

Periodic Graph Exploration Using an Oblivious Agent

Jurek Czyzowicz¹ Leszek Gasieniec²
David Ilcinkas³ Ralf Klasing³

¹Université du Québec en Outaouais, Canada

²University of Liverpool, United Kingdom

³CNRS and University of Bordeaux (LaBRI), France

DYNAMO meeting
September 26, 2008

Problem

Periodic graph exploration

A **mobile entity**, called *agent*, has to **visit every node** of an unknown anonymous graph **infinitely often**.

Efficiency measure

Period: length of the tour, i.e., maximal number of edge traversals between two visits of the same node

Motivation: Network maintenance by a software agent

Problem

Periodic graph exploration

A **mobile entity**, called *agent*, has to **visit every node** of an unknown anonymous graph **infinitely often**.

Efficiency measure

Period: length of the tour, i.e., maximal number of edge traversals between two visits of the same node

Motivation: Network maintenance by a software agent

Problem

Periodic graph exploration

A **mobile entity**, called *agent*, has to **visit every node** of an unknown anonymous graph **infinitely often**.

Efficiency measure

Period: length of the tour, i.e., maximal number of edge traversals between two visits of the same node

Motivation: Network maintenance by a software agent

Unknown, anonymous graphs

Unknown

- Unknown topology
- Unknown size

Anonymous

- No node labeling
- Local port numbering at node v from 1 to $\deg(v)$

Unknown, anonymous graphs

Unknown

- Unknown topology
- Unknown size

Anonymous

- No node labeling
- Local port numbering at node v from 1 to $\deg(v)$

Unknown, anonymous graphs

Unknown

- Unknown topology
- Unknown size

Anonymous

- No node labeling
- **Local port numbering** at node v from 1 to $\deg(v)$

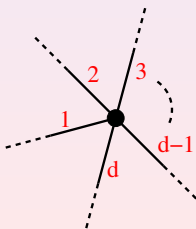
Unknown, anonymous graphs

Unknown

- Unknown topology
- Unknown size

Anonymous

- No node labeling
- **Local port numbering** at node v from 1 to $\deg(v)$



Memory constraint

Objective

Use agents with a **memory of constant size**

Justifications

- Simple and cost effective agents
- Facilitates design and analysis of algorithms

Model

The agent is modeled as a finite Mealy automaton.

Memory constraint

Objective

Use agents with a **memory of constant size**

Justifications

- **Simple and cost effective agents**
- Facilitates design and analysis of algorithms

Model

The agent is modeled as a finite Mealy automaton.

Memory constraint

Objective

Use agents with a **memory of constant size**

Justifications

- **Simple and cost effective agents**
- Facilitates design and analysis of algorithms

Model

The agent is modeled as a finite **Mealy automaton**.

Mealy automaton

Input

- S : current state
- i : input port number
- d : node's degree

Output

- S' : new state
- j : output port number

Transition function

- $f : (S, i, d) \mapsto (S', j)$

Oblivious agent (one single state)

- Transition functions $f_d : i \rightarrow j$ for $d \geq 1$

Mealy automaton

Input

- S : current state
- i : input port number
- d : node's degree

Output

- S' : new state
- j : output port number

Transition function

- $f : (S, i, d) \mapsto (S', j)$

Oblivious agent (one single state)

- Transition functions $f_d : i \rightarrow j$ for $d \geq 1$

Motivations (cont'd)

USTCON (undirected st-connectivity)

- $G = \{V, E\}$ an undirected graph
- $s, t \in V$ two vertices of G

Are s and t in the same connected component of G ?

- L = class of problems solvable by deterministic log-space computations
- $SL (\supseteq L)$ = class of problems solvable by symmetric non-deterministic log-space computations

Motivations (cont'd)

USTCON (undirected st-connectivity)

- $G = \{V, E\}$ an undirected graph
- $s, t \in V$ two vertices of G

Are s and t in the same connected component of G ?

- L = class of problems solvable by deterministic log-space computations
- $SL (\supseteq L)$ = class of problems solvable by symmetric non-deterministic log-space computations

Motivations (cont'd)

USTCON (undirected st-connectivity) SL-complete

- $G = \{V, E\}$ an undirected graph
- $s, t \in V$ two vertices of G

Are s and t in the same connected component of G ?

- L = class of problems solvable by deterministic log-space computations
- $SL (\supseteq L)$ = class of problems solvable by symmetric non-deterministic log-space computations

Motivations (cont'd)

USTCON (undirected st-connectivity) SL-complete

- $G = \{V, E\}$ an undirected graph
- $s, t \in V$ two vertices of G

Are s and t in the same connected component of G ?

- L = class of problems solvable by deterministic log-space computations
- $SL (\supseteq L)$ = class of problems solvable by symmetric non-deterministic log-space computations

Reingold, STOC 2005

Undirected ST-Connectivity in Log-Space

$USTCON \in L \Rightarrow SL=L$

Impossibility results

Rollik, Acta Informatica, 1980

An agent able to explore the n -node graphs needs $\Omega(\log n)$ memory bits.

A pebble is a node-marker that can be dropped at and removed from nodes.

Essays in Memory of Shimon Even, 2006

Even with a pebble, the agent still needs $\Omega(\log n)$ memory bits.

A JAG (Jumping Automaton for Graphs) is a team of finite automata that cooperate constantly. Moreover an automaton can jump to a vertex occupied by another automaton.

SIAMJC, 1980

No JAG can explore all graphs.

Impossibility results

Rollik, Acta Informatica, 1980

An agent able to explore the n -node graphs needs $\Omega(\log n)$ memory bits.

A **pebble** is a node-marker that can be dropped at and removed from nodes.

Essays in Memory of Shimon Even, 2006

Even with a pebble, the agent still needs $\Omega(\log n)$ memory bits.

A JAG (Jumping Automaton for Graphs) is a team of finite automata that cooperate constantly. Moreover an automaton can jump to a vertex occupied by another automaton.

SIAMJC, 1980

No JAG can explore all graphs.

Impossibility results

Rollik, Acta Informatica, 1980

An agent able to explore the n -node graphs needs $\Omega(\log n)$ memory bits.

A **pebble** is a node-marker that can be dropped at and removed from nodes.

Fraigniaud et al., Essays in Memory of Shimon Even, 2006

Even with a pebble, the agent still needs $\Omega(\log n)$ memory bits.

A JAG (Jumping Automaton for Graphs) is a team of finite automata that cooperate constantly. Moreover an automaton can jump to a vertex occupied by another automaton.

Alfred Aho, SIAMJC, 1980

No JAG can explore all graphs.

Impossibility results

Rollik, Acta Informatica, 1980

An agent able to explore the n -node graphs needs $\Omega(\log n)$ memory bits.

A **pebble** is a node-marker that can be dropped at and removed from nodes.

Fraigniaud et al., Essays in Memory of Shimon Even, 2006

Even with a pebble, the agent still needs $\Omega(\log n)$ memory bits.

A **JAG (Jumping Automaton for Graphs)** is a team of finite automata that cooperate constantly. Moreover an automaton can jump to a vertex occupied by another automaton.

SIAMJC, 1980

No JAG can explore all graphs

Impossibility results

Rollik, Acta Informatica, 1980

An agent able to explore the n -node graphs needs $\Omega(\log n)$ memory bits.

A **pebble** is a node-marker that can be dropped at and removed from nodes.

Fraigniaud et al., Essays in Memory of Shimon Even, 2006

Even with a pebble, the agent still needs $\Omega(\log n)$ memory bits.

A **JAG (Jumping Automaton for Graphs)** is a team of finite automata that cooperate constantly. Moreover an automaton can jump to a vertex occupied by another automaton.

Cook, Rackoff, SIAMJC, 1980

No JAG can explore all graphs.

Giving advice

Providing **additionnal information** does help.

Model

- An oracle puts bits of **advice** at the graph nodes to help the agent.
- The agent can read these bits as an input of its transition function.

Chang and Kulkarni, ACM Trans. Algo., 2008

- 1 bit of advice per node:
Constant memory suffices for constant-degree graphs.
- 2 bits of advice per node:
Constant memory suffices for arbitrary graphs.

In both cases, $\text{period} = O(m)$

Giving advice

Providing **additionnal information** does help.

Model

- An oracle puts bits of **advice** at the graph nodes to help the agent.
- The agent can read these bits as an input of its transition function.

Cohen, Fraigniaud, I., Korman, Peleg, ACM Trans. Algo., 2008

- **1 bit** of advice per node:
Constant memory suffices for **constant-degree graphs**.
- **2 bits** of advice per node:
Constant memory suffices for **arbitrary graphs**.

In both cases, **period** = $O(m)$

Setting port numbers

Observation

All impossibility results are based on a **misleading assignment of the port numbers**.

A solution

Port numbers are set to help the automaton.

SIROCCO, 2005

There exist an algorithm for setting the port numbers, and an oblivious agent using them, such that the agent explores all graphs of size n within the period $10n$.

Setting port numbers

Observation

All impossibility results are based on a **misleading assignment of the port numbers**.

A solution

Port numbers are set to help the automaton.

SIROCCO, 2005

There exist an algorithm for setting the port numbers, and an oblivious agent using them, such that the agent explores all graphs of size n within the period $10n$.

Setting port numbers

Observation

All impossibility results are based on a **misleading assignment of the port numbers**.

A solution

Port numbers are set to help the automaton.

Dobrev, Jansson, Sadakane, Sung, SIROCCO, 2005

There exist an algorithm for setting the port numbers, and an **oblivious agent** using them, such that the agent explores all graphs of size n within the **period $10n$** .

The non-oblivious case

Better upper bounds are known for constant-memory agents.

Chen et al., TCS, 2003

Length of the tour $\leq 4n$

Chen et al.,

JCSS, 2007

Length of the tour $\leq 3.75n$

Chen et al., under submission

Length of the tour $\leq 3.5n$

The non-oblivious case

Better upper bounds are known for constant-memory agents.

Ilcinkas, TCS, 2008

Length of the tour $\leq 4n$

JCSS, 2007

Length of the tour $\leq 3.75n$

, under submission

Length of the tour $\leq 3.5n$

The non-oblivious case

Better upper bounds are known for constant-memory agents.

Ilcinkas, TCS, 2008

Length of the tour $\leq 4n$

Gasieniec, Klasing, Martin, Navarra, Zhang, JCSS, 2007

Length of the tour $\leq 3.75n$

under submission

Length of the tour $\leq 3.5n$

The non-oblivious case

Better upper bounds are known for constant-memory agents.

Ilcinkas, TCS, 2008

Length of the tour $\leq 4n$

Gasieniec, Klasing, Martin, Navarra, Zhang, JCSS, 2007

Length of the tour $\leq 3.75n$

Czyzowicz et al., under submission

Length of the tour $\leq 3.5n$

Our results

Question

What is the **minimum** α such that there exist an algorithm for **setting the port numbers**, and an **oblivious agent** using it, such that the automaton explores all graphs of size n **within the period** $\alpha \cdot n$?

Main result

$$2.8 \leq \alpha \leq 4.333 \dots$$

Complementary result

If there exists a spanning tree T of $G = (V, E)$ such that none of the nodes is saturated (i.e. $\forall v \in V \quad \deg_T(v) \neq \deg_G(v)$), then period $2n$ can be achieved by an oblivious agent.

Our results

Question

What is the **minimum** α such that there exist an algorithm for **setting the port numbers**, and an **oblivious agent** using it, such that the automaton explores all graphs of size n **within the period** $\alpha \cdot n$?

Main result

$$2.8 \leq \alpha \leq 4.333\dots$$

Complementary result

If there exists a spanning tree T of $G = (V, E)$ such that none of the nodes is saturated (i.e. $\forall v \in V \quad \deg_T(v) \neq \deg_G(v)$), then period $2n$ can be achieved by an oblivious agent.

Our results

Question

What is the **minimum** α such that there exist an algorithm for **setting the port numbers**, and an **oblivious agent** using it, such that the automaton explores all graphs of size n **within the period** $\alpha \cdot n$?

Main result

$$2.8 \leq \alpha \leq 4.333 \dots$$

Complementary result

If there exists a **spanning tree** T of $G = (V, E)$ such that **none** of the nodes is **saturated** (i.e. $\forall v \in V \quad \deg_T(v) \neq \deg_G(v)$), then period $2n$ can be achieved by an oblivious agent.

A useful observation

Property

The algorithm of **any** oblivious agent able to explore all graphs is “equivalent” to the **Right-Hand-on-the-Wall** rule.

Right-Hand-on-the-Wall rule is $f_d : j \rightarrow j + 1$ for $d \geq 1$

Proof

For any degree d , the transition function f_d has to be a cyclic permutation, which is “equivalent” to the Right-Hand rule.

A useful observation

Property

The algorithm of **any** oblivious agent able to explore all graphs is “equivalent” to the **Right-Hand-on-the-Wall** rule.

Right-Hand-on-the-Wall rule is $f_d : i \rightarrow i + 1$ for $d \geq 1$

Proof

For any degree d , the transition function f_d has to be a cyclic permutation, which is “equivalent” to the **Right-Hand** rule.

A useful observation

Property

The algorithm of **any** oblivious agent able to explore all graphs is “equivalent” to the **Right-Hand-on-the-Wall** rule.

Right-Hand-on-the-Wall rule is $f_d : i \rightarrow i + 1$ for $d \geq 1$

Proof

For any degree d , the transition function f_d has to be a **cyclic permutation**, which is “equivalent” to the **Right-Hand rule**.

A useful observation

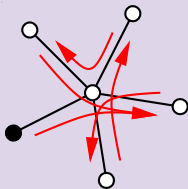
Property

The algorithm of **any** oblivious agent able to explore all graphs is “equivalent” to the **Right-Hand-on-the-Wall** rule.

Right-Hand-on-the-Wall rule is $f_d : i \rightarrow i + 1$ for $d \geq 1$

Proof

For any degree d , the transition function f_d has to be a **cyclic permutation**, which is “equivalent” to the **Right-Hand rule**.



A useful observation

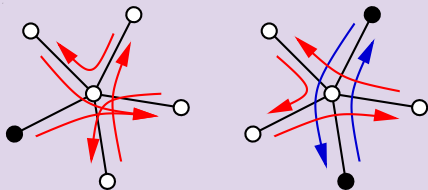
Property

The algorithm of **any** oblivious agent able to explore all graphs is “equivalent” to the **Right-Hand-on-the-Wall** rule.

Right-Hand-on-the-Wall rule is $f_d : i \rightarrow i + 1$ for $d \geq 1$

Proof

For any degree d , the transition function f_d has to be a **cyclic permutation**, which is “equivalent” to the **Right-Hand rule**.



A useful observation

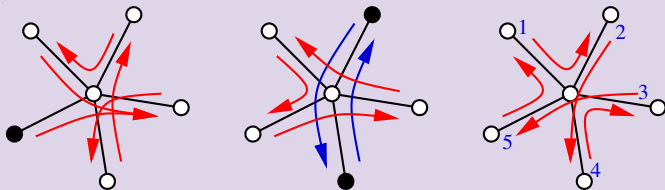
Property

The algorithm of **any** oblivious agent able to explore all graphs is “equivalent” to the **Right-Hand-on-the-Wall** rule.

Right-Hand-on-the-Wall rule is $f_d : i \rightarrow i + 1$ for $d \geq 1$

Proof

For any degree d , the transition function f_d has to be a **cyclic permutation**, which is “equivalent” to the **Right-Hand rule**.



A useful observation

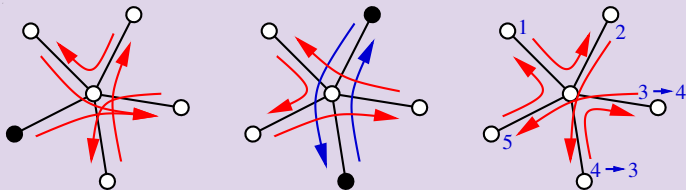
Property

The algorithm of **any** oblivious agent able to explore all graphs is “equivalent” to the **Right-Hand-on-the-Wall** rule.

Right-Hand-on-the-Wall rule is $f_d : i \rightarrow i + 1$ for $d \geq 1$

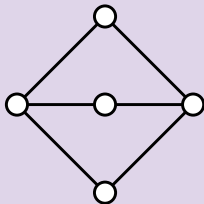
Proof

For any degree d , the transition function f_d has to be a **cyclic permutation**, which is “equivalent” to the **Right-Hand rule**.



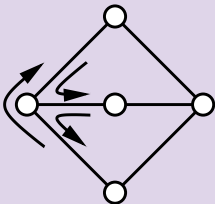
Lower bound

Lower bound: $\alpha \geq 2.4$



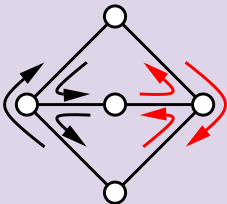
Lower bound

Lower bound: $\alpha \geq 2.4$



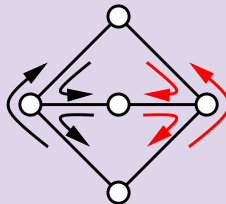
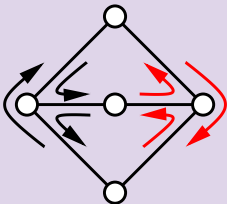
Lower bound

Lower bound: $\alpha \geq 2.4$



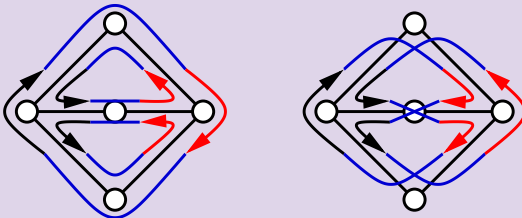
Lower bound

Lower bound: $\alpha \geq 2.4$



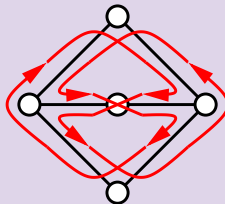
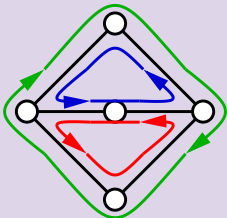
Lower bound

Lower bound: $\alpha \geq 2.4$



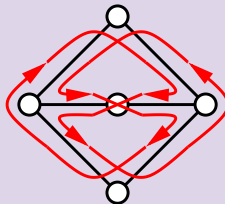
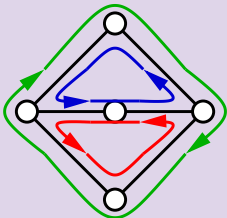
Lower bound

Lower bound: $\alpha \geq 2.4$

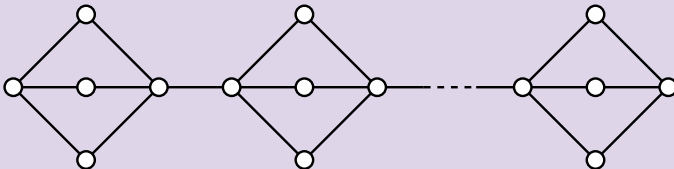


Lower bound

Lower bound: $\alpha \geq 2.4$



Lower bound: $\alpha \geq 2.8$



General technique

Specific directed spanner

Construction of a spanning directed subgraph H of the symmetric directed version of G such that

- for every node, $\#$ incoming arcs = $\#$ outgoing arcs
- for every node, either it is saturated or an arc incident to it belongs to H but not its symmetric arc
- there exists a spanning tree composed of pairs of symmetric arcs

Property

From H , one can construct a tour spanning G .

Performance

Length of the tour \leq number of arcs in H

General technique

Specific directed spanner

Construction of a spanning directed subgraph H of the symmetric directed version of G such that

- for every node, $\#$ incoming arcs = $\#$ outgoing arcs
- for every node, either it is saturated or an arc incident to it belongs to H but not its symmetric arc
- there exists a spanning tree composed of pairs of symmetric arcs

Property

From H , one can construct a tour spanning G .

Performance

Length of the tour \leq number of arcs in H

General technique

Specific directed spanner

Construction of a spanning directed subgraph H of the symmetric directed version of G such that

- for every node, $\#$ incoming arcs = $\#$ outgoing arcs
- for every node, either it is saturated or an arc incident to it belongs to H but not its symmetric arc
- there exists a spanning tree composed of pairs of symmetric arcs

Property

From H , one can construct a tour spanning G .

Performance

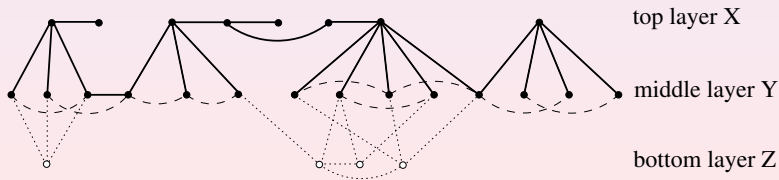
Length of the tour \leq number of arcs in H

Three-layer partition

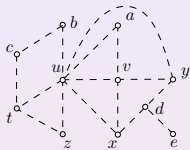
Definition

A **three-layer partition** of a graph $G = (V, E)$ is a 4-uplet (X, Y, Z, T) such that

- the three sets X , Y and Z form a **partition** of V
- $Y = N_G(X)$ and $Z = N_G(Y) \setminus X$
- T is a tree of node-set $X \cup Y$ where **all nodes in X are saturated**

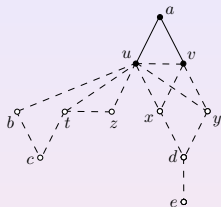


How to construct it

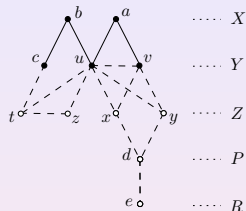


all nodes belong to R here

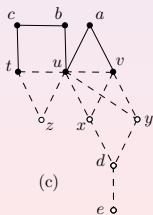
(-)



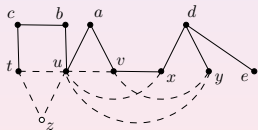
(a)



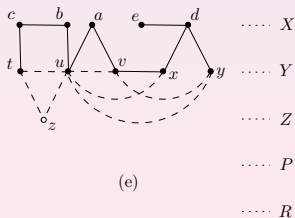
(b)



(c)



(d)



(e)

Conclusion and perspectives

Open problem

Exact value for minimum α

Variants

Best tour for a given graph (NP-hard problem)

Conclusion and perspectives

Open problem

Exact value for minimum α

Variant

Best tour for a given graph (NP-hard problem)

Thank You
for your attention