Yale University
Department of Computer Science

Oblivious periodic traversal of anonymous, undirected graphs with advice and adversarial port numbers

John Maheswaran

YALEU/DCS/TR-1475
April 2013

Oblivious periodic traversal of anonymous, undirected graphs with advice and adversarial port numbers

John Maheswaran[*]
Yale University
Department of Computer Science
john.maheswaran@yale.edu

Abstract

We consider the problem of periodic graph traversal [2], which has previously been studied in a variety of settings [2, 5, 3, 7, 13, 12, 10, 11]. The problem of periodic graph traversal is concerned with an agent having to visit every node of a graph and return to its start location and state. The periodic graph traversal problem can be extended by labeling each node to help the agent explore the graph [3]. We consider oblivious agents (agents with no persistent memory) and graphs with low maximum degree. Ports (the points where edges are incident to vertices) are numbered by an adversary.

We prove that two labels are both necessary and sufficient to explore all degree three bounded graphs and prove that there are only two (trivially isomorphic) agent algorithms that can explore all such graphs. We provide a two label labeling algorithm that allows these agent algorithms to explore all such graphs in a period of length $4n-2$ where n is the number of vertices in the graph.

For the special case of Hamiltonian graphs, which includes almost all regular graphs [17], we provide an algorithm that improves the period length to n using $\Delta(G)$ labels.

In the case of degree three bounded graphs, we provide a 5-label algorithm that improves the period length to $2\frac{1}{2}n-1$.

In the general case of all graphs, the best possible worst-case period length is $2n-2$. We give a $2^{\Delta(G)}-1$ label algorithm (7-label for degree three bounded graphs) that achieves this for all graphs.

Finally we extend our results to graphs of higher degree and prove upper bounds of 6 and 24 on the number of labels required for an agent

[*]Supported in part by NSF grant CCF-0916389

to explore all degree 4 and 5 bounded graphs respectively. We also give agent and labeling algorithms for degree 4 and 5 bounded graphs that give period lengths of $4n-2$. This result relies on reducing the problem to the previously studied problem of assigning port numbers[7, 5, 13] to allow the agent to traverse the graph. The period length we give of $4n-2$ is equal to the best known period length in the case of assigning port numbers benevolently, although this is still above the theoretical lower bound of $2n-2$.

1 Introduction

We consider the problem of periodic graph traversal [2] with advice [3]. In this problem, an agent (also referred to as a robot) is tasked with exploring a graph and visiting all the nodes in that graph. The nodes of the graph have advice labels that help the agent to navigate the graph.

The agent or mobile entity may be used to model a variety of scenarios, for example a piece of software navigating through a computer network or a robot navigating an unknown terrain or environment. It may be the case in a practical application that the agent is just tasked with visiting every location infinitely often, however due to the fact that the agent is fi4nite-state, the traversal must be periodic. The agent aims to visit all nodes eventually returning to its start node and start state after one traversal. This is known as a *periodic traversal* or traversing the graph in a *periodic manner*. The robot may be modeled as a finite state machine (FSM), however in this paper we are concerned with oblivious agents, that is agents which are stateless, or equivalently single state FSMs. We consider anonymous, undirected graphs. The edges do not have weights or labels.

The periodic graph traversal problem is hard from the point of view of an oblivious agent as the agent has no persistent memory so as soon as it leaves one node, it forgets that it has been there. It does not know which nodes it has visited and which it has not and yet it still has to perform a traversal of all nodes. In a completely unlabeled graph it is not possible in general for the agent to perform a periodic traversal [2]. So there still needs to be some way for the agent to tell where to go. Nodes do not have *unique* labels, however the edges of the graph are labeled with *port numbers* that identify the points where edges are incident to vertices.

Benevolent port numbers are assigned by an algorithm in a pre-processing step, whereas *adversarial* port numbers are assigned by an adversary and not under control of the algorithm. When an agent enters a node it knows the port number that it entered the node through. In this paper we primarily consider adversarial port numbers in Section 4.2 however we do consider benevolent port numbers in Section 4.1.

1.1 Periodic graph traversal

The periodic graph traversal model consists of an autonomous agent and a graph in which the agent is situated. The agent has the goal of visiting every node in the graph repeatedly. The period length is defined to be the maximum number times the agent crosses an edge between visiting an arbitrary node twice where the agent enters that node in the same state (if a non-oblivous FSM) through the same port [7].

Note that the agent may visit certain nodes multiple times during one period of the traversal. It is not necessarily the case that every edge of the graph will be traversed by the robot, just that every node is visited. The agent may be modeled as either a finite state machine (FSM) in which case it has a constant amount of memory, or as an oblivious agent in which case it is a single state FSM with no persistent memory. The model follows the formalism of Ilcinkas [12].

1.2 Port numbers

In the periodic traversal model, the graph is connected and has undirected edges. Edges are not labeled either so the graph is referred to as being *anonymous* however the points where the edges are incident to a given node, known as ports, are numbered.

Definition 1. *A* port *is defined as a pair* $(v, (v, v'))$ *of a node* v *and an edge incident to that node,* (v, v')

These port numbers may be provided benevolently by a preprocessing algorithm, or may be supplied by an adversary as part of the unknown graph. Port numbers are in the range zero to the degree of the node minus one, $\{0, 1, ..., d(v) - 1\}$.

1.3 Agent

The agent starts off exploring the graph in start state s_0, at an arbitrary node and port selected by an adversary. From this the agent can see only the port numbers of the edges connected to the node it is located at. It does not know the endpoints of the edges and indeed has no other knowledge of the graph. The agent may only pick a port through which to exit the current node.

When the agent arrives at a new node it knows the port number through which it entered the node and the degree of the node. The agent starts off entering an initial node through an initial port. If the agent is a non-oblivious FSM it also has current state. In this paper we only consider oblivious FSMs where the agent has only a single state. The agent computes its exit port based on the input port and the degree of the node. In the basic oblivious model the exit port depends only on the input port. Later we will extend this to allow the exit port to depend on the label of the node as well. This involves a preprocessing step of adding labels to nodes. An agent's behavior is described by its transition function f.

Definition 2. *For the case of benevlent port numbers, we use the following definition of* transition function equivalence *taken from Czyzowicz* et al. *[5]. Two agent algorithms with transition functions f and f' are said to be* equivalent *if for any $d > 0$ (where d is node degree) there exists a permutation σ on $\{0, \ldots, d-1\}$ such that $f'(i, d) = \sigma_d^{-1}(f(\sigma_d(i), d))$ (written by as $f' = \sigma \circ f \circ \sigma^{-1}$ for brevity).*

It has been proven [5] that in the basic oblivious model with benevolent port numbers, any agent that successfully traverses all graphs with benevolent port numbers is equivalent to one that operates under the *right-hand rule.* Using this rule, when it enters a node through a port numbered n, it exits through port number $(n + 1) \bmod (d(v))$ where $d(v)$ is the degree of the node v.

1.4 Advice

The model may optionally be extended by adding small amounts of extra information to each node, known as advice. Work focuses on local advice situated at each node [3, 11]. As in the case of the existing research on advice, we also consider local advice[1].

This advice serves as a sort of signpost to help guide the agent through the graph. In this paper we measure the amount of advice in terms of the number of labels $|C|$ where C is the set of labels. A graph with $|C| = 1$ is said to not have advice.

Advice in the context of this model was first introduced by Cohen *et al.* [3] where they primarily investigated labeling schemes for *non-oblivious* finite

[1]There is also the less extensively studied idea of global advice where the agent is given some information about the graph topology prior to performing its traversal. For example one of the agent algorithms of Cohen *et al.* [3] relies on knowing the maximum degree of the graph ahead of time.

$\Delta(G)$	Number of labels used	Special case
≥ 5	$> \lfloor \log \Delta(G) \rfloor - 2$	
	$> \frac{\Delta(G)}{2} - 1$	graphs with loops

Table 1: Previous results of Cohen *et al.* [3]

$\Delta(G)$	Number of labels used	Period length
	2	$4n - 2$
3	5	$(2\frac{1}{2})n - 1$
	7	$2n - 2$
4	6	$4n - 2$
5	24	$4n - 2$
any	$2^{\Delta(G)} - 1$	$2n - 2$

Table 2: Our results - general cases

state automata and also proved several impossibility results for *oblivious* FSMs.

1.5 Summary of paper

In this paper we consider several new aspects of the graph exploration model. We first give a formal description of the model (Section 2) and then provide a survey of related work (Section 3). We then present our new results (Section 4). We consider extending the benevolent ports model with advice and provide an upper bound on the period length (Section 4.1). We consider the special case of Hamiltonian graphs in the benevolent ports model without advice and provide an upper bound on period length (Section 4.1.1). We also examine the adversarial ports model with advice and provide an upper bound on the period length for both the special case of Hamiltonian graphs and for arbitrary graphs (Section 4.2). For all degree three bounded graphs we give an algorithm to explore in period length $(2\frac{1}{2})n - 1$ using $|C| = 5$ labels (Section 4.2.3). We then show an exact bound of $|C| = 2$ required to explore all degree three bounded graphs, however this leads to a longer period length of $4n - 2$ (Section 4.2.4). We then extend the result to give upper bounds of $|C| \leq 6$ and $|C| \leq 24$ for graphs where $\Delta(G) = 4$ and $\Delta(G) = 5$ respectively (Section 4.2.5). Finally we give our conclusions and areas for future work in Section 5. The existing results are summarized in Table 1 and our new results are summarized in Tables 2 and 3.

$\Delta(G)$	Number of labels used	Period length
3	3	n
any	$\Delta(G)$	n

Table 3: Our results for Hamiltonian graphs

2 Formal Model

Consider an arbitrary graph $G = (V, E)$ that an agent may explore. Here V is the set of nodes or vertices of the graph and E is the set of edges, that is a set of pairs of vertices.

Define $\delta(v)$ as the set of edges incident to v: $\delta(v) = \{(v, v')\} \subseteq E$.

Let $n = |V|$ be the number of nodes in the graph.

A *port* is defined as a pair $(v, (v, v'))$ of a node v and an edge incident to that node, (v, v')

Define P as the set of all ports $P = \{(u, (u, v)) \mid u \in V, (u, v) \in E\}$.

The degree $d(v)$ of a node is defined as the number of edges incident to that node in G, $d(v) = |\delta(v)|$.

A port numbering is a family of bijective functions $p_v : \delta(v) \to \{0, \ldots, d(v)-1\}$ defined for each vertex v to map from the set of edges incident to v to the subset of the natural numbers $\{0, 1, \ldots d(v) - 1\}$.For brevity we may write $p_v(v')$ instead of $p_v(v, v')$ where v' is adjacent to v.

Define the overall graph port numbering function which maps from all ports in a graph to port numbers, $q : P \to \{0, \ldots, \Delta(G) - 1\}$, formally defined as $q(v, (v, v')) = p_v(v')$.

If an algorithm defines q then the model is said to have *benevolent port numbers* whereas if the algorithm does not define q then the model is said to have *adversarial port numbers.*

The labeling function $c : V \to C$ maps from nodes to labels where C is the set of labels.

If $|C| \geq 2$ then the model is said to have *advice* whereas if $|C| = 1$ then c is a constant function and the model is said to not have advice.

Define the maximum degree of the graph $\Delta(G) = \max_{v \in V} d(v)$ as the largest degree over all vertices.

An agent consists of a set of states S and a partial function f referred to as the agent's *transition function.* The set of states is the set of states from the FSM for the agent. Define D as the set of all port numbers in a graph $\{0, 1, \ldots, \Delta(G) - 1\}$. The transition function takes a state $s \in S$, the input port number $i \in D$, the degree of a node $d(v) \in D$, the label of the node $c(v) \in C$ and returns a state and an outgoing port $j \in D$.

$$f : S \times D \times D \times C \to S \times D$$

The transition function f is a partial function and is not defined for values where $i \geq d(v)$ as there are no such numbered ports. It also does not make sense for the function to return $j \geq d(v)$ as there are no such numbered ports of v. For *oblivious agents* $|S| = 1$ so the transition function may be simplified by removing S.

$$f : D \times D \times C \to D$$

In the case where we do not have advice we have $|C| = 1$ and so f depends only on the input port and the degree of the node.

$$f : D \times D \to D$$

An example of such a transition function is the right-hand rule [7] for which we have transition function

$$f(i, d(v)) = (i + 1) \bmod (d(v))$$

For oblivious agents without advice every transition function that successfully traverses all graphs is equivalent (by Definition 2) to the right-hand rule [5].

We can also consider a transition function as a mapping between ports $g : P \to P$ so g gives the next entry port after the agent enters the given port. Formally we define g as follows:

$$g((v, (v, v')) = (u, (u, v)) \mid f(p_v(v'), d(v), c(v)) = (v, (v, u))$$

We refer to g as the agent's *port mapping function*.

Define a cycle of graph $G = (V, E)$ as a sequence of nodes (possibly with duplicates) $v_0, v_1, \ldots v_{k-1} v_0$ where $(v_i, v_{i+1}) \in E$ and $\{v_0, v_1, \ldots, v_k\} = V$. An agent performs a periodic traversal of the cycle $v_0, v_1, \ldots v_{k-1} v_0$ if it begins by entering a node v_i through port number $p_{v_i}(v_{i-1})$ and the following holds: If the agent enters a node v_i from node v_{i-1}, then its transition function computes the exit port that keeps it on the cycle by going to the next node v_{i+1}. Formally the agent's transition function satisfies the following:

$$f(p_{v_i}(v_{i-1}), d(v_i), c(v_i)) = p_{v_i}(v_{(i+1) \bmod k})$$

Note that the agent will traverse the cycle if it begins by entering a node, say v_i through port number $p_{v_i}(v_{i-1})$, however it will not necessarily traverse the cycle if it initially enters v_i through a differently numbered port. For agents with state we have the additional requirement that they return to their initial state s_0 at the end of the cycle, $f(i, d(v), c(v), s_{k-1}) = (j, s_0)$.

Definition 3. *An* agent-route *is defined for a given agent algorithm with port mapping function g and a start port p_0 as the sequence of ports that the agent A enters starting from p_0 until it enters a port it has previously entered. Formally for any $k \geq 0$ we have a sequence of ports is $p_0 p_1 \dots p_{k-1} p_k$ where $g^i(p_0) = p_i$ and $p_k \in \{p_0, \dots, p_{k-1}\}$.*

Definition 4. *A* traversal *is defined as an agent-route where $p_0 = p_k$ (the agent eventually returns to where it started).*

Definition 5. *We define* agent algorithm equivalence *between two agent algorithms as follows. Agent algorithm A with port mapping function g is equivalent to B with port mapping function g' if and only if $g = g'$.*

Theorem 1. *Two agents algorithms A and B with port mapping functions g and g' respectively are equivalent according to Definition 5 if and only if at every port in a graph then A and B both select the same exit port.*

Proof. Follows from the fact that each port $(u, (u, v))$ is uniquely connected to one other port $(v, (v, u))$. If g and g' map from the same entry ports to exit ports, then these exit ports are uniquely connected to the next entry ports, so $g = g'$. Conversely if g and g' map from the an entry port to the same next entry port, then this entry port is by definition connected to a single port, the exit port. So g and g' select the same exit port for a given entry port.

□

An exploration labeling scheme [3] consists of a pair $(\mathcal{L}, \mathcal{R})$ of a labeling algorithm $\mathcal{L}$ and an agent algorithm R. Given any graph G with any port numbering, the algorithm $\mathcal{L}$ labels the nodes of G (according to c), and the agent $\mathcal{R}$ (governed by transition function f) explores G with the help of the labeling produced by $\mathcal{L}$. We can consider the benevolent ports model as a special case of the adversarial ports model where $\mathcal{L}$ labels each node with a complete mapping from the set of adversarial port numbers to the benevolent port numbers. This pair is fixed for a given class of graphs. It is not permissible to have, for example multiple agent algorithms and the labeling algorithm to choose between them.

Definition 6. *Two exploration labeling schemes,* $(\mathcal{L}, \mathcal{R})$ *and* $(\mathcal{L}', \mathcal{R}')$*, are* equivalent *if and only if their corresponding port mapping functions are equal.*

In general we do not specify the start location of an agent. An agent is said to be able to traverse a graph if and only if there it performs a periodic traversal regardless of its transition function's initial inputs.

3 Related work

Shannon [19] presented the first known algorithm for graph exploration, based on the right-hand rule. Rabin [15] conjectured that no FSM with a finite number of pebbles (markers that may be dropped by the agent at nodes) can explore all graphs. Budach [2] showed that no robot can explore all graphs using zero pebbles. Blum and Kozen [1] showed that no robot can explore all graphs using three pebbles. Kozen [14] later proved that no agent can explore all graphs using 4 pebbles. Finally Rollik [18] proved Rabin's conjecture that no FSM with a finite number of pebbles can explore all graphs.

Without pebbles Fraigniaud and Ilcinkas [8] proved that a robot needs $\Theta(D \log \Delta(G))$ bits of memory where D is the diameter of the graph. However trees may be explored in a period length of only $2n - 2$ by oblivious agents [6, 10].

We now provide a summary of work directly related to the model we described in the previous section, the model with anonymous undirected graphs with port numbers and possibly local advice. Agents are FSMs with small amount of memory or no memory (oblivious agents).

Budach [2] first proved that no oblivious agent can explore all graphs. Rollik [18] extended this result to prove that not even a finite team of agents can explore all graphs or even all planar cubic graphs [18]. Cook and Rackoff [4] considered a similar model called a Jumping Automaton for Graphs (JAG) where a finite team of FSMs collaborate to explore graphs and each FSM is able to teleport to the location of any other FSM. They proved that even JAGs cannot explore all graphs. Fraigniaud *et al.* [8] proved than an FSM requires at least n states to be able to explore all graphs with n vertices. This was shown to be sufficient by Reingold [16] who proved that an n state agent can explore any graph with polynomial period length $O(n)$. Fraigniaud *et al.* [9] later investigated the impact of the agent's memory size on its graph exploration capabilities.

Dobrev *et al.* [7] introduced the idea of the benevolent ports model and provided a port numbering algorithm that allowed oblivious agents to explore all graphs with period length $10n$. Czyzowicz *et al.* [5] provided a new port numbering algorithm that gives a period length of $(4\frac{1}{3})n$ and proved a lower bound on period length of $(2.8)n - O(1)$. Most recently Kosowski and Navarra [13] provided another port numbering algorithm that reduced the period length to $4n - 2$. We make use of this algorithm in our results by using labels to reduce certain cases of the adversarial model to the benevolent ports models which can be solved using the work of Kosowski and Navarra.

Ilcinkas [12] considered the case of a non-oblivious agent with a small amount of memory. Ilcinkas proved an upper bound on period length of $4n - 2$. This bound was improved by Gasieniec *et al.* [10] to $3.75n - 2$. Czyzowicz *et al.* [5] improved this bound further to $3.5n - 2$.

Cohen *et al.* [3] introduced the idea of local advice into the model and proved that $|C| = 3$ is sufficient for a non-oblivious FSM to explore all graphs. Cohen *et al.* also proved that for graphs with degree $\Delta(G) > 4$ certain lower bounds on $|C|$: For loop free graphs (graphs where we never have $(v, v) \in E$) they proved $|C| \geq \lfloor \log \Delta(G) \rfloor - 2$ and for graphs with loops they proved $|C| \geq \frac{\Delta(G)}{2} - 1$. In our results we consider the cases where where $\Delta(G) \leq 4$. We provide an exact bound of $|C| = 2$ for $\Delta(G) = 3$ and provide upper bounds of $|C| \leq 6$ for $\Delta(G) = 4$ and $|C| \leq 24$ for $\Delta(G) = 5$. Our work applies to loop-free graphs.

Zhang *et al.* [11] considered the model with advice using labels where $|C| = 2$. They focused on labeling schemes that not only enabled the agent to explore the graph but also allowed the system designer to adjust the ratio of the number of different labels.

4 Our results

We consider several variants of the graph exploration model that have not previously been investigated. Specifically we provide several agent and labeling algorithms for oblivious agents in the model with advice. We first consider extending the benevolent ports model with advice and then look at the adversarial ports model with advice.

4.1 Benevolent ports model with advice

We first consider the benevolent ports model with advice and prove an upper bound on the number of labels required to perform the shortest possible periodic traversal. The result focuses on reducing the graph to a spanning tree which is then traversed. The algorithm works by constructing a spanning tree for the graph and then performing a right-hand rule traversal on the spanning tree. Labels and port numbers are assigned in a way that provides sufficient information to be able to identify which edges are in the spanning tree and which are not. The agent is then able to traverse the graph by traversing a spanning tree.

Theorem 2. *In the benevolent ports model with advice, a traversal of period length $2n - 2$ may be obtained with $|C| = \Delta(G)$*

Proof. We prove this theorem by presenting a labeling algorithm that describes how port numbers and labels are assigned in a general case graph. We also present an agent algorithm that describes how the agent operates in order to perform a periodic traversal of period length $2n - 2$ given an appropriately labeled graph.

Labeling algorithm The port numbers are assigned as follows. A spanning tree $T = (V' \subseteq V, E' \subseteq E)$ is constructed for the graph $G = (V, E)$. Say there are k spanning tree edges incident to a node. Then the ports of that node that connect to edges in the spanning tree are arbitrarily labeled $0 \ldots k - 1$. The remaining ports are labeled $k \ldots (d(v) - 1)$. That is $p_v(v, v') \in \{0, 1, \ldots, k - 1\}$if $(v, v') \in E'$and $p_v(v, v') \in \{k, \ldots, d(v) - 1\}$ if $(v, v') \notin E'$.

We define the labeling function c as follows. We define $c(v)$ to identify the highest numbered port of v that is in the spanning tree. Formally $c(v) = k$. We require $\Delta(G)$ labels to identify any port number from $0 \ldots \Delta(G) - 1$.

Agent algorithm The agent algorithm is a modification of the right-hand rule. We modify the right-hand rule by limiting it to consider only ports connected to edges in the spanning tree. Formally, the agent's transition function is $f(i, d(v)) = (i + 1) \bmod (c(v))$. Our labeling scheme defines $c(v)$ to signify the highest-numbered port that is part of the spanning tree.

We can partition the set of ports P into the set of ports P_1 connected to edges in the spanning tree, and the remaining ports $P_2 = P \backslash P_1$. Now consider the agent's port mapping function g and the port mapping function for the right-hand rule on the spanning tree g'. If we restrict the domain of g to P_1 then by definition of g, we have that g is equivalent by Definition 5 g'. Hence for all ports in P_1 the agent's transition function is equivalent to performing right-hand rule on the spanning tree. For all other ports P_2 , by definition of g we have $g(p \in P_2) \in P_1$. So if the agent starts at a non-tree port, then it will exit on a tree port and converge.

Agent traversal The graph from the agent's perspective has now been reduced to the spanning tree. It is known that an agent performing right-hand rule on a tree will perform a periodic traversal [19]. Hence the agent will perform a periodic traversal of the graph exactly following the spanning tree of the graph giving a traversal period length of $2n-2$. $\square$

This is a significant improvement over the oblivious benevolent ports agent which has worst case period length $4n-2$ [13]. The cost involved is $|C| = \Delta(G)$.

4.1.1 Hamiltonian graph

The previous section focused on reducing an arbitrary graph to a tree which can be easily traversed by a right-hand rule agent. An alternative to reducing to a tree is to reduce to a Hamiltonian cycle for Hamiltonian graphs. The agent is then able to follow the Hamiltonian cycle in order to perform a periodic traversal of the graph. Note that this algorithm applies to almost all regular graphs, since almost all regular graphs are Hamiltonian [17] (as $n \to \infty$ the proportion of Hamiltonian n node regular graphs goes to 1).

Theorem 3. *For the benevolent ports model without advice, for any graph containing a Hamiltonian cycle, a periodic traversal exists with a period length of n.*

Proof. We prove this by providing a port numbering algorithm that given a graph and a Hamiltonian cycle within the graph, numbers the ports such that the agent is able to perform a periodic traversal coinciding with the Hamiltonian cycle, which has period length n.

Port numbering

Let $v_0, v_1, ...v_{n-1}, v_0$ be the Hamiltonian cycle. Now for each vertex define the port numbering function as follows. We label the next edge in the Hamiltonian cycle with 0 and all other edges arbitrarily. Formally, for all $0 \leq i < (n-1)$ we have $p_{v_i}(v_i, v_{i+1}) = 0$ and $p_{v_i}(v_i, v_{j \neq i+1}) \in \{1, \ldots d(v)-1)\}$. For $i = (n-1)$ define $p_{v_i}(v_i, v_0) = 0$ and $p_{v_i}(v_i, v_{j \neq 0}) \in \{1, \ldots d(v)-1)\}$.

The port labeled 0 identifies the edge connected to the next vertex on the Hamiltonian cycle.

Agent algorithm

The agent transition function is defined as the constant function $f(i, d(v)) = 0$. The agent now exactly follows the Hamiltonian cycle, completing a periodic traversal of the graph exactly corresponding to the Hamiltonian cycle with period length n.

□

4.2 Adversarial ports model with advice

We now consider the adversarial ports model, that is where each p_v is defined by an adversary. We consider the model with advice, that is $|C| > 1$. A common idea when attempting to find periodic traversals is to reduce or decompose the graph to a simpler form such as a tree or a Hamiltonian cycle which can be traversed more easily.

4.2.1 Hamiltonian graph: exact bound

In the case where the graph contains a Hamiltonian cycle, it is possible to label the graph such that the agent performs a periodic traversal following a Hamiltonian cycle.

Theorem 4. *For a Hamiltonian graph in the adversarial model there exists labeling and agent algorithms with $|C| = \Delta(G)$ that perform a periodic traversal with $\pi = n$.*

Proof. Consider some Hamiltonian cycle in the graph $v_0, v_1, \ldots v_{n-1}, v_0$. The labeling function identifies the port that connects to the node next in the Hamiltonian cycle. Define the labeling function as $c(v_i) = p_{v_i}^{-1}(v_i, v_{i+1})$ for $0 \leq i < (n-1)$ and $c(v_{n-1}) = p_{v_{n-1}}^{-1}(v_{n-1}, v_0)$. We have $\forall v.p_v(v, v') \in \{0, 1, \ldots, \Delta(G)\}$ and hence $\forall v.c(v) \in \{0, 1, \ldots, \Delta(G)\}$ and $|C| = \Delta(G)$.

Define the agent transition function as $f(i, d(v)) = c(v)$. Now the agent will perform a periodic cycle exactly coinciding with the Hamiltonian cycle which has period length n.

□

4.2.2 Arbitrary graph: upper bound

In this section we look at reducing arbitrary graphs to their spanning tree and performing a traversal of this, hence giving us an upper bound on $|C|$ required to perform a periodic traversal.

Theorem 5. *For an arbitrary graph in the adversarial model and* $|C| = 2^{\Delta(G)} - 1$ *there exists a periodic traversal with* $\pi = 2n - 2$.

Proof. For a general case graph with adversarial ports, we can use the spanning tree technique that we used for non-adversarial ports. That is, we first find a spanning tree T in the graph. Define $d_T(v)$ as the degree of a vertex v in the spanning tree T. We then add advice that identifies whether each port is connected to an edge in the spanning tree or not.

Labeling algorithm There are $2^{\Delta(G)} - 1$ possible combinations of ports, this is because if we could have every combination of edges connected to a maximum degree node, there would be $2^{\Delta(G)}$ (there are this many subsets of the set of all ports). However each edge must be connected by at lease one edge to the tree so we can exclude one combination that correspond to no edges (the empty subset), giving $2^{\Delta(G)} - 1$. Hence we require $|C| = 2^{\Delta(G)} - 1$ labels to be able to identify each possible combination of which edges are in the tree and which are not. So we define the labeling function as a set of port numbers connected to tree edges: $c(v) = \{a_0, a_1, \ldots, a_{d_T(v)}\}$ where $p(v, v') = a_i$ and $a_i \in c(v)$ if and only if $(v, v') \in T$.

Agent algorithm The agent performs a right-hand rule traversal restricted to the spanning tree. The transition function is $f(i, d(v), c(v)) = i'$where $i' = i'' \bmod (d(v))$ where i'' is the least integer greater than i such that $i'' \in c(v)$.

Agent traversal The agent now performs right hand rule on the spanning tree as all non-tree edges are identified by c and skipped by the agent. It has been proven that an agent operating under the right hand rule performs a traversal of a tree [19, 6, 3, 7] so in traversing the graph in this way, the agent will perform a periodic traversal of the tree and thus have traversal length of $2n - 2$. □

4.2.3 Degree three bounded graphs: 5-label algorithm

In Section 4.2.4 we show that it is possible to traverse all graphs where $\Delta(G) = 3$ with only $|C| = 2$ with $\pi = 4n-1$. However it may be the case that we want to use more labels to decrease the period length. We now present an algorithm based on a tree decomposition of graphs where $\Delta(G) = 3$ using $|C| = 5$. We improve $\pi = 4n-1$ to $\pi = (\frac{3}{2})n-1$. We do presenting an agent

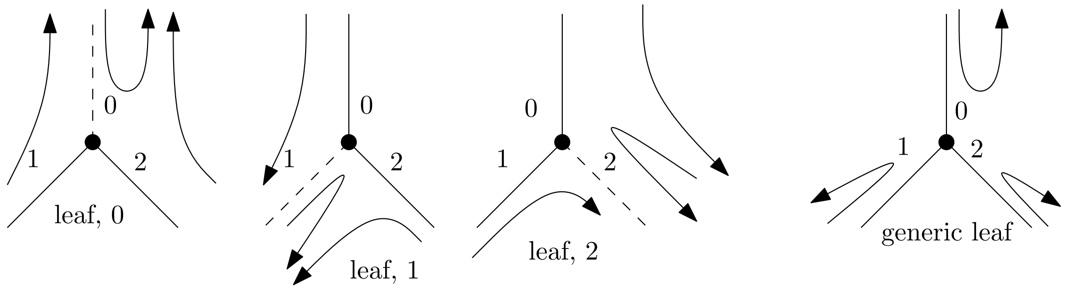

Figure 4.1: Leaf labeling in 7-label algorithm (left) and 5-label algorithm (right)

algorithm and labeling scheme based on tree decomposition that explores all degree three bounded graphs using 5 labels.

Theorem 6. *For graphs with $\Delta(G) = 3$, there exists an agent with transition function f, and a labeling function c with $|C| = 5$ such that the agent executes a periodic traversal of length $\pi = \frac{3}{2}n - 1$ starting from any initial edge.*

Proof. We know that we can construct a spanning tree and identify it using 7 labels (using the algorithm from Section 4.2.2). An agent is said to *bounce* off a node, say v_1, if it moves from one node v_0 to another node v_1 and then back to v_0 in 2 steps. Formally $f(i, d(v_0), c(v_0)) = p_{v_0}(v_1)$ and $f(p_{v_1}(v_0), d(v_1), c(v_0)) = p_{v_1}(v_0)$.

In order to only use 5 labels instead of 7, the agent will simply bounce off nodes labeled as leaves. Figure 4.1 shows that the 7-label algorithm uses 3 labels to label leaves, however we could instead label them all the same and reduce the total number of labels from 7 to 5.

However the agent would get stuck if it started between two leaves (just bouncing back and forth between them) so in this case we label one of the leaves as if it were a degree 2 node so that the agent would leave the tree, bounce off the other leaf, and then re-enter the tree. This is shown in Figure 4.2.

From Figure 4.2 we can see that this means that in comparison to the 7-label tree, we have to do at most one extra edge traversal for every two leaves.

Formally, we define our labeling function c as follows.

Labeling algorithm Construct a spanning tree $T \subseteq G = (V_T, E_T)$.

Define the set of edges incident to a node in the tree, $\delta_T(v) = \{(v, v') \mid (v, v') \in E_T\}$. Define the degree of the node in the tree $d_T(v) = |\delta_T(v)|$.

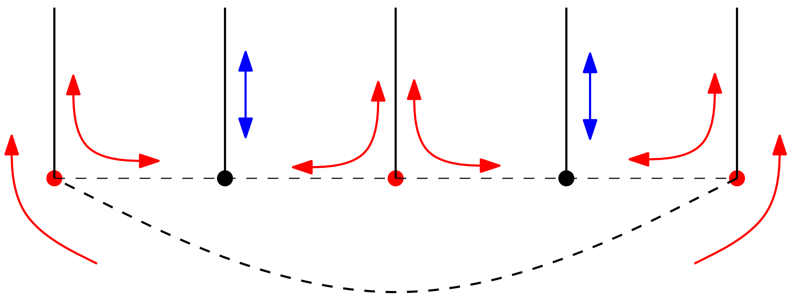

Figure 4.2: A set of leaves connected in a cycle. Solid lines are tree edges. Arrows show how the agent behaves at each vertex for the 5-label algorithm.

Define $g_T(v) = \{p_v(v, v') \mid (v, v') \in T\}$ as the set of port numbers of a node that are incident to edges in tree T.

We will be labeling nodes first by their degree in the tree, then with extra information for nodes that have degree two in the tree. Labels will be of the form $c(v) \in \{(1, -), (2, \{u, v\}), (3, -)\}$. The left part of the pair encodes the degree of the node in the tree and the right part of the pair encodes any extra information required to traverse that node. While these labels do not follow the $\{0, 1, \ldots, |C| - 1\}$ convention of our model, it is straightforward to encode them by mapping each label to a number $\{0, 1, \ldots, 4\}$.

We label all nodes with degree 3 in the tree with label $(3, -)$: Let $c(v) = (3, -)$ if and only if $d_T(v) = 3$.

Now label all nodes with degree 2 in the tree as specified below. This identifies which two of their three edges are in the tree:

$$c(v) = \begin{cases} (2, \{0, 1\}) & \text{if } d_T(v) = 2\ \& g_T(v) = \{0, 1\} \\ (2, \{0, 2\}) & \text{if } d_T(v) = 2\ \&\ g_T(v) = \{0, 2\} \\ (2, \{1, 2\}) & \text{if } d_T(v) = 2\ \&\ g_T(v) = \{1, 2\} \end{cases}$$

Define the graph containing only nodes that are leaves in the tree and only edges that are non-tree edges: Let $T' = (G \backslash T) \backslash \{v \mid d_T(v) \in \{2, 3\}\}$. This graph T' is a is composed of of paths and cycles. This is because every node has at least one tree edge incident to it, so has at most degree two in T'. Every node in T' satisfies $d_T(v) = 1$ (is a leaf in tree T by definition of T'). For each path pick a node v_0 at the end of the path and then we have $v_0 v_1 \ldots v_{k-1}$. For cycles arbitrarily pick a node v_0 as the endpoint and an orientation for the cycle and then we have $v_0 v_1 \ldots v_{k-1} v_0$. Now starting at v_1 label every alternate v_i with 0: For each line or cycle, define $c(v_i) = (1, -)$ if and only if $i \bmod 2 = 1$.

Now consider all nodes in the line apart from v_{k-1}. We will label the remaining unlabeled nodes in $v_0 v_1 \dots v_{k-2}$ in a similar way. Specifically, for all $v_0 v_1 \dots v_{k-2}$, if $i \bmod 2 = 0$ then define:

$$c(v_i) = \begin{cases} (2, \{0,1\}) & \text{if } g_T(v_i) \cup \{p_{v_i}(v_i, v_{i+1})\} = \{0,1\} \\ (2, \{0,2\}) & \text{if } g_T(v_i) \cup \{p_{v_i}(v_i, v_{i+1})\} = \{0,2\} \\ (2, \{1,2\}) & \text{if } g_T(v_i) \cup \{p_{v_i}(v_i, v_{i+1})\} = \{1,2\} \end{cases}$$

Finally consider node v_{k-1}. If $(k-1) \bmod 2 = 1$ then we have already have label $c(v_{k-1}) = (1, -)$. If $(k-1) \bmod 2 = 0$ then label it as follows:

$$c(v_i) = \begin{cases} (2, \{0,1\}) & \text{if } g_T(v_i) \cup \{p_{v_i}(v_i, v_{i-1})\} = \{0,1\} \\ (2, \{0,2\}) & \text{if } g_T(v_i) \cup \{p_{v_i}(v_i, v_{i-1})\} = \{0,2\} \\ (2, \{1,2\}) & \text{if } g_T(v_i) \cup \{p_{v_i}(v_i, v_{i-1})\} = \{1,2\} \end{cases}$$

We have now defined $c(v)$ for all vertices of graph G.

Agent algorithm The agent transition function defined for nodes where $c(v) \neq (1, -)$ as performing right-hand rule restricted to the edges denoted by label $c(v)$. Formally, for $c(v) = (3, -)$ we perform right-hand rule$f(i, d(v), (3, -)) = (i+1) \bmod d(v)$. For $c(v) = (2, \{u, v\})$ we perform right hand rule restricted to the edges denoted by the label $\{u, v$.

The transition rule is $f(i, d(v), (2, \{0,1\})) = j \bmod d(v)$ where j is the least integer greater than i such that $j \bmod d(v) \in \{0,1\}$. Similarly we have, $f(i, d(v), (2, \{0,2\})) = j \bmod d(v)$ where j is the least integer greater than i such that $j \bmod d(v) \in \{0,2\}$. Similarly again we have $f(i, d(v), (2, \{1,2\})) = j \bmod d(v)$ where j is the least integer greater than i such that $j \bmod d(v) \in \{1,2\}$.

For nodes where $c(v) = (1, -)$ the agent simply bounces back out of the port it came in from, $f(i, d(v), (1, -)) = i$.

The agent's transition function f is shown by Table 4. The input port is i and the output port j. The $d(v)$ is omitted as f does not depend on $d(v)$.

Agent traversal Consider the periodic traversal $v_0 v_1 \dots v_{k-1} v_0$ obtained by an agent performing right hand rule on spanning tree T. Now for all nodes where $d_T(v) \in \{2,3\}$ the agent performs the right-hand rule restricted to spanning tree T, consistent with the periodic traversal. If the agent starts on a non-tree edge and hits a node where $d_T(v) = 2$ then it exits on a tree edge and converges to the traversal. However for nodes where $d_T(v) = 1$ we have one of two cases, either $c(v) = 0$ or $c(v) \in \{\{0,1\}, \{0,2\}, \{1,2\}\}$.

i	$c(v)$	j
	$(1, -)$	0
	$(2, \{0, 1\})$	1
0	$(2, \{0, 2\})$	2
	$(2, \{1, 2\})$	1
	$(3, -)$	1
	$(1, -)$	1
	$(2, \{0, 1\})$	0
1	$(2, \{0, 2\})$	2
	$(2, \{1, 2\})$	2
	$(3, -)$	2
	$(1, -)$	2
	$(2, \{0, 1\})$	0
2	$(2, \{0, 2\})$	0
	$(2, \{1, 2\})$	1
	$(3, -)$	0

Table 4: Degree 3 bounded graph agent transition function

If the agent starts on a tree edge and hits a node where $d_T(v) = 1$ and $c(v) \in \{\{0, 1\}, \{0, 2\}, \{1, 2\}\}$ then it will exit on a non-tree edge. However by definition this edge is connected to a node v' where $c(v') = 0$. Hence the agent will then bounce back to node v and then re-enter the tree where it left. This corresponds to extending the periodic cycle by two nodes, $v'v$. For example say we enter v_1 and then leave the tree, we are still performing a periodic traversal $v_0 v_1 v' v_1 \dots v_{k-1} v_0$ albeit two nodes longer than the tree traversal. If the agent starts on a non-tree edge and hits a node where $d_T(v) = 1$ and $c(v) \in \{\{0, 1\}, \{0, 2\}, \{1, 2\}\}$ then by definition it will exit on an edge in this new traversal and converge.

Figure 4 shows how the agent behaves if it encounters several leaves that are connected together. Figure 4.3 shows how an agent may encounter a leaf connected to another leaf, exit the spanning tree and then bounce back and re-enter the spanning tree.

For nodes where $c(v) = 0$, if the agent starts on an edge in this new traversal, it will bounce off the node and remain on the traversal. If it starts on a non-traversal edge and hits a node where $c(v) = 0$, it will bounce back and by definition of $c(v)$ enter a node where $c(v) \in \{\{0, 1\}, \{0, 2\}, \{1, 2\}\}$ and hence exit on a traversal edge. So the agent always converges to a periodic

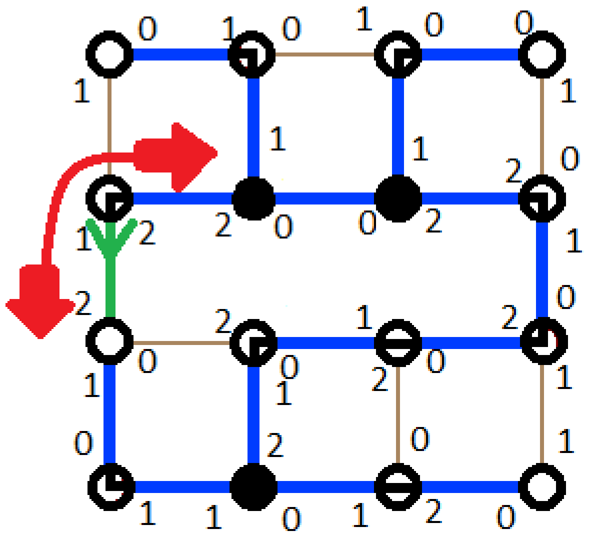

Figure 4.3: A spanning tree extended with an edge that can only be entered by the node at one end. White nodes satisfy $c(v) = (1, -)$, black nodes satisfy $c(v) = (3, -)$ and striped nodes satisfy $c(v) = (2, \{u, v\})$ where the stripes mark the edges u and v.

traversal.

The periodic traversal is the tree traversal, extended by two nodes for at most half the leaves, which works out to at most one node per leaf. There are at most $\frac{n}{2}+1$ leaves in the spanning tree as we can pick a leaf node to be the root, and then excluding the root we have a binary tree with $n-1$ nodes and hence at most $\frac{(n-1)+1}{2}$ leaves. Adding the root node back (as it is a leaf) gives at most $\frac{n}{2}+1$ leaves overall. The traversal of the spanning tree has length $2n-2$ and we traverse the spanning tree and at most one extra edge for every leaf giving an overall period length of at most $2n-2+\frac{n}{2}+1=(2\frac{1}{2})n-1$. □

4.2.4 Degree three bounded graphs: exact bound on number of labels

In the previous section we presented a 5-label algorithm with $(2\frac{1}{2})n-1$. In this section we present an exact bound of $|C|=2$ required to explore all degree three bounded graphs when the period length is unrestricted. We prove this exact bound by reducing the problem to the benevolent ports case.

Our technique results in a longer period length than the $|C|=5$ algorithm. Since we reduce to the benevolent ports case, the period length is dependent on existing benevolent ports algorithms. The current upper bound for this is $4n-2$ [13] and even the lower bound of $2.8n$ [5] is greater than the worst case period length of our $|C|=5$ algorithm which we previously described in Section 4.2.3. It has been proven that it is impossible to explore all cubic planar graphs, and hence all degree three bounded graphs with adversarial port numbers when $|C|=1$ [18]. In this section we show that by increasing $|C|=1$ to $|C|=2$ it is possible to explore not only all cubic planar graphs, but indeed all degree three bounded graphs.

We show that there are only two agent algorithms that could explore all possible degree three bounded graphs.

We subsequently provide a labeling algorithm that allows one of these agent algorithms to explore all such graphs.

Theorem 7. *In the adversarial ports model with advice, with $|C|=2$, for graphs where $\Delta(G)=3$, there are exactly two agent algorithms that can explore all such graphs.*

Proof. We prove this by giving a graph that cannot be explored by all but two of the possible agent algorithms. Consider a subset of degree three bounded graphs where $\forall v.d(v)=3$, referred to as *cubic graphs.* There are 6 independent situations in which we can specify the behavior of the agent:

It may enter a node with $c(v) = 0$ or $c(v) = 1$ and will enter through port $i \in 0, 1, 2$. Hence there are 6 different scenarios where the agent has to decide which port to exit out of. The exit port is chosen from three different ones $\{0, 1, 2\}$. Hence there are at most $3^6 = 729$ agent algorithms. We now demonstrate how certain structures of graph restrict these choices down two algorithms.

Consider the pentagon graph in Figure 4.4. This is a graph structure that may be connected on to another graph by the edge at the top. Every node in the pentagon has degree three. The agent must enter from the top. The agent enters through port 0 and if the agent can only exit through port 1 when it enters through 0, it will traverse the marked loop of 4 nodes, missing the circled node. For an agent to explore this structure it must have $f(0, b) = 2$ for some b (degree omitted as we are considering cubic graphs where for all nodes $d(v) = 3$) for some $c(v)$.

Similarly, at each node in the pentagon we could renumber all ports currently numbered 1 with 2 and vice versa. This new permutation of port numbers then gives us $f(0, a) = 1$ for some a. So where $a, b \in \{0, 1\},\ a \neq b$ we have:

$$\begin{aligned} f(0, a) &= 1 \\ f(0, b) &= 2 \end{aligned}$$

Similarly we could have another 4 pentagons, each with a different permutation of the port numberings, in order to constrain the agent's transition function as shown below, where $c, d, e, f \in \{0, 1\},\ c \neq d,\ e \neq f$:

$$\begin{aligned} f(1, c) &= 0 \\ f(1, d) &= 2 \\ f(2, e) &= 0 \\ f(2, f) &= 1 \end{aligned}$$

These constraints mean that any agent that can execute a periodic traversal in all cubic graphs can never bounce back off a node with $f(i, b) = i$ for any b. If this is part of the transition function then the agent cannot explore all 6 pentagons.

Now we put these 6 pentagons together into a single graph, shown in Figure 4.7. Each of the small pentagons represents a different numbering of the pentagon in Figure 4.4. Now say the agent starts on the edge marked

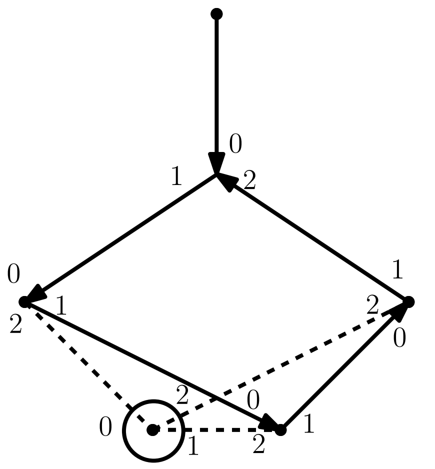

Figure 4.4: Pentagon structure. When the agent enters the pentagon from the top, if it never enters 0 and exits 2, then the agent never traverses the circled node (or the dashed edges).

$f(0,0)=1$	$f(0,1)=2$
$f(1,0)=2$	$f(1,1)=0$
$f(2,0)=0$	$f(2,1)=1$

Figure 4.5: *Right-hand rule/left-hand rule* agent algorithm

by the arrowhead, going towards the right. When the agent enters the first node through port 0, it must do one of two things.

It either exits through port 1, traverses the connected pentagon, then re-enters the node through port 1 and exit through port 2. When the agent comes back it must enter through port 2 and exit through port 0 in order to traverse the whole graph. This restricts f as shown in Figure 4.5 (without loss of generality we consider the node as labeled $c(v) = 0$).

Or alternatively when the agent enters the first node through port 0 it exits through port 2, and when it comes back it enters through port 2, exits through port 1, traverses the pentagon, then enters through 1 and exits through 0. This restricts f as follows:

We refer to the algorithm in Figure 4.5 as the *right-hand rule/left-hand rule* algorithm as for nodes where $c(v) = 0$ it performs right-hand rule, and for all other nodes where $c(v) = 1$ it does the opposite which we refer to

$f(0,0) = 2$	$f(0,1) = 1$
$f(1,0) = 0$	$f(1,1) = 2$
$f(2,0) = 1$	$f(2,1) = 0$

Figure 4.6: *Left-hand rule/right-hand rule* agent algorithm

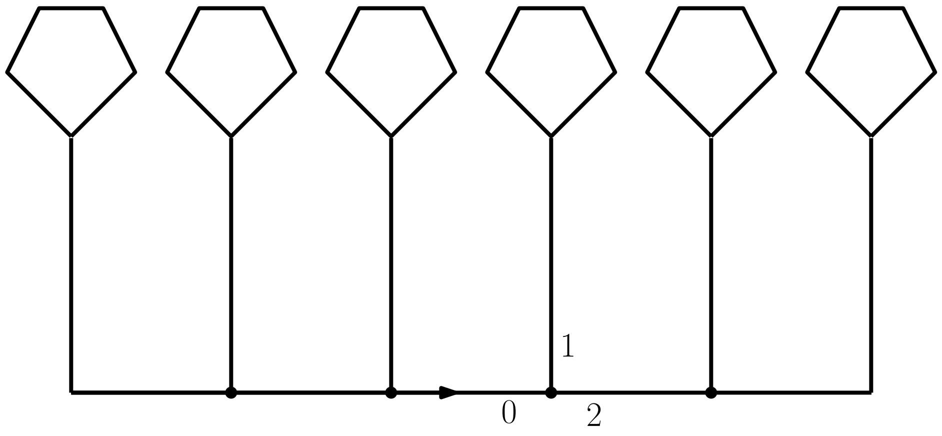

Figure 4.7: 6 pentagon graph

as left-hand rule. Similarly we refer to the algorithm in Figure 4.6 as the *left-hand rule/right-hand rule* algorithm. Say we have algorithm $\mathcal{R}$ performing right-hand rule/left-hand rule and algorithm $\mathcal{R}'$ performing left-hand rule/right-hand rule. Now for any labeling algorithm $\mathcal{L}$ defined by labeling function $c(v)$, we can construct $\mathcal{L}'$ defined by $c'(v)$ where $c'(v) = 1$ if and only if $c(v) = 0$, otherwise $c'(v) = 0$. Now the two labeling schemes $(\mathcal{L}, \mathcal{R})$ and $(\mathcal{L}', \mathcal{R}')$ are equivalent (by Definition 6) because for any input port p, if $\mathcal{L}$ makes the first algorithm choose output port based on the right-hand rule, then $\mathcal{L}'$ will also make $\mathcal{R}'$ choose the output port based on the right-hand rule. Similarly for the case where $\mathcal{L}$ makes $\mathcal{R}$ select the output port based on the left-hand rule.

So no other algorithm can possibly explore the 6 pentagon graph (Figure 4.7).

□

Note that while we have proven that no algorithm that is not equivalent by defintion 6 to right-hand rule/left-hand rule (Figure 4.5) can explore all degree three bounded graphs, we have not yet shown that this algorithm can explore all degree three bounded graphs. It turns out that right-hand rule left-hand rule can explore all degree three bounded graphs.

We will next show that the case of the adversarial ports model with advice can be reduced to that of benevolent ports model without advice where it was proven by Dobrev *et al.* [7] that it is possible to assign the port numbers by defining p_v such than an agent can traverse all graphs. The initial period length $10n$ was subsequently improved to $(4\frac{1}{3})n$ by Czyzowicz *et al.* [5] and more recently this result was improved by Kosowski and Navarra [13] to $4n-2$. The technique of Kosowski and Navarra is based on a graph decomposition based on a spanning tree of the graph. The decomposition then has its ports numbered in such a way to allow a benevolent agent to perform a periodic traversal.

We now show how we can use such benevolent port numbering algorithms to solve the case of the adversarial ports model with advice. We first give a discussion of the algorithm, before providing a formal proof in Theorem 8.

The idea is to reduce the problem of traversing a graph in adversarial ports with advice to traversing with benevolent ports without advice. We show that right-hand rule/left-hand rule can simulate any agent algorithm in the benevolent ports model without advice.

In the benevolent ports model without advice, a labeling algorithm supplies the port numbers to each node, so that a right-hand rule agent performs a periodic traversal. For degree three nodes, there are 6 different ways of assigning port numbers, shown in Figure 4.8. The arrow in Figure 4.8 shows how right-hand rule behaves on each of the different port numbering, for example for the first node the right-hand rule enters through 0 and exits 1, enters 1 and then exits 2, and enters 2 and then exits 0.

From Figure 4.8 we can see that of the 6 different port number assignments, the right hand rule only behaves in one of two different ways, traversing the node in a clockwise or counter-clockwise manner. These directions correspond to the parity of port number permutations. described in the paragraph below. A diagram for the left-hand rule would be the same except the direction of all the arrows is reversed.

So when benevolent port numbering algorithms assign benevolent port numbers to cubic graphs the benevolent port algorithm is choosing between one of two behaviors for the agent at that node. For a node pick an ordering of its ports, say p_1, p_2, p_3. The set of possible port numberings is $\{012, 021, 120, 102, 201, 210\}$. We partition this set into two equivalence classes: the set of even permutations $\{012, 120, 201\}$ and the set of odd permutations $\{021, 102, 210\}$. Now if we pick a port numbering from the set of even permutations we get one right hand rule behavior, and if we pick an odd permutation then we get the other behavior.

Now consider the case with adversarial port numbers for a graph G. If we first take G (ignoring port numbers) and run a benevolent ports algorithm, we obtain definitions for each port numbering function, say p'_v. For each node $v \in V$ pick an ordering of its ports $p_{v0}p_{v1}p_{v2}$. In the adversarial case we already have each p_v defined by an adversary. For each node v, then p_v and p'_v either number $p_0p_1p_2$ with port numbers in the same equivalence class (both assign even permutations or both assign odd permutations), or p_v and p'_v assign port numbers from different equivalence classes (one assigns an odd permutation and the other assigns an even permutation). For all nodes where p_v and p'_v assign numbers from the same equivalence class, define $c(v) = 0$. In this case right-hand/left-hand rule behaves equivalently to right-hand rule because Theorem 1 states that for two agent algorithms to behave equivalently, they must select the same exit port for every entry port. Where $c(v) = 0$ then right-hand/left-hand rule picks the output port based on the right-hand rule, so is equivalent to the right-hand rule algorithm. Since in this case (where $c(v) = 0$) right-hand/left-hand rule is equivalent to right hand rule, this is consistent with a benevolent port numbering traversal.

In the remaining nodes where p_v and p'_v assign port numbers from different equivalence classes, define $c(v) = 1$. Say that the benevolent port numbering p'_v picks an even permutation, but the adversarial numbering p_v uses an odd permutation. Then this means that if we perform left-hand rule on the node, then we will traverse the ports in the order $p_3p_2p_1$ which will be an even permutation (due to the fact that for strings of length three, reversing an odd permutation gives an even permutation). Labeling the node $c(v) = 1$ makes right-hand rule left-hand rule perform left-hand rule on the node and traverse the node consistently with the periodic traversal of a benevolent port numbering algorithm.

Hence at every node in the adversarial case the right-hand rule/left-hand rule agent behaves identically to the right-hand rule in the case of benevolent port numbers. Hence by Theorem 1 the right-hand rule/left-hand rule performs a periodic traversal identical to that of a benevolent port numbering algorithm.

The algorithm is easily extended from cubic graphs to degree 3 bounded graphs as for each node where $d(v) = 1$ there is no choice but to exit from the entry port, and for nodes where $d(v) = 2$ we just exit from the port we did not enter from, consistent with the right-hand (and indeed left-hand) rules.

The right-hand rule/left-hand rule algorithm can traverse all graphs where

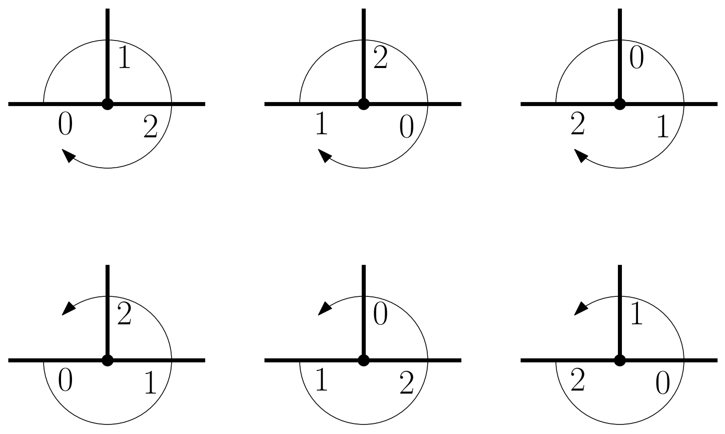

Figure 4.8: Port numberings for degree 3 nodes

$\Delta(G) = 3$ in period length $4n-2$ as using the above algorithm we know that the right-hand rule/left-hand rule algorithm can traverse all graphs where $\Delta(G) = 3$ in a path identical to that obtained using a benevolent port numbering algorithm. The benevolent port numbering algorithm of Kosowski and Navarra [13] gives a period length of $4n-2$, hence using this with the above theorem gives a period length of $4n-2$ for the right-hand rule/left-hand rule algorithm.

The $4n-2$ bound is not known to be optimal, so it is possible that this result could improve with better benevolent ports algorithms, since the best known lower bound on benevolent ports algorithms' period length is $(2.8)n$.

4.2.5 Higher degree graphs: upper bounds on number of labels

We can use the technique of the previous section to create algorithms to explore higher degree graphs. While this technique is applicable to graphs of any $\Delta(G)$, we show below that this technique only gives an improvement over the algorithm from Section 4.2.2 for small $\Delta(G)$.

Theorem 8. *A graph G with adversarial port numbers defined by port numbering functions $q_v : \delta(v) \to \{0, \ldots, d(v)-1\}$ and overall port numbering function $q(v,(v,v')) = q_v(v')$ can be explored by an oblivious agent using at most $|C| = (\Delta(G)-1)!$ labels.*

Proof. A benevolent port numbering algorithm A takes a graph G and returns a port numbering function, $A(G) = r$ where $r : P \to \{0, \ldots, \Delta(G)-1\}$.

Say A defines r to allow a right-hand rule agent to explore G. So A defines for each node v a function $r_v : \delta(v) \to \{0, \ldots, d(v)-1\}$ and $r(v, (v, v')) = r_v(v')$.

A right-hand rule agent R has transition function $f(i, d(v)) = (i+1) \bmod d(v)$ and so port mapping function

$$g((u, (u, v)) = (v'(v', u))$$

where:

$$\begin{aligned} f(r_u(v), d(u)) &= x \\ r_u^{-1}(x) &= (u, (u, v')) \end{aligned}$$

Eliminating x for entry port $(u, (u, v))$ gives exit port:

$$(u, (u, v')) = r_u^{-1}(f(r_u(v), d(u))$$

Expanding f gives $(u, (u, v')) = r_u^{-1}((r_u(v) + 1) \bmod d(u))$.

Define a port numbering function s as a *rotation* of r if and only if $s_v(p) = r_v(p) + i \bmod d(v)$ where $i \in \{0, 1, 2, \ldots\}$ and $s(v, (v, v')) = s_v(v')$.

We now show that, when s is a rotation of r, given an input port $(u, (u, v))$ then the exit port $(u, (u, v'))$ chosen by right hand rule f is the same for s and r:

Using s we have:

$$(u, (u, v')) = s_u^{-1}(f(s_u(v), d(u))$$

Expanding f gives:

$$(u, (u, v')) = s_u^{-1}((s_u(v) + 1) \bmod d(u))$$

Expand s_u in terms of r_u:

$$(u, (u, v')) = s_u^{-1}((r_u(v) + i + 1) \bmod d(u))$$

Expand s_u^{-1} in terms of r_u^{-1}:

$$\begin{aligned} (u, (u, v')) &= r_u^{-1}((r_u(v) + i + 1 - i) \bmod d(u)) \\ &= r_u^{-1}((r_u(v) + 1) \bmod d(u)) \\ &= r_u^{-1}((f(r_u(v), d(u))) = s_u^{-1}((f(s_u(v), d(u))) \end{aligned}$$

So if we use a rotation s of a benevolent port numbering r then the right-hand rule still pick the same output ports for a given input port. Theorem 1 states that if two algorithms pick the same output port for a given input port then they are equivalent. Hence right-hand rule using s is equivalent to right-hand rule using r.

For a node v, a benevolent port algorithm defines a port numbering function $r_v : \delta(v) \rightarrow \{0, \dots, d(v) - 1\}$. Function r_v must by definition be bijective and so there are at most $d(v)!$ possible functions from which r_v is chosen. We have shown that for a right-hand rule agent then any rotation s_v results in identical agent behavior. For any r_v there are d possible rotations of r_v.

Define an equivalence relation where r_v is equivalent to s_v if and only if s_v is a rotation of r_v. We have shown that for any two elements of the same equivalence class, a right-hand rule agent will behave identically. Since there are d rotations for any r_v and r_v is chosen from a set of size $d(v)!$ then there are $d(v)!/d = (d(v) - 1)!$ equivalence classes. A node of maximum degree will hence have $(\Delta(G) - 1)!$ equivalence classes.

We now return to the original problem of how to label graph G. We do this as follows:

For each node v of degree $\Delta(G)$ we have $(\Delta(G) - 1)!$ equivalence classes, say $E_0 E_1 \dots E_k$. A benevolent port algorithm A will define for each v a port numbering function r_v. We now label v using one of the equivalence classes, $c(v) = E_i$ where $r_v \in E_i$.

Now define the agent transition function to perform the right-hand rule using an element of E_i (where p_v is the adversarial port numbering function):

$$\begin{aligned} f(i, d(v), c(v)) &= p_v(s_v^{-1}((s_v(p_v^{-1}(i)) + 1) \bmod d(v))) \\ s_v &\in c(v) \end{aligned}$$

It does not matter which $s_v \in c(v)$ we pick as we have shown that they result in identical behavior for a right-hand rule agent.

Using this labeling scheme which requires $|C| = (\Delta(G) - 1)!$ labels, the agent will traverse the graph identically to a right-hand rule agent traversing G using benevolent port numbers.

□

Note that while this technique can be used for graphs of any $\Delta(G)$, there is only an improvement in $|C|$ for small $\Delta(G)$. This is because the number of labels used by this technique grows as $O(\Delta(G)!)$, whereas the tree-based algorithm of Section 4.2.2 grows at rate $O(2^{\Delta(G)})$. Hence for large $\Delta(G)$ the tree-based algorithm is more efficient, however for $\Delta(G) < 6$ we get an improvement in $|C|$. For $\Delta(G) = 4$ we get $|C| = 6$ and for $\Delta(G) = 5$ we get $|C| = 24$ (as opposed to $|C| = 15$ and $|C| = 31$ respectively for the tree-based algorithm).

5 Conclusions

We have shown a new exact bound of $|C| = 2$ for the number of labels needed for an oblivious agent to explore degree three bounded graphs with adversarial port numbers and local advice at each node.

We have shown that there is in fact only one such agent algorithm (up to trivial isomorphism of swapping label values) that can explore all such graphs. This algorithm worked by reducing the case of adversarial ports with advice to benevolent ports without advice. The upper bound on period length is $4n - 2$ when reducing to previous results of Kosowski and Navarra [13]. It may be possible to improve on this period length in future work.

We have also shown a new 5-label algorithm that explores all degree three bounded graphs with similar conditions, with period length $(2\frac{1}{2})n - 1$. It remains open as to whether this technique can be extended to graphs of higher degree.

We have proved an upper bound of $|C| = (\Delta(G) - 1)!$ for all graphs in the adversarial model with advice. This gives new upper bounds for graphs where $|C| \in 4, 5$. This technique gives a period length of $4n - 2$ as it reduces to the previous results of Kosowski and Navarra [13]. It remains open as to whether these upper bounds are tight or not.

6 Acknowledgements

The author would like to thank James Aspnes, Dana Angluin and Joan Feigenbaum for their useful discussions.

References

[1] M. Blum and D. Kozen, *On the power of the compass, or*, Why mazes are easier to search than graphs," in FOCS **78** (1978), 132–142.

[2] Lothar Budach, *Environments, labyrinths and automata*, Fundamentals of Computation Theory, 1977, pp. 54–64.

[3] Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, and David Peleg, *Label-guided graph exploration by a finite automaton*, ICALP (Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, eds.), Lecture Notes in Computer Science, vol. 3580, Springer, 2005, pp. 335–346.

[4] Stephen A. Cook and Charles Rackoff, *Space lower bounds for maze threadability on restricted machines*, SIAM J. Comput. **9** (1980), no. 3, 636–652.

[5] Jurek Czyzowicz, Stefan Dobrev, Leszek Gasieniec, David Ilcinkas, Jesper Jansson, Ralf Klasing, Ioannis Lignos, Russell A. Martin, Kunihiko Sadakane, and Wing-Kin Sung, *More efficient periodic traversal in anonymous undirected graphs*, SIROCCO (Shay Kutten and Janez Zerovnik, eds.), Lecture Notes in Computer Science, vol. 5869, Springer, 2009, pp. 167–181.

[6] Krzysztof Diks, Pierre Fraigniaud, Evangelos Kranakis, and Andrzej Pelc, *Tree exploration with little memory*, J. Algorithms **51** (2004), no. 1, 38–63.

[7] Stefan Dobrev, Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung, *Finding short right-hand-on-the-wall walks in graphs*, SIROCCO (Andrzej Pelc and Michel Raynal, eds.), Lecture Notes in Computer Science, vol. 3499, Springer, 2005, pp. 127–139.

[8] Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg, *Graph exploration by a finite automaton*, Theor. Comput. Sci. **345** (2005), no. 2-3, 331–344.

[9] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc, *Impact of memory size on graph exploration capability*, Discrete Applied Mathematics **156** (2008), no. 12, 2310–2319.

[10] Leszek Gasieniec, Ralf Klasing, Russell A. Martin, Alfredo Navarra, and Xiaohui Zhang, *Fast periodic graph exploration with constant memory*, J. Comput. Syst. Sci. **74** (2008), no. 5, 808–822.

[11] Liang Hu, Meng Zhang, Yi Zhang, and Jijun Tang, *Label-guided graph exploration with adjustable ratio of labels*, Int. J. Found. Comput. Sci. **23** (2012), no. 4, 903–930.

[12] David Ilcinkas, *Setting port numbers for fast graph exploration*, SIROCCO (Paola Flocchini and Leszek Gasieniec, eds.), Lecture Notes in Computer Science, vol. 4056, Springer, 2006, pp. 59–69.

[13] Adrian Kosowski and Alfredo Navarra, *Graph decomposition for improving memoryless periodic exploration*, MFCS (Rastislav Královic and Damian Niwinski, eds.), Lecture Notes in Computer Science, vol. 5734, Springer, 2009, pp. 501–512.

[14] D. Kozen, *Automata and planar graphs*, Fund. Computat. Theory (FCT) **2** (1979), 43–2.

[15] MO Rabin, *Maze threading automata*, An unpublished lecture presented at MIT and UC Berkeley, 1967.

[16] Omer Reingold, *Undirected st-connectivity in log-space*, STOC (Harold N. Gabow and Ronald Fagin, eds.), ACM, 2005, pp. 376–385.

[17] Robert W. Robinson and Nicholas C. Wormald, *Almost all regular graphs are hamiltonian*, Random Struct. Algorithms **5** (1994), no. 2, 363–374.

[18] Hans-Anton Rollik, *Automaten in planaren graphen*, Acta Inf. **13** (1980), 287–298.

[19] C.E. Shannon, *Presentation of a maze-solving machine*, 8th Conf. of the Josiah Macy Jr. Found.(Cybernetics), 1951, pp. 173–180.