Euler Tour Lock-In Problem in the Rotor-Router Model*

I Choose Pointers and You Choose Port Numbers

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

[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***
{nicolas.hanusse,david.ilcinkas,ralf.klasing}@labri.fr
[4] Dept of Algorithms and System Modeling, Gdańsk Univ. of Technology, Poland
adrian@kaims.pl

Abstract. The *rotor-router model*, also called the *Propp machine*, was first considered as a deterministic alternative to the random walk. It is known that the route in an undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$, 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 recent work [11] Yanovski et al. 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. The case study 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. First, we observe that in certain (easy) cases the lock-in can be achieved in time $O(m)$. On the other hand we show that if adversary $\mathcal{A}$ is solely responsible for the assignment of ports and pointers, the lock-in time $\Omega(m \cdot D)$ can be enforced in any graph with m edges and diameter D. Furthermore, we show 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 bounded by $O(m \cdot \min\{\log m, D\})$. We also propose a class of graphs in which the lock-in requires time $\Omega(m \cdot \log m)$. In the remaining two cases we show that the lock-in requires time $\Omega(m \cdot D)$ in graphs with the worst-case topology. In addition, however, we present non-trivial classes of graphs with a large diameter in which the lock-in time is $O(m)$.

* This work was done during the visit of the second author in Bordeaux.
** 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.

I. Keidar (Ed.): DISC 2009, LNCS 5805, pp. 423–435, 2009.

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 based on graphs, very 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.

The family of *anonymous graphs* provides foundations for a model that has found its application in network communication, graph exploration and stabilisation of distributed processes. In principle, due to minimalistic assumptions, any solution provided in this model constitutes also a valid solution in any other communication graph-based model. 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.

The *rotor-router mechanism* was introduced as a deterministic alternative to the random walk and studied in the context of a wide selection of network problems, including work on load balancing problems in [6, 5], graph exploration [7,2,8], and stabilisation of distributed processes [9,3,11]. The rotor-router mechanism is represented by an undirected anonymous graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. The nodes in V bear no names, however, the endpoints of edges in E, called *ports*, are arranged in a *cyclic order* at each node. Furthermore, each node is equipped with a *pointer* that indicates the current exit port to be adopted by an agent on the conclusion of the next visit to this node. The rotor-router mechanism guarantees that after each consecutive visit at a node its pointer is moved to the next port in the cyclic order. Due to a limited number of configurations in a graph G of a bounded size it is intuitive that a walk of the agent controlled by the rotor-router mechanism must be locked-in in a loop eventually. Rather surprisingly, however, Priezzhev et al. [9] proved that an agent traversing a finite graph gets locked-in in an Euler tour based on arcs obtained by replacing each edge in G with two arcs having opposite directions. Later, Bhatt et al. [3] proved that the lock-in time is bounded by $O(m \cdot n)$. This bound was further improved by Yanovski et al. in [11] to $2mD$, where D is the diameter of G. Related models of traversal in undirected graphs were studied in [4].

1.1 Our Contribution and Outline of the Paper

In this paper 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 time $O(m)$. Also the case $\mathcal{A}(\circlearrowleft)\mathcal{P}(f)$, where $\mathcal{P}$ sets the pointers after the port numbers are assigned by $\mathcal{A}$, reduces to the border case where $\mathcal{P}$ is solely in charge of the initial configuration. On the other hand, in the case $\mathcal{A}$-*all* where the adversary $\mathcal{A}$ solely decides about the initial configuration, we show that in any graph with m edges and diameter D the adversary $\mathcal{A}$ is able to enforce the lower bound $\Omega(m \cdot D)$ for the lock-in matching the upper bound from [11].

Furthermore, we show that if $\mathcal{A}$ provides its own port numbering after the initial setup of pointers by $\mathcal{P}$, case $\mathcal{P}(f)\mathcal{A}(\circlearrowleft)$, the complexity of the lock-in problem is bounded by $O(m \cdot \min\{\log m, D\})$. We also propose a respective 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 $O(m)$.

We conclude with the proof that in the remaining two cases the lock-in requires time $\Omega(m \cdot D)$ in graphs with the worst-case topology. In the case $\mathcal{A}(f)\mathcal{P}(\circlearrowleft)$ where $\mathcal{P}$ responds by appropriate port assignment to initial setup of pointers by $\mathcal{A}$, we show that 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 $O(m)$. Finally, in the case $\mathcal{P}(\circlearrowleft)\mathcal{A}(f)$ 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 $O(m)$. Our results are summarised in Table 1.

Table 1. Minimum and maximum values of the lock-in time in considered cases

Scenario	Worst case	Best case
Case $\mathcal{P}$-*all*	$\Theta(m)$	$\Theta(m)$
Case $\mathcal{A}(\circlearrowleft)\mathcal{P}(f)$	$\Theta(m)$	$\Theta(m)$
Case $\mathcal{P}(f)\mathcal{A}(\circlearrowleft)$	$\Theta(m \cdot \min\{\log m, D\})$	$\Theta(m)$
Case $\mathcal{A}(f)\mathcal{P}(\circlearrowleft)$	$\Theta(m \cdot D)$	$\Theta(m)$
Case $\mathcal{P}(\circlearrowleft)\mathcal{A}(f)$	$\Theta(m \cdot D)$	$\Theta(m)$ for all $D \leq \sqrt{n}$
Case $\mathcal{A}$-*all*	$\Theta(m \cdot D)$	$\Theta(m \cdot D)$

1.2 The Euler Tour Lock-In Problem Revisited

In this section we provide basic definitions and we recall known facts in relation to performance of the rotor-router mechanism in anonymous graphs. Recall that $G = (V, E)$ is an input graph in which the starting node is denoted by s.

Definition 1. *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 $4m$.*

Definition 2. *For any $v \in V$ let $E_G(v)$ denote the set of edges of G that are incident to node v.*

Definition 3. *Let $H = (X, C)$ be a connected subgraph of G induced by some $C \subseteq E$. We denote by $N_G(H)$ the subgraph of G induced by the set $\bigcup_{v \in X} E_G(v)$.*

Definition 4 (port assignment). *A port assignment to the nodes of graph G is a collection of bijective functions between $E_G(v)$ and $\{1, \ldots, \deg(v)\}$, one for each $v \in V$.*

Definition 5 (pointer assignment). *An initial pointer assignment to the nodes of an undirected graph $G = (V, E)$ is a function $f : V \to E$, s.t., for all $v \in V$, $f(v) \in E_G(v)$.*

Definition 6. *A node becomes saturated when all its incident edges are traversed in both directions for the first time.*

Note that when a node becomes saturated, its pointer returns to the initial position for the first time.

Lemma 1 ([3]). *Let $G = (V, E)$ be a graph with a starting node $s \in V$, an assignment of ports and pointers. The Euler tour lock-in in G is performed in phases $\{P_i\}_{i \geq 1}$. Each phase starts when the mobile agent leaves s via edge $f(s)$ indicated by the initial assignment of pointers and continues until the agent traverses all edges incident to s in both directions. The following properties hold:*

- *While the agent is visiting nodes saturated in some earlier phase, it retraces the route of phase P_{i-1}.*
- *If the agent encounters a node u that has been visited but not saturated in an earlier phase, it suspends the retracing of the tour of phase P_{i-1}. A new tour starts at u and ends there. Node u is now saturated. The tour of phase P_{i-1} is resumed (via port $f(s)$).*
- *Every edge is traversed at most once in each direction during each phase.*

Eventually all nodes in G get saturated. In other words, there exists integer $j \geq 1$, s.t., starting from the phase P_j the agent adopts the same (Euler) tour in G.

One can conclude from Lemma 1 that during each phase P_i the agent gets locked-in in a subgraph G_i of G where (1) G_0 contains a single node s; (1) each G_i is a subgraph of G_{i+1}; and (2) all edges in G that are incident to nodes in G_i are present in G_{i+1}, i.e., $N_G(G_i) \subseteq G_{i+1}$. Since the number of edges in each G_i is bounded by m the following theorem follows.

Theorem 1 ([11]). *For any graph, any starting node, and any initial pointer and port assignments, the lock-in (which is equivalent with exploration of all edges in G) is achieved in time $O(m \cdot D)$.*

2 Case Study of the Lock-In Problem

In this section we study the game between $\mathcal{P}$ and $\mathcal{A}$ in detail.

2.1 Border Cases

In this section we briefly discuss border cases in which for any graph G the complexity of the lock-in problem is either $\Theta(m)$ or $\Theta(mD)$.

Cases with Lock-in Time $\Theta(m)$. Consider first the case $\mathcal{P}$-*all* where the player $\mathcal{P}$ is solely responsible for the initial setup of port numbers and pointers. Since we assume unbounded computational power of $\mathcal{P}$ clearly the player can choose a configuration that locks-in the agent in an Euler tour right from the beginning. Similarly, also in the case $\mathcal{A}(\circlearrowleft)\mathcal{P}(f)$, after the adversary $\mathcal{A}$ sets port numbers, $\mathcal{P}$ can respond with an appropriate assignment of pointers that leads to an Euler tour instantly for any input graph G. Thus in those two cases the agent visits all edges in G locking-in itself in an Euler tour in time $O(m)$.

Case with Lock-in Time $\Theta(m \cdot D)$. 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 proof of the complexity $\Theta(m \cdot D)$ is more complex. We start with the following lemma.

Lemma 2. *Given an input graph $G = (V, E)$ with a starting node $s \in V$. For any subset $C \subseteq E$, s.t. 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, s.t., 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 H. Fix the corresponding sequence of edge traversals $e_1, \ldots, e_{2|C|}$, s.t. e_1 is an edge incident to s (each undirected edge in C is traversed exactly twice by $\mathcal{C}$, once in each direction). We now define a port assignment and an assignment of pointers f to the nodes of G.

For any node $v \in V$, 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 $k < \deg_G(v)$ 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, s.t., for any $i \leq k$, edge e_{v_i} is the port with number i. If $k < \deg_G(v)$, extend this port assignment, s.t., edges in $E \setminus C$ receive higher port numbers than edges in C. Finally, define $f(v)$ to be the edge e_{v_1} if $k \geq 1$, otherwise, define $f(v)$ to 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 3. *Let $G = (V, E)$ be an undirected graph with a starting node $s \in V$, a given port assignment, and a pointer assignment f 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, s.t., during Phase $i+1$ of exploration the agent traverses all edges of $N_G(H)$ in both directions, but it traverses no further 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 the second phase. Therefore all edges in D will be traversed in both directions.

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 $f(v)$ 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 (toward v) during Phase $i+1$. This leads to a contradiction, since an edge is never traversed twice in the same direction during the same phase. □

Theorem 2. *For any graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ there exists a starting node s, and a port and pointer assignment in G, s.t., the lock-in requires time at least $\frac{1}{4} \cdot mD$.*

Proof. Let T be the BFS tree of G rooted in an arbitrary node $u \in V$. Let $r \geq \frac{D}{2}$ be the height of T. Finally, let H be the subgraph of G induced by the nodes of the lowermost $\frac{r}{2}$ levels of T, and let $H_1 = (X, C)$ be a connected component of H that contains at least one node from the r-th level of T (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, s.t., the first phase of the exploration starting from s explores exactly $G_1 = H_1$. This is feasible due to Lemma 2. 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 multiple applications of Lemma 3. In this case, the exploration from s will require at least $2 \cdot \frac{m}{2} \cdot \frac{r}{2} \geq \frac{1}{4} \cdot mD$ 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, s.t., $G_1 = H_2$ and for any $i \geq 2$, $G_i = N_G(G_{i-1})$. The exploration will again require at least $\frac{1}{4} \cdot mD$ edge traversals before visiting all nodes. □

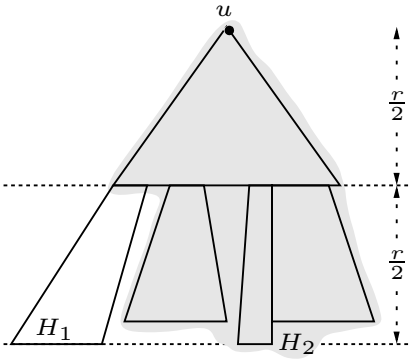

Fig. 1. The partition of G into subgraphs that is described in the proof of Theorem 2. Either subgraph H_1 or subgraph H_2 contains at least half of the edges of G.

2.2 Almost Linear Lock-In – Case $\mathcal{P}(f)\mathcal{A}(\circlearrowleft)$

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 3. *For any graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ and any starting point s there exists a pointer assignment, s.t. for any port assignment the lock-in can be obtained in time $O(m \cdot \min\{\log m, D\})$.*

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

Take an arbitrary BFS tree T in G rooted in 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 top rank r belongs to only one child the node v adopts r as its own rank. Otherwise, i.e., when the top rank is shared by at least two children the node v adopts the rank $r + 1$. One can prove that the rank ρ 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 [10]. 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 $\leq \rho - i + 1$. And indeed G_1 contains all nodes with rank ρ 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 $\leq \rho - i + 1$ belong to G_i. These include the nodes that are connected to downward chains with rank $\rho - i$ with nodes still not present in G_i. But note that due to Lemma 1 all edges incident to nodes in G_i are present in G_{i+1} which means that each downward chain with rank $\rho - i$ will be accessed and all of their nodes will be traversed

when G_{i+1} is formed. This proves that G_ρ contains all nodes from G and $G_{\rho+1}$ contains all nodes and edges. Since $\rho \leq \min\{\log m, D\}$ and the number of edges in each G_i is bounded by m, the lock-in time is $O(m \cdot \min\{\log m, D\})$. □

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 complexity of the lock-in problem in such graphs is $O(m)$.

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

Theorem 4. *For any m and $D \leq m$, there exists a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ with a starting node s, s.t., for any pointer assignment there is a port assignment for which the lock-in requires time $\Omega(m \cdot \min\{\log m, D\})$.*

Proof. Consider a graph formed of a complete graph K with $\Omega(m)$ edges and $O(\sqrt{m})$ nodes connected by a path of length $\max\{1, D - \log m\}$ with a complete binary tree B of height $\min\{\log m, D\}$. 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 pointer at an internal node v, if i is the smallest integer such that v belongs to G_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 after the agent arrives in 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, see Lemma 1, 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 and 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 is to ensure 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, see Lemma 1, 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 two consecutive nodes belong to the same G_i.

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

2.3 The Two Remaining Cases

In the last part of the paper 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 lock-in time of $O(m)$.

Case $\mathcal{A}(f)\mathcal{P}(\circlearrowleft)$. 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 locking-in 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 lock-in is achieved in time $O(m)$.

Theorem 5. *For any m and $D \leq m$, there is a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ with starting node s, and a pointer assignment, s.t. for any port assignment the lock-in requires time $\Omega(mD)$.*

Proof. Let G be a lollipop graph obtained by connecting a complete graph K_a to a path P_D with D nodes via a bridging edge. The diameter of G is $D + 1$ and it is always possible to pick $a = \Theta(\sqrt{m})$, s.t., the number of edges of G is between m and $4m$. Thus $G \in \mathcal{G}_{m,D}$. Let s be a node of K_a different from the node connecting K_a to P_D, and let the pointers within K_a point towards s (the pointer of s can initially be on an arbitrary port). Finally, set the pointers at each node of P_D towards K_a.

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_a. During the second phase the agent will traverse all $\Theta(m)$ edges of K_a in both directions, and it will return to K_a 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_a; the lower bound of $\Omega(m \cdot D)$ for the lock-in time follows. □

Theorem 6. *For any m and $D \leq m$, there is a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$, s.t., for any starting node s and for any pointer assignment, there exists a port assignment for which the lock-in is achieved in time $\leq 24m$.*

Proof. Let $G = (V, E)$ be a chain of length D of complete bipartite graphs $K_{2,2}$. The number of edges in G 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 Figure 2. In both cases, the diameter of 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 f be the pointer assignment supplied by the adversary $\mathcal{A}$. Denote the set of

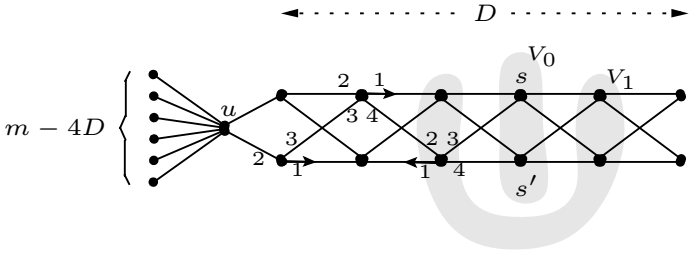

Fig. 2. The construction described in the proof of Theorem 6

nodes of the chain of complete bipartite graphs 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 opposite to s side in the chain (i.e., the node 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 $f(u)$. If $f(u)$ connects u to the chain, assign port number $\deg(u)$ to the other edge that connects u to the chain, and the rest of the ports arbitrarily. Otherwise, assign port numbers $\deg(u) - 1$ and $\deg(u)$ to the edges connecting u to the chain, and set the rest of the port numbers arbitrarily.
- For a node $v \in X$ at the endpoint of the chain, if v is not connected to u then set ports arbitrarily. If v is connected to u, then assign port 1 to $f(v)$ and assign the smallest possible port to the edge connecting v to u (if it is not $f(v)$).
- For any other $v \in X$, port 1 is always assigned to $f(v)$. Let $i \geq 1$ be the distance of v from s, thus $v \in V_i$. If $f(v)$ 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 an analogous 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 $\mathcal{P}(\circlearrowleft)\mathcal{A}(f)$. In the case where the adversary $\mathcal{A}$ responds by a pointer assignment to player $\mathcal{P}$'s initial port assignment, we first prove existence of a family of graphs in which locking-in requires time $\Omega(mD)$, matching the general worst-case upper bound from Theorem 1.

Theorem 7. *For any m and $D \leq m$, there is a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$ with a starting node $s \in V$, s.t. for any port assignment there exists a pointer assignment under which the lock-in time in G is $\Omega(mD)$.*

Proof. Follows immediately from Theorem 5. □

We show, however, that there is also a non-trivial class of graphs with diameter $O(\sqrt{n})$ in which the lock-in time is $O(m)$ in this case.

Theorem 8. *For any $D \leq \sqrt{n}$, there is a graph $G = (V, E)$ in $\mathcal{G}_{m,D}$, s.t., for any starting node s, there is a port assignment, s.t., for any possible pointer assignment the lock-in time is $O(m)$.*

Proof. For any a, let $G_a = (V, E)$ be the graph consisting of a chain of length a of complete bipartite graphs $K_{a,a}$. We will show that for any starting node s, there is a port assignment, s.t., for any possible pointer assignment the lock-in time in G_a is bounded by $8m$, where $m = |E|$.

Let $s \in V$ be a starting node with eccentricity ϵ. Let V_0 be the subset of nodes on the same level of the chain as 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}.

Consider an arbitrary node $v \in V_i$, for some i. The degree of this node is $2a$. 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 are assigned arbitrarily.

Regardless of the adversary's initial pointer assignment f, 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.

For the remaining part of the proof we consider only nodes and edges on the side of V_0 that contains V_ϵ. 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. For the sake of the proof, 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 $(V_i \rightarrow V_{i+1})$ 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 $(V_{i+1} \rightarrow V_i)$ during the second phase. By the previous observation, at least $y_i - a$ edges of E_{i+1} will be traversed in the direction $(V_{i+1} \rightarrow V_{i+2})$. 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 node at the end-point of the chain.

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. □

3 Further Work and Open Problems

Herein 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 2.3 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 $O(mD \log m)$. 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 [1].

One could also imagine a game in which a player and its adversary choose assignments of ports and pointers in consecutive nodes visited by the agent in alternative turns. What is the complexity of such a game?

Acknowledgement

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

References

1. Aldous, D., Fill, J.: Reversible Markov Chains and Random Walks on Graphs (2001), http://stat-www.berkeley.edu/users/aldous/RWG/book.html
2. Afek, Y., Gafni, E.: Distributed Algorithms for Unidirectional Networks. SIAM Journal on Computing 23(6), 1152–1178 (1994)
3. Bhatt, S., Even, S., Greenberg, D., Tayar, R.: Traversing Directed Eulerian Mazes. Journal of Graph Algorithms and Applications 6(2), 157–173 (2002)
4. Cooper, C., Ilcinkas, D., Klasing, R., Kosowski, A.: Derandomizing Random Walks in Undirected Graphs Using Locally Fair Exploration Strategies. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Niko-letsea, S. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 411–422. Springer, Heidelberg (2009)
5. Cooper, J.N., Spencer, J.: Simulating a random walk with constant error. Combinatorics, Probability and Computing 15, 815–822 (2006)
6. Doerr, B., Friedrich, T.: Deterministic Random Walks on the Two-Dimensional Grid. Combinatorics, Probability and Computing 18(1-2), 123–144 (2009)
7. Fraenkel, A.S.: Economic traversal of labyrinths. Mathematics Magazine 43, 125–130 (1970)
8. Gąsieniec, L., Radzik, T.: Memory efficient anonymous graph exploration. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 14–29. Springer, Heidelberg (2008)
9. Priezzhev, V.B., Dhar, D., Dhar, A., Krishnamurthy, S.: Eulerian walkers as a model of selforganized criticality. Physics Review Letters 77, 5079–5082 (1996)
10. Strahler, A.N.: Hypsometric (area-altitude) analysis of erosional topography. Geological Society of America Bulletin 63(11), 1117–1142 (1952)
11. Yanovski, V., Wagner, I.A., Bruckstein, A.M.: A Distributed Ant Algorithm for Efficiently Patrolling a Network. Algorithmica 37, 165–186 (2003)