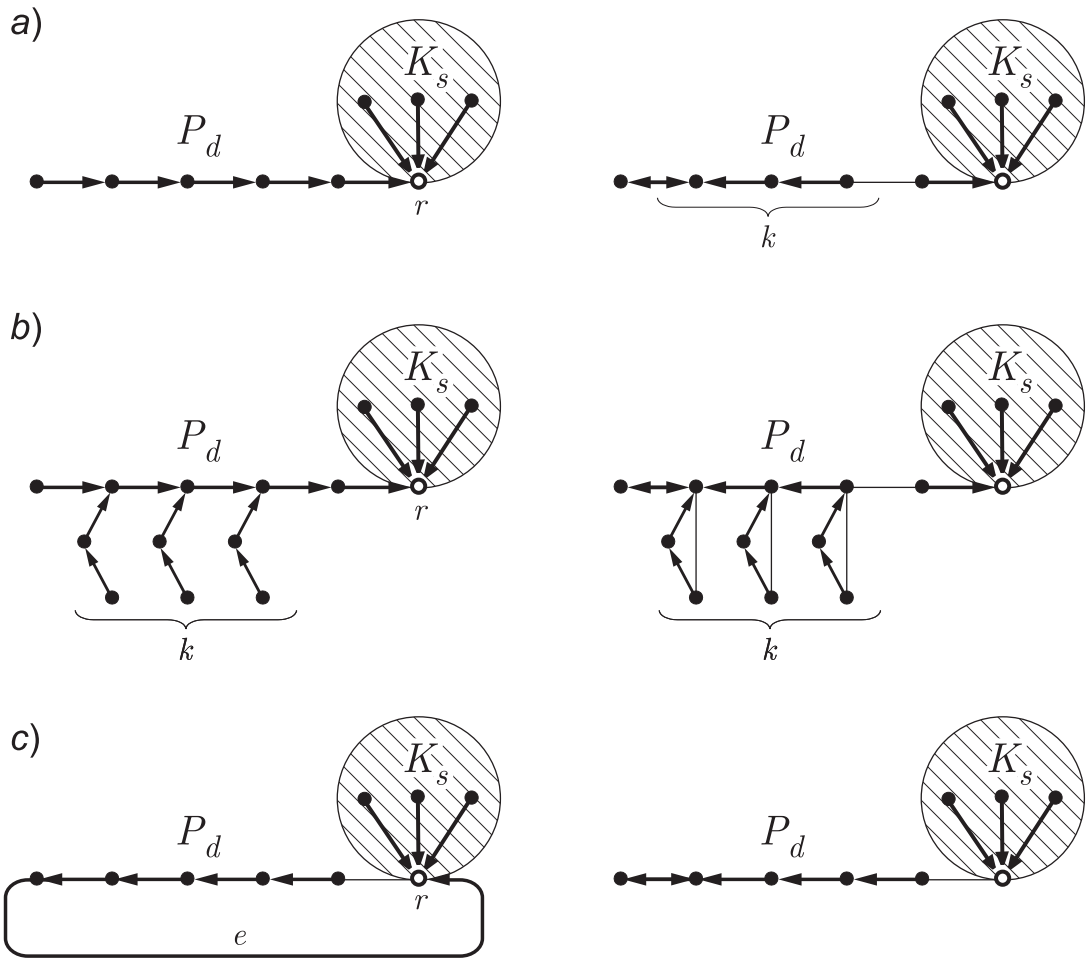

Fig. 6. Worst-case examples for the stabilization period of the rotor-router after changes to the graph: (a) modification of k port pointers, (b) addition of k edges, (c) removal of a single edge e.

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 local cyclic orders. More precisely, if at some step after the stabilization period the local cyclic orders change in any way but the shadow port pointers remain the same (that is, they point to the same edges as before), then the agent immediately enters a new Eulerian cycle (no need for any additional stabilization 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 stabilization and periodicity of exploration even in the static case (no faults, no dynamic changes of the graph).

Acknowledgment

We would like to thank Shay Kutten for several inspiring discussions during the early stages of this work.

References

1. Afek, Y., Gafni, E.: Distributed algorithms for unidirectional networks. SIAM J. Comput. 23(6), 1152–1178 (1994)
2. Akbari, H., Berenbrink, P.: Parallel rotor walks on finite graphs and applications in discrete load balancing. In: Blelloch, G.E., Vöcking, B. (eds.) 25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013. pp. 186–195. ACM (2013)
3. Aldous, D., Fill, J.A.: Reversible markov chains and random walks on graphs (2002), unfinished monograph, recompiled 2014, available at `http://www.stat.berkeley.edu/~aldous/RWG/book.html`
4. Ambühl, C., Gasieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. ACM Transactions on Algorithms 7(2), 17 (2011)
5. 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), 1–21 (2002)
6. Berenbrink, P., Klasing, R., Kosowski, A., Mallmann-Trenn, F., Uznanski, P.: Improved analysis of deterministic load-balancing schemes. In: Georgiou, C., Spirakis, P.G. (eds.) Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015. pp. 301–310. ACM (2015)
7. Bhatt, S.N., Even, S., Greenberg, D.S., Tayar, R.: Traversing directed eulerian mazes. J. Graph Algorithms Appl. 6(2), 157–173 (2002)
8. Chalopin, J., Das, S., Gawrychowski, P., Kosowski, A., Labourel, A., Uznanski, P.: Lock-in problem for parallel rotor-router walks. CoRR abs/1407.3200 (2014)
9. Cooper, C., Ilcinkas, D., Klasing, R., Kosowski, A.: Derandomizing random walks in undirected graphs using locally fair exploration strategies. Distributed Computing 24(2), 91–99 (2011)
10. Cooper, J.N., Spencer, J.: Simulating a random walk with constant error. Combinatorics, Probability & Computing 15(6), 815–822 (2006)
11. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. Journal of Graph Theory 32(3), 265–297 (1999)
12. Dereniowski, D., Kosowski, A., Pajak, D., Uznanski, P.: Bounds on the cover time of parallel rotor walks. In: 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France. pp. 263–275 (2014)
13. Doerr, B., Friedrich, T.: Deterministic random walks on the two-dimensional grid. Combinatorics, Probability & Computing 18(1-2), 123–144 (2009)
14. Fraenkel, A.S.: Economic traversal of labyrinths. Mathematics Magazine 43(3), 125–130 (1970)
15. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theor. Comput. Sci. 345(2-3), 331–344 (2005)
16. Gasieniec, L., Radzik, T.: Memory efficient anonymous graph exploration. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG. Lecture Notes in Computer Science, vol. 5344, pp. 14–29 (2008)
17. Holroyd, A.E., Levine, L., Mészáros, K., Peres, Y., Propp, J., Wilson, D.B.: Chip-firing and rotor-routing on directed graphs. Progress in Probability, vol. 60, pp. 331–364. Birkhäuser Basel (2008)
18. Klasing, R., Kosowski, A., Pajak, D., Sauerwald, T.: The multi-agent rotor-router on the ring: a deterministic alternative to parallel random walks. In: ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013. pp. 365–374 (2013)

19. Kosowski, A., Pajak, D.: Does adding more agents make a difference? A case study of cover time for the rotor-router. In: Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II. pp. 544–555 (2014)
20. Priezzhev, V.B., Dhar, D., Dhar, A., Krishnamurthy, S.: Eulerian walkers as a model of self-organized criticality. Phys. Rev. Lett. 77, 5079–5082 (Dec 1996), http://link.aps.org/doi/10.1103/PhysRevLett.77.5079
21. Reingold, O.: Undirected connectivity in log-space. J. ACM 55(4) (2008)
22. Strahler, A.N.: Hypsometric (area-altitude) analysis of erosional topography. Geological Society of America Bulletin 63(11), 1117–1142 (1952)
23. Tutte, W.T., Smith, C.A.B.: On unicursal paths in a network of degree 4. The American Mathematical Monthly 48(4), 233–237 (1941)
24. Wagner, I.A., Lindenbaum, M., Bruckstein, A.M.: Distributed covering by ant-robots using evaporating traces. IEEE T. Robotics and Automation 15(5), 918–933 (1999)
25. Yanovski, V., Wagner, I.A., Bruckstein, A.M.: A distributed ant algorithm for efficiently patrolling a network. Algorithmica 37(3), 165–186 (2003)