Jeux des gendarmes et du voleur dans les graphes.

Nicolas Nisse

LRI, Université Paris-Sud, France.

Réunion FRAGILE
19 juin 2007

Outline

Motivation: Practical Applications

Genese

A speleologist is lost in a caves'network. What is the smallest number of persons that is required to save him? How to compute a rescue strategy? [Breish 67, Parson 78]

Auto-coordination of mobile agents

- Surveillance of building,
- Localisation of a mobile target,
- Clearing of a contaminated pipeline's network,
- Clearing of a contaminated internet network, etc.

Motivation: Practical Applications

Genese

A speleologist is lost in a caves'network. What is the smallest number of persons that is required to save him? How to compute a rescue strategy? [Breish 67, Parson 78]

Auto-coordination of mobile agents

- Surveillance of building,
- Localisation of a mobile target,
- Clearing of a contaminated pipeline's network,
- Clearing of a contaminated internet network, etc.

Motivations: Fundamental Approaches

VLSI design

Embedding of circuit layout.

Pebble games

Model for the allocation of registers in a processor.

- Number of pebbles = space complexity
- Number of *moves* = time complexity

Graph Minors Theory, Robertson and Seymour

Wagner's conjecture: any minor-closed class of graphs admits a finite obstruction set (e.g., Kuratowski's theorem);
Tree-like decompositions of graphs excluding a minor.

Motivations: Fundamental Approaches

VLSI design

Embedding of circuit layout.

Pebble games

Model for the allocation of registers in a processor.

- Number of pebbles = space complexity
- Number of *moves* = time complexity

Graph Minors Theory, Robertson and Seymour

Wagner's conjecture: any minor-closed class of graphs admits a finite obstruction set (e.g., Kuratowski's theorem);
Tree-like decompositions of graphs excluding a minor.

Motivations: Fundamental Approaches

VLSI design

Embedding of circuit layout.

Pebble games

Model for the allocation of registers in a processor.

- Number of pebbles = space complexity
- Number of *moves* = time complexity

Graph Minors Theory, Robertson and Seymour

Wagner's conjecture: any minor-closed class of graphs admits a finite obstruction set (e.g., Kuratowski's theorem);
Tree-like decompositions of graphs excluding a minor.

General problem

Context

A fugitive is running in a graph.
A team of searchers is aiming at capturing the fugitive.

Goal(Alternative goal)

To design a strategy that capture **any** fugitive (clear the contaminated graph) using the **fewest searchers as possible**.

Variants of graph searching games

- **fugitive/searchers' visibility**: visible or invisible; (Case fugitive and searchers invisible: random walk, graph's exploration)
- **playing rules**: turn by turn, or simultaneous moves;
- **way to capture the fugitive**: same location, domination;
- **fugitive/searchers'moves**: move along edges or/and jump from a vertex to another one;
- **fugitive/searchers' velocity**: bounded speed or arbitrary fast.

Variants of graph searching games

- **fugitive/searchers' visibility**: visible or invisible; (Case fugitive and searchers invisible: random walk, graph's exploration)
- **playing rules**: turn by turn, or simultaneous moves;
- **way to capture the fugitive**: same location, domination;
- **fugitive/searchers'moves**: move along edges or/and jump from a vertex to another one;
- **fugitive/searchers' velocity**: bounded speed or arbitrary fast.

Variants of graph searching games

- **fugitive/searchers' visibility**: visible or invisible; (Case fugitive and searchers invisible: random walk, graph's exploration)
- **playing rules**: turn by turn, or simultaneous moves;
- **way to capture the fugitive**: same location, domination;
- **fugitive/searchers'moves**: move along edges or/and jump from a vertex to another one;
- **fugitive/searchers' velocity**: bounded speed or arbitrary fast.

Variants of graph searching games

- **fugitive/searchers' visibility**: visible or invisible; (Case fugitive and searchers invisible: random walk, graph's exploration)
- **playing rules**: turn by turn, or simultaneous moves;
- **way to capture the fugitive**: same location, domination;
- **fugitive/searchers'moves**: move along edges or/and jump from a vertex to another one;
- **fugitive/searchers' velocity**: bounded speed or arbitrary fast.

Variants of graph searching games

- **fugitive/searchers' visibility**: visible or invisible; (Case fugitive and searchers invisible: random walk, graph's exploration)
- **playing rules**: turn by turn, or simultaneous moves;
- **way to capture the fugitive**: same location, domination;
- **fugitive/searchers'moves**: move along edges or/and jump from a vertex to another one;
- **fugitive/searchers' velocity**: bounded speed or arbitrary fast.

Taxonomy of graph searching games

	fugitive's caracteristics			
	bounded speed		arbitrary fast	
	visible	invisible	visible	invisible
turn by turn game	Cops and robber Quilliot 83, Nowakowski and Winkler 83	Clarke and Nowakowski 00	?	?
simultaneous moves	?	Fomin 98	Seymour and Thomas 93	Graph searching Breish 67, Parson 78

Table: Classification of the graph searching games

Taxonomy of graph searching games

	fugitive's caracteristics			
	bounded speed		arbitrary fast	
	visible	invisible	visible	invisible
turn by turn game	Cops and robber Quilliot 83, Nowakowski and Winkler 83	Clarke and Nowakowski 00	?	?
simultaneous moves	?	Fomin 98	Seymour and Thomas 93	Graph searching Breish 67, Parson 78

Table: Classification of the graph searching games

Search Strategy, Parson. [GTC,1978] Variant of Kirousis and Papadimitriou. [TCS,86]

Sequence of two basic operations,. . .

1. Place a searcher at a vertex of the graph;
2. Remove a searcher from a vertex of the graph.

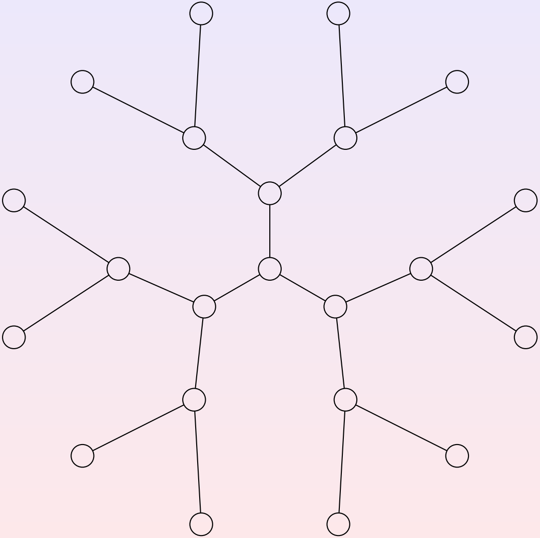
. . . that must result in catching the fugitive

The fugitive moves from one vertex to another by following the paths of the graph.
It is caugth when it meets a searcher at a vertex.

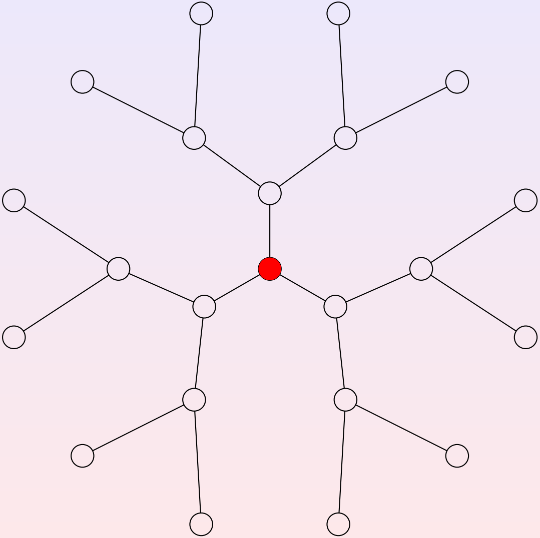
The node-search number

Let $s(G)$ be the smallest number of searchers needed to catch an **invisible** fugitive in a graph G.

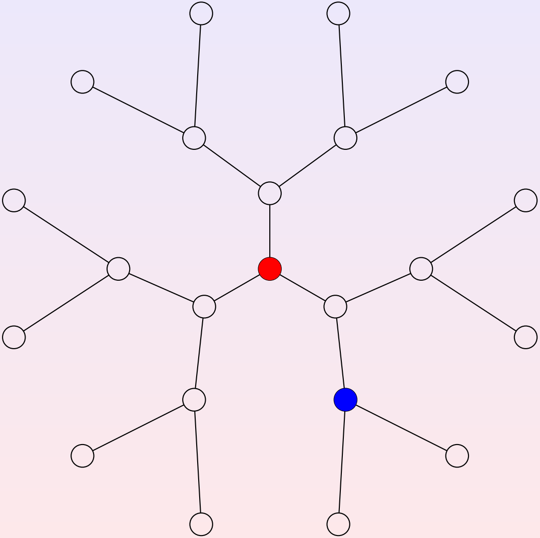
Simple example: a ternary tree



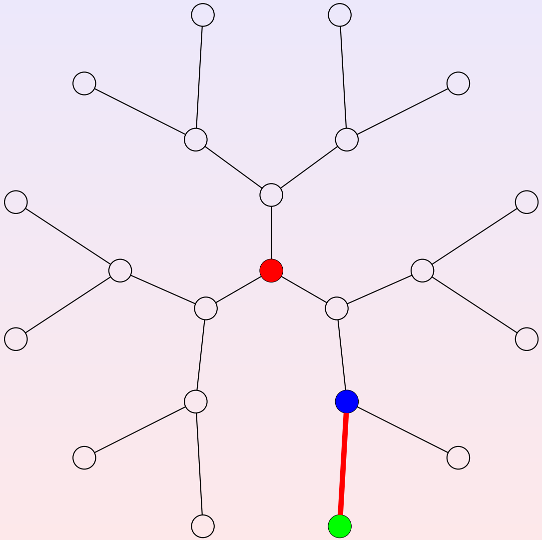
Simple example: a ternary tree



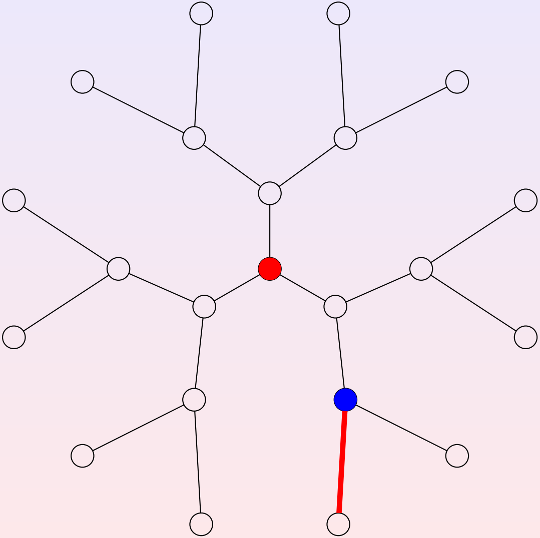
Simple example: a ternary tree



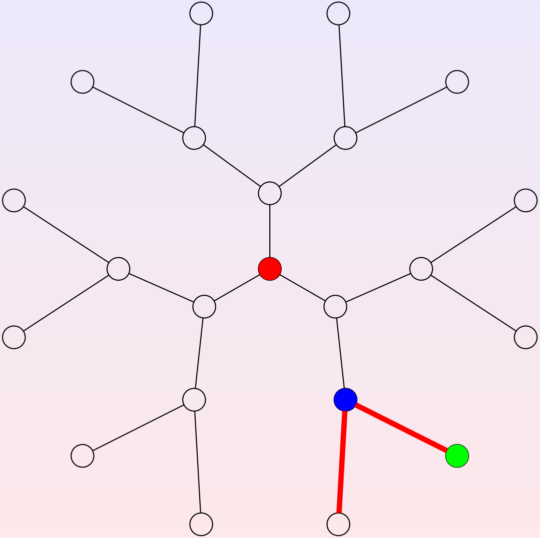
Simple example: a ternary tree



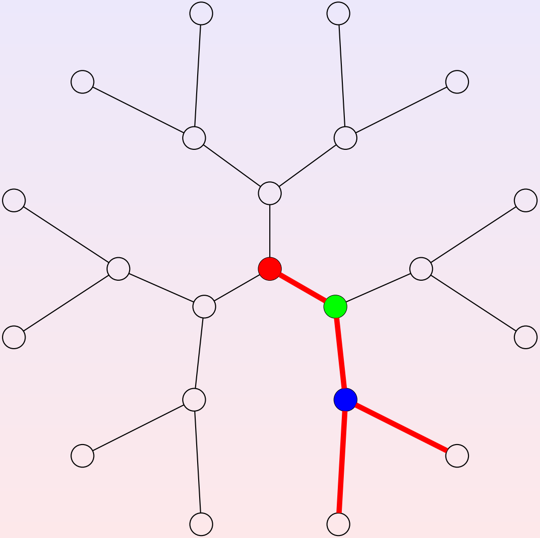
Simple example: a ternary tree



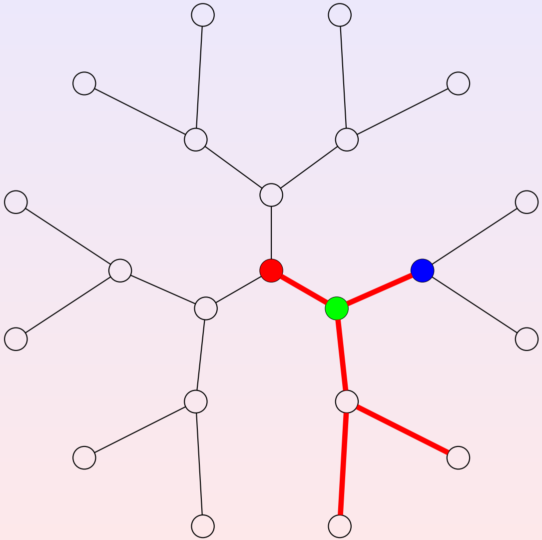
Simple example: a ternary tree



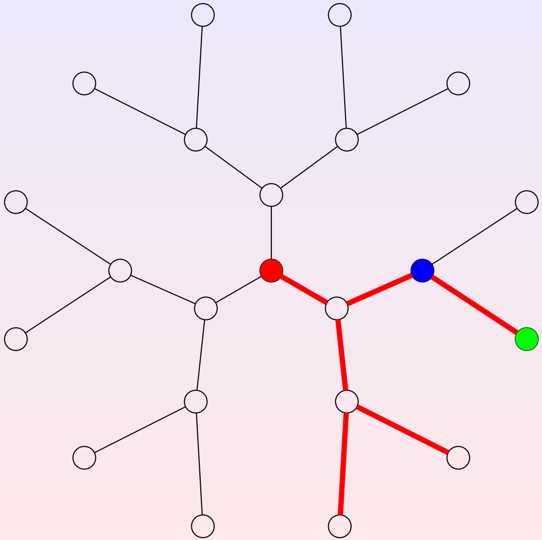
Simple example: a ternary tree



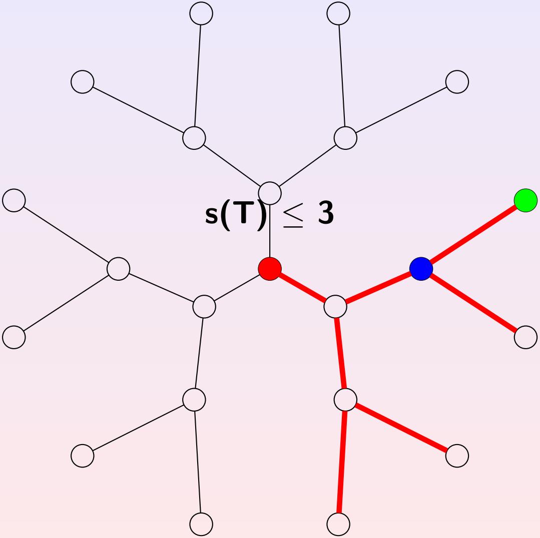
Simple example: a ternary tree



Simple example: a ternary tree



Simple example: a ternary tree



Visibility of the fugitive

Visible fugitive

The fugitive is visible if, at every step, searchers know its position.
Let $vs(G)$ be the visible search number of the graph G.

Obviously, for any graph G, $vs(G) \leq s(G)$.

In trees

For any n-nodes tree T, $s(T) \leq 1 + \log_3(n-1)$ (tight)
Megiddo *et. al.* [JACM 88]

For any tree T (with at least 2 vertices), $vs(T) = 2$.

Visibility of the fugitive

Visible fugitive

The fugitive is visible if, at every step, searchers know its position.
Let $vs(G)$ be the visible search number of the graph G.

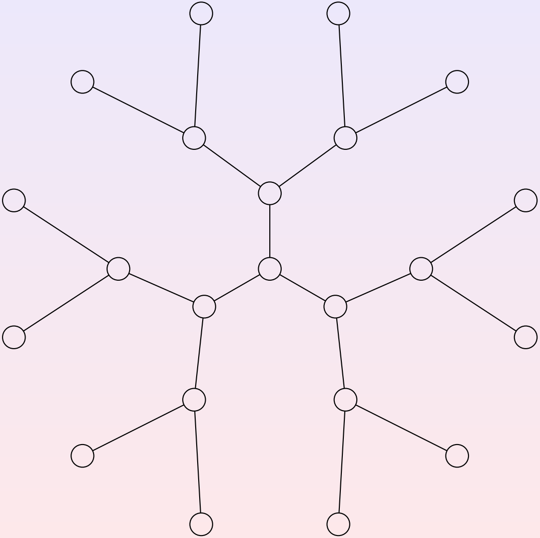
Obviously, for any graph G, $vs(G) \leq s(G)$.

In trees

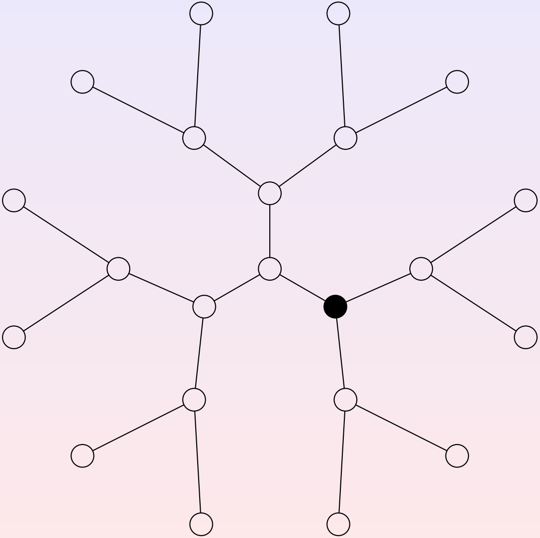
For any n-nodes tree T, $s(T) \leq 1 + \log_3(n-1)$ (tight)
Megiddo *et. al.* [JACM 88]

For any tree T (with at least 2 vertices), $vs(T) = 2$.

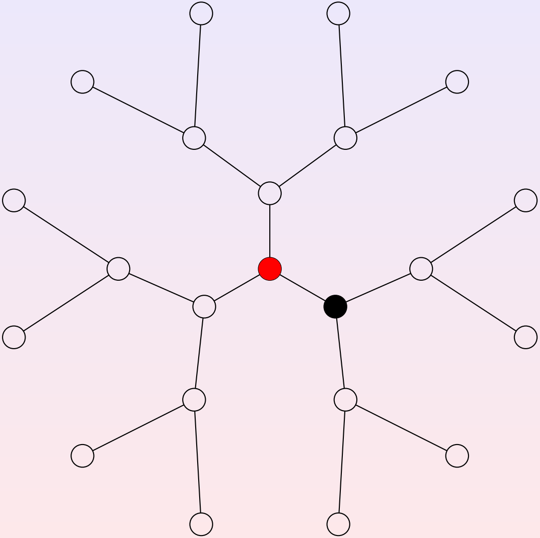
Visible graph searching in a tree



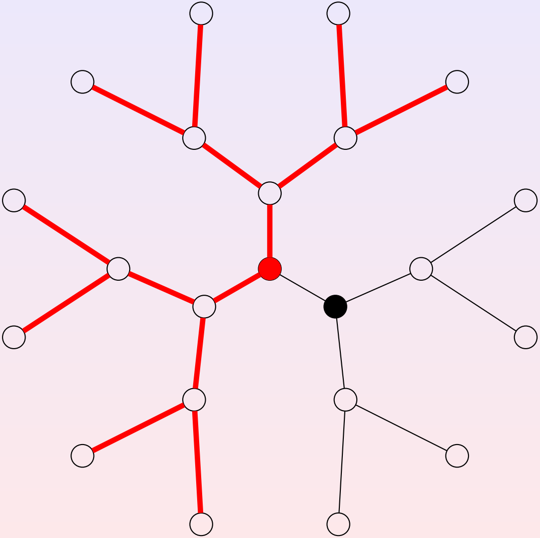
Visible graph searching in a tree



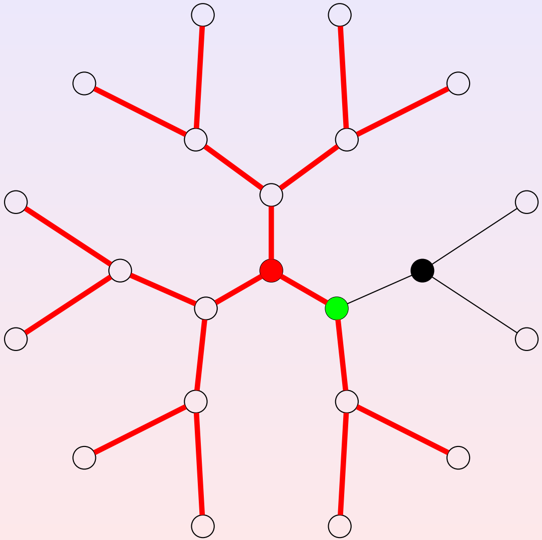
Visible graph searching in a tree



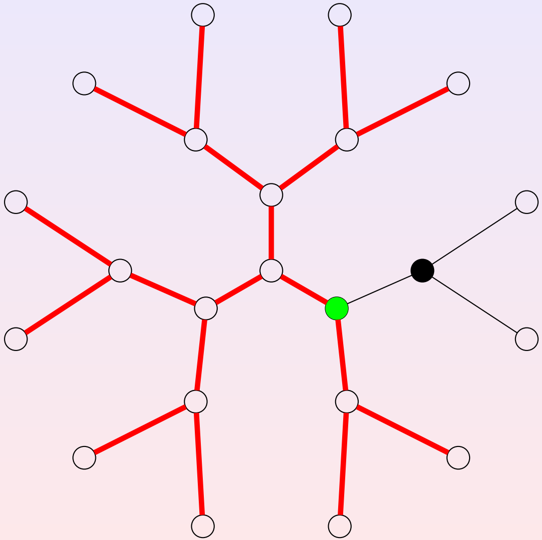
Visible graph searching in a tree



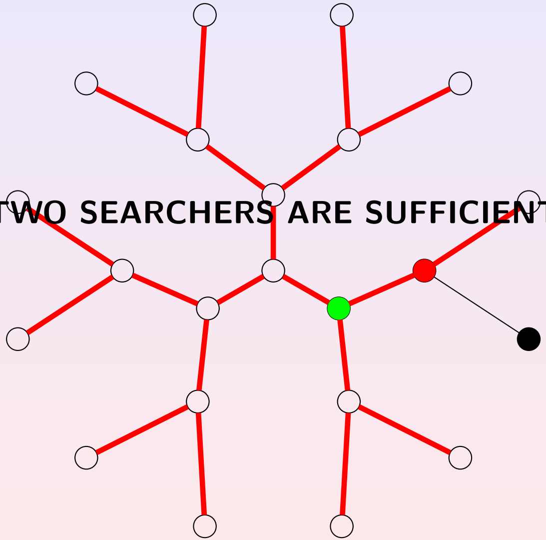
Visible graph searching in a tree



Visible graph searching in a tree



Visible graph searching in a tree



TWO SEARCHERS ARE SUFFICIENT

NP-hardness

The following problems are NP-hard

Input: a graph G, an integer $k > 0$, — Megiddo *et. al.*, [JACM 88]
Output: $s(G) \leq k$?

Input: a graph G , an integer $k > 0$, — Seymour and Thomas [JCTB 93]
Output: $vs(G) \leq k$?

Remark: linear in the class of trees, Skodinis [JAlg 03]

Do these problems belong to NP? Certificate?

NP-hardness

The following problems are NP-hard

Input: a graph G, an integer $k > 0$,		Megiddo *et. al.*,
Output: $s(G) \leq k$?		[JACM 88]
Input: a graph G , an integer $k > 0$,		Seymour and Thomas
Output: $vs(G) \leq k$?		[JCTB 93]

Remark: linear in the class of trees, Skodinis [JAlg 03]
Do these problems belong to NP? Certificate?

Monotonicity and NP-completeness

A vertex v is recontaminated if the fugitive can move to v after v has been occupied by a searcher.

Monotonicity

A search strategy is monotone if no recontamination ever occurs. That is, a vertex is occupied by a searcher only once.

Recontamination does not help

Threre always exists an optimal monotone search strategy.

invisible fugitive:	LaPaugh, [JACM 93]	Bienstock and Seymour [JAlg 91]
visible fugitive:	Seymour and Thomas [JCTB 93]	

Corollary: The above problems belong to NP.

Monotonicity and NP-completeness

A vertex v is recontaminated if the fugitive can move to v after v has been occupied by a searcher.

Monotonicity

A search strategy is monotone if no recontamination ever occurs. That is, a vertex is occupied by a searcher only once.

Recontamination does not help

There always exists an optimal monotone search strategy.

invisible fugitive:	LaPaugh, [JACM 93]	Bienstock and Seymour [JAlg 91]
visible fugitive:	Seymour and Thomas [JCTB 93]	

Corollary: The above problems belong to NP.

Monotonicity and NP-completeness

A vertex v is recontaminated if the fugitive can move to v after v has been occupied by a searcher.

Monotonicity

A search strategy is monotone if no recontamination ever occurs. That is, a vertex is occupied by a searcher only once.

Recontamination does not help

There always exists an optimal monotone search strategy.

invisible fugitive: LaPaugh, [JACM 93] Bienstock and Seymour [JAlg 91]

visible fugitive: Seymour and Thomas [JCTB 93]

Corollary: The above problems belong to NP.

Search numbers and graphs'decompositions

Thanks to the monotonicity, we get:

Search number and Pathwidth (pw)

For any graph G, $s(G) = pw(G) + 1$
Kinnersley [IPL 92],
Ellis, Sudborough, and Turner [Inf.Comp.94]

Visible search number and Treewidth (tw)

For any graph G, $vs(G) = tw(G) + 1$
Seymour and Thomas [JCTB 93]

Outline

Non-deterministic Graph Searching

Invisible fugitive

An Oracle permanently knows the position of the fugitive

One extra operation is allowed

Searchers can perform a query to the oracle: “What is the current position of the fugitive?”

Sequence of three basic operations

1. Place a searcher at a vertex of the graph;
2. Remove a searcher from a vertex of the graph;
3. Perform a query to the Oracle.

Tradeoff number of searchers / number of queries

q-limited (non-deterministic) search number, $s_q(G)$

Non-deterministic Graph Searching

Invisible fugitive

An Oracle permanently knows the position of the fugitive

One extra operation is allowed

Searchers can perform a query to the oracle: "What is the current position of the fugitive?"

Sequence of three basic operations

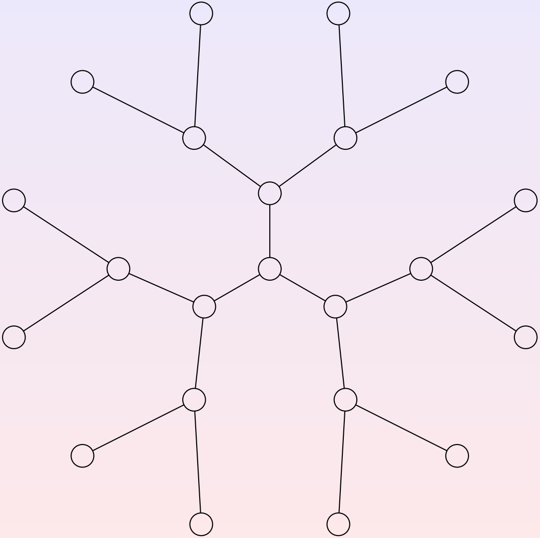
1. Place a searcher at a vertex of the graph;
2. Remove a searcher from a vertex of the graph;
3. Perform a query to the Oracle.

Tradeoff number of searchers / number of queries

q-limited (non-deterministic) search number, $s_q(G)$

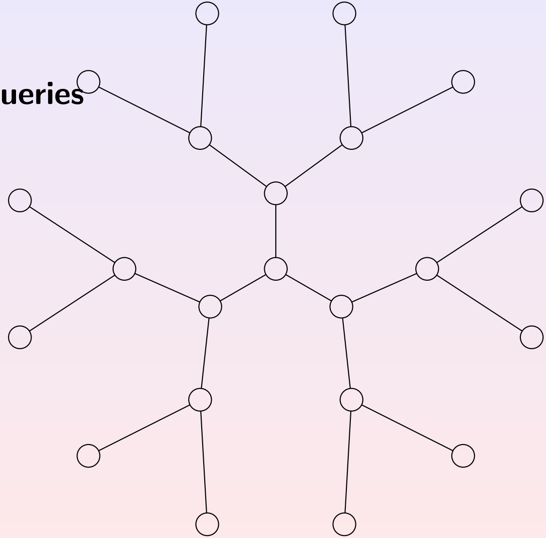
Example with q=2:

$s_0(T)=3$



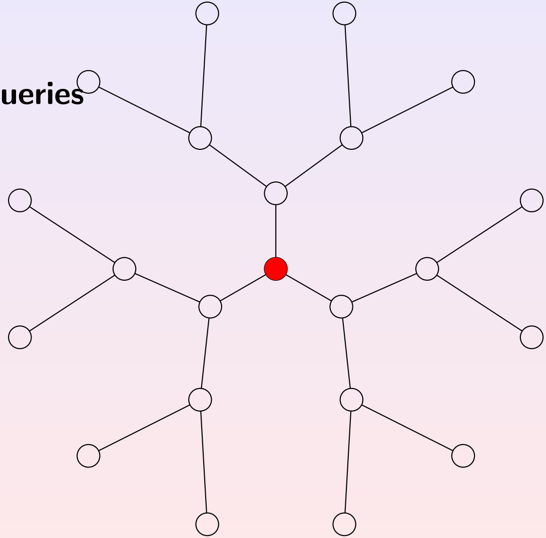
Example with q=2:

2 remaining queries



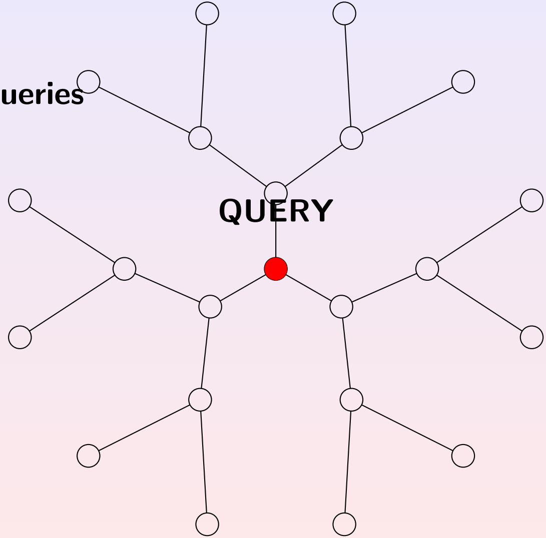
Example with q=2:

2 remaining queries



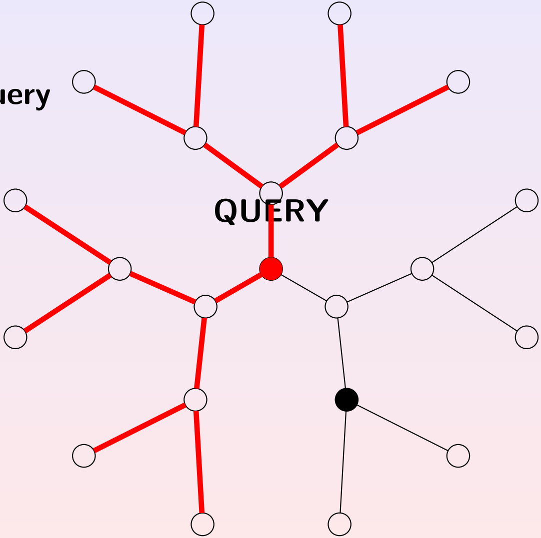
Example with q=2:

2 remaining queries



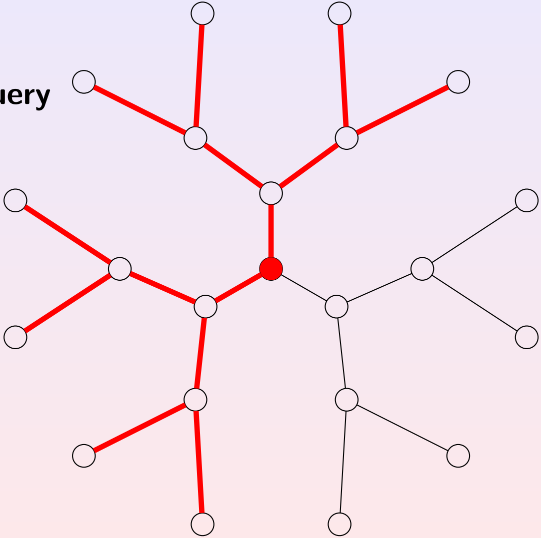
Example with q=2:

1 remaining query



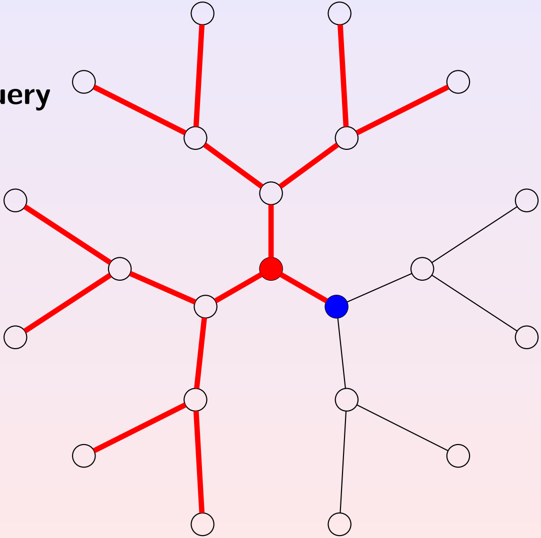
Example with q=2:

1 remaining query



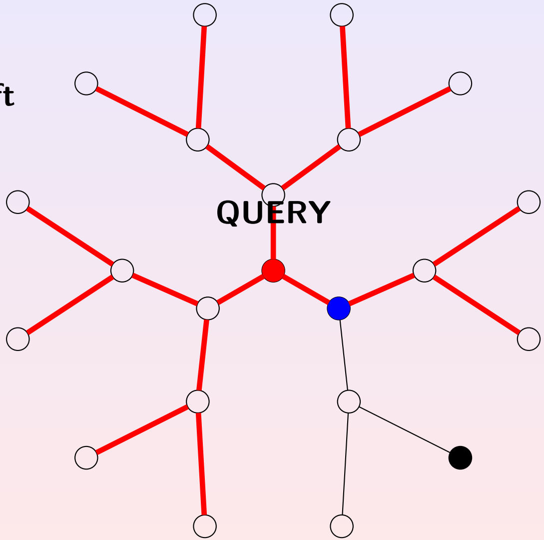
Example with q=2:

1 remaining query



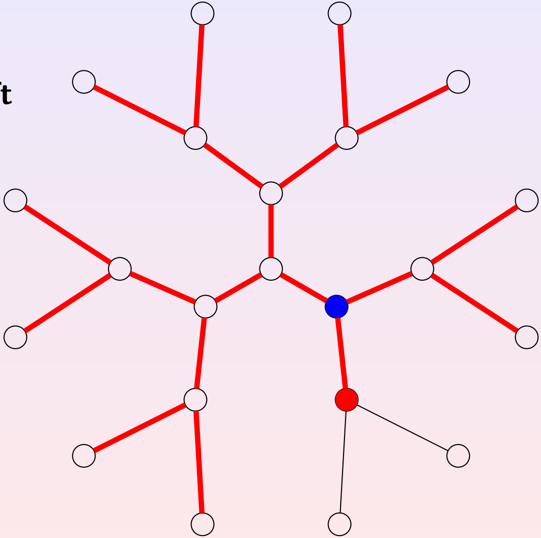
Example with q=2:

no query left



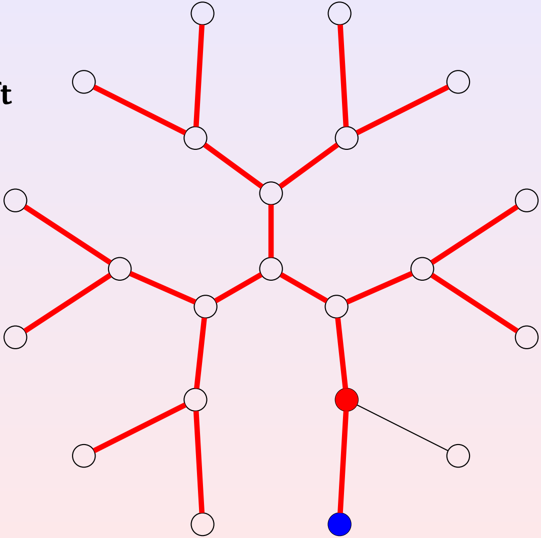
Example with q=2:

no query left



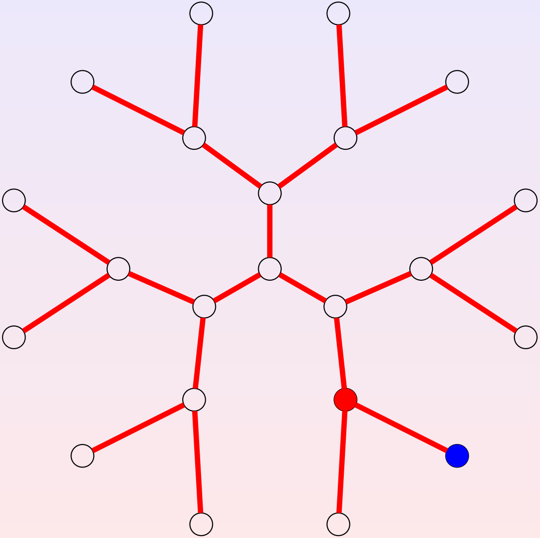
Example with q=2:

no query left

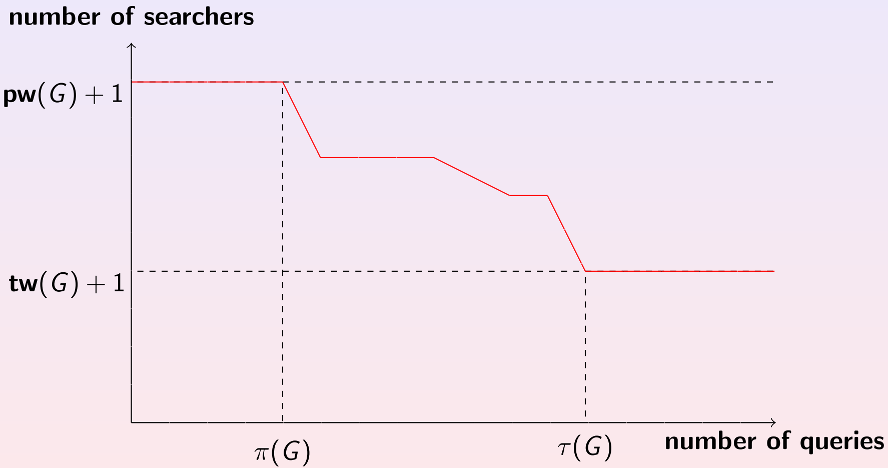


Example with q=2:

$s_2(T)=2$



Controlled Amount of Nondeterminism



Results

In collaboration with F. Mazoit

For any $q \geq 0$, recontamination does not help to catch a fugitive in G performing at most q queries.

- Constructive proof;
- Generalize the existing proofs ($q = 0$ and $q = \infty$).

In collaboration with F.V. Fomin and P. Fraigniaud

- Equivalence between non-deterministic graph searching and branched tree-decomposition;
- Exponential exact algorithm computing $s_q(G)$ in time $O^*(2^n)$;
- $s_q(G) \leq 2\, s_{q+1}(G)$ (almost tight).

Outline

Connected Graph Searching

Limits of the Parson's model

- Searchers cannot move at will in a real network;
- It would be better to let searchers be grouped.

Connected Search Strategy, Barrière *et. al.*, [SPAA 02]

At any step, the cleared part of the graph must induced a connected subgraph.
Let $cs(G)$ be the connected search number of the graph G.

Two main questions

What is the cost of connectivity? ratio cs/s?
Monotonicity property of connected graph searching?

Connected Graph Searching

Limits of the Parson's model

- Searchers cannot move at will in a real network;
- It would be better to let searchers be grouped.

Connected Search Strategy, Barrière *et. al.*, [SPAA 02]

At any step, the cleared part of the graph must induced a connected subgraph.
Let $cs(G)$ be the connected search number of the graph G.

Two main questions

What is the cost of connectivity? ratio cs/s?
Monotonicity property of connected graph searching?

Connected Graph Searching

Limits of the Parson's model

- Searchers cannot move at will in a real network;
- It would be better to let searchers be grouped.

Connected Search Strategy, Barrière *et. al.*, [SPAA 02]

At any step, the cleared part of the graph must induced a connected subgraph.
Let $cs(G)$ be the connected search number of the graph G.

Two main questions

What is the cost of connectivity? ratio cs/s?
Monotonicity property of connected graph searching?

The cost of connectedness

In terms of number of searchers

For any tree T, $s(T) \leq cs(T) \leq 2\ s(T) - 2$. (tight)
Barrière, Flocchini, Fraigniaud, and Thilikos [WG 03]

For any connected graph G, $cs(G) \leq s(G)\ (2 + \log|E(G)|)$.
Fomin, Fraigniaud, and Thilikos 04

About monotonicity

Recontamination does not help in trees.
Barrière, Flocchini, Fraigniaud, and Santoro [SPAA 02]

Recontamination helps in general.
Alspach, Dyer, and Yang [ISAAC 04]

Results: Case of a invisible fugitive

In collaboration with P. Fraigniaud

For any n-nodes connected graph G, $cs(G)/s(G) \leq \log n$.

Graphs with bounded chordality k

(T,X) an optimal tree-decomposition of G
$cs(G) \leq (tw(G)\lfloor k/2 \rfloor + 1)cs(T)$.

$\Rightarrow cs(G)/s(G) \leq 2\,(tw(G)+1)$ if G chordal

Results: Case of a visible fugitive

In collaboration with P. Fraigniaud

For any n-nodes graph G, $cvs(G)/vs(G) \leq \log n$ (tight for monotone strategies).

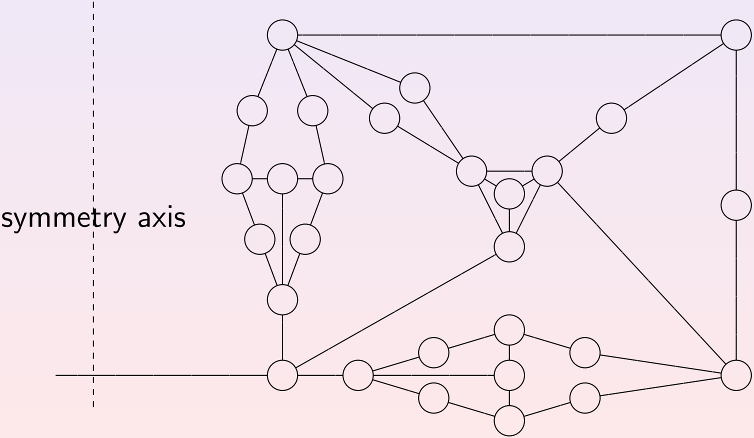
In visible connected graph searching, recontamination helps

- For any $k \geq 4$, there exists a graph G such that $\mathbf{cvs}(G) = 4k+1$ and any monotone connected visible search strategy uses at least $4k+2$ searchers.

Non-monotonicity

Recontamination helps in visible connected graph searching

Let G be the graph below: $mcvs(G) > cvs(G) = 4$.



Outline

Graph searching in a distributed way

Distributed search problem

To design a *distributed protocol* that enables the *minimum number* of searchers to clear the network.
The searchers must compute themselves a strategy.

In this part, we consider connected search strategies.

mcs refers to the smallest number of searchers required to catch an invisible fugitive in a monotone connected way.

Graph searching in a distributed way

Distributed search problem

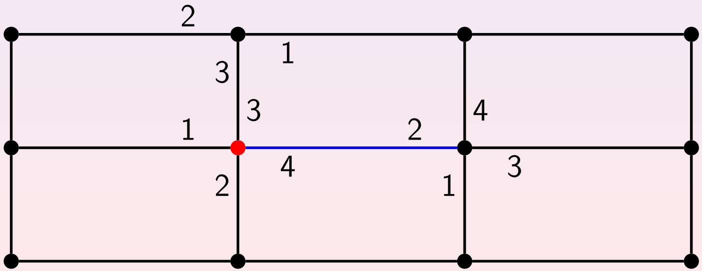
To design a *distributed protocol* that enables the *minimum number* of searchers to clear the network.
The searchers must compute themselves a strategy.

In this part, we consider connected search strategies.

mcs refers to the smallest number of searchers required to catch an invisible fugitive in a monotone connected way.

Distributed graph searching: Environment

- undirected connected graph;
- local orientation of the edges;
- whiteboards on vertices;
- synchronous/asynchronous environment.



Distributed graph searching: the searchers

- autonomous mobile computing entities with distinct IDs;
- automata with $O(\log n)$ bits of memory.

Decision is computed locally and depends on:

- its current state;
- the states of the other searchers present at the vertex;
- the content of the local whiteboard;
- if appropriate the incoming port number.

A searcher can decide to:

- leave a vertex via a specific port number;
- switch its state.
- write/erase content of the local whiteboard.

Distributed graph searching, related work

The searchers have a prior knowledge of the topology.

Protocols to clear specific topologies

- **Tree**. Barrière *et. al.*, [SPAA 02]
- **Mesh**. Flocchini, Luccio, and Song. [CIC 05]
- **Hypercube**. Flocchini, Huang, and Luccio. [IPDPS 05]
- **Tori**. Flocchini, Luccio, and Song. [IPDPS 06]
- **Siperski's graph**. Luccio. [FUN 07]

A monotone connected and optimal strategy is performed.

Remark:

Compared with the synchronous case, an additional searcher may be necessary and is sufficient in an asynchronous network to clear a graph in a monotone connected way [CIC 05].

Distributed graph searching, related work

The searchers have a prior knowledge of the topology.

Protocols to clear specific topologies

- **Tree**. Barrière *et. al.*, [SPAA 02]
- **Mesh**. Flocchini, Luccio, and Song. [CIC 05]
- **Hypercube**. Flocchini, Huang, and Luccio. [IPDPS 05]
- **Tori**. Flocchini, Luccio, and Song. [IPDPS 06]
- **Siperski's graph**. Luccio. [FUN 07]

A monotone connected and optimal strategy is performed.

Remark:

Compared with the synchronous case, an additional searcher may be necessary and is sufficient in an asynchronous network to clear a graph in a monotone connected way [CIC 05].

Results

In collaboration with L. Blin, P. Fraigniaud and S. Vial

Distributed protocol that enable $mcs(G)$ searchers to clear an unknown graph G in a connected way

(Automaton: $O(\log n)$ bits of memory, whiteboards'size: $O(m \log n)$ bits).

Problems: the strategy is not monotone and may be performed in expentional time.

In collaboration with D. Soguet

No distributed protocol enables $mcs(G)$ searchers to clear an **unknown** graph G in a **monotone** connected way.

$\Theta(n \log n)$ bits of information must be provided to the searchers

Results

In collaboration with L. Blin, P. Fraigniaud and S. Vial

Distributed protocol that enable $mcs(G)$ searchers to clear an unknown graph G in a connected way

(Automaton: $O(\log n)$ bits of memory, whiteboards'size: $O(m \log n)$ bits).

Problems: the strategy is not monotone and may be performed in expentional time.

In collaboration with D. Soguet

No distributed protocol enables $mcs(G)$ searchers to clear an **unknown** graph G in a **monotone** connected way.

$\Theta(n \log n)$ bits of information must be provided to the searchers

Results

In collaboration with L. Blin, P. Fraigniaud and S. Vial

Distributed protocol that enable $mcs(G)$ searchers to clear an unknown graph G in a connected way

(Automaton: $O(\log n)$ bits of memory, whiteboards'size: $O(m \log n)$ bits).

Problems: the strategy is not monotone and may be performed in expentional time.

In collaboration with D. Soguet

No distributed protocol enables $mcs(G)$ searchers to clear an **unknown** graph G in a **monotone** connected way.

$\Theta(n \log n)$ bits of information must be provided to the searchers

Advice, size of advice [Fraigniaud *et al.* 06]

A distributed problem $\mathcal{P}$

Instance of $\mathcal{P}$ (for example a graph G)

Advice: information that can be used to solve $\mathcal{P}$ *efficiently*

Information is modelized by

- An oracle $\mathcal{O}$ that assigns at any instance G a string of bits $\mathcal{O}(G)$ that is distributed on the vertices of G.
- size of advice $|\mathcal{O}(G)|$

Examples

- wake-up (linear number of messages): $\Theta(n \log n)$ bits;
- broadcast (linear number of messages): $O(n)$ bits;
- tree exploration, MST, graph coloring ...

Advice, size of advice [Fraigniaud *et al.* 06]

A distributed problem $\mathcal{P}$
Instance of $\mathcal{P}$ (for example a graph G)
Advice: information that can be used to solve $\mathcal{P}$ *efficiently*

Information is modelized by

- An oracle $\mathcal{O}$ that assigns at any instance G a string of bits $\mathcal{O}(G)$ that is distributed on the vertices of G.
- size of advice $|\mathcal{O}(G)|$

Examples

- wake-up (linear number of messages): $\Theta(n \log n)$ bits;
- broadcast (linear number of messages): $O(n)$ bits;
- tree exploration, MST, graph coloring ...

Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

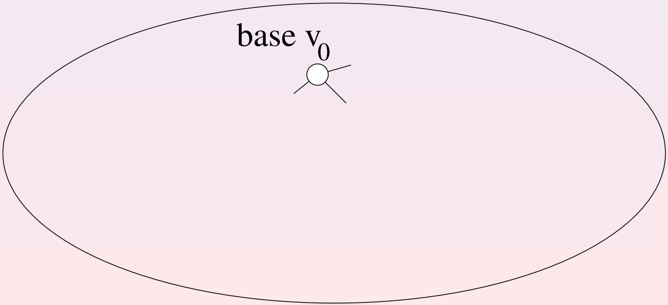
$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

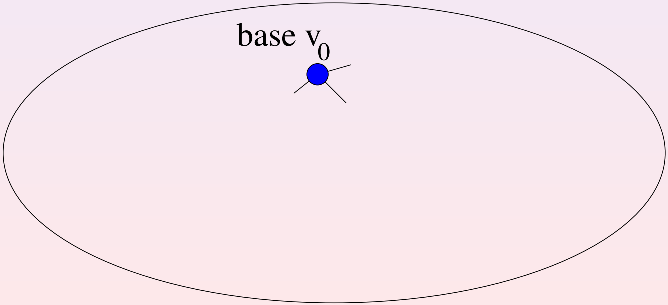


Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \rightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

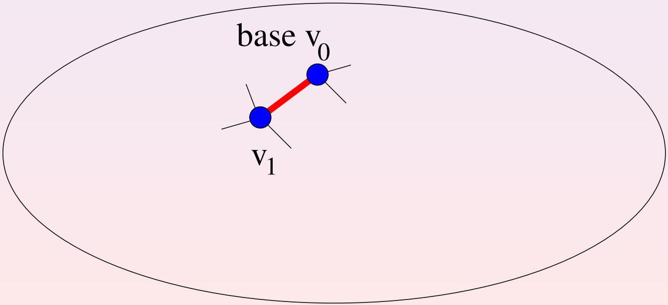


Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

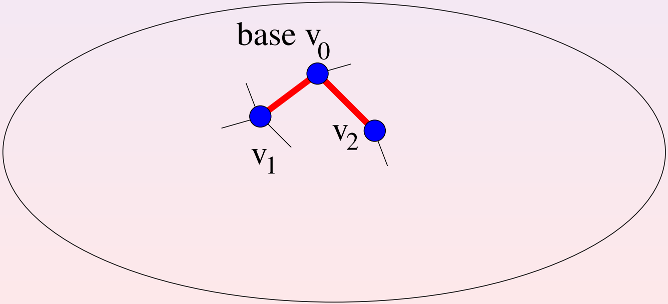


Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

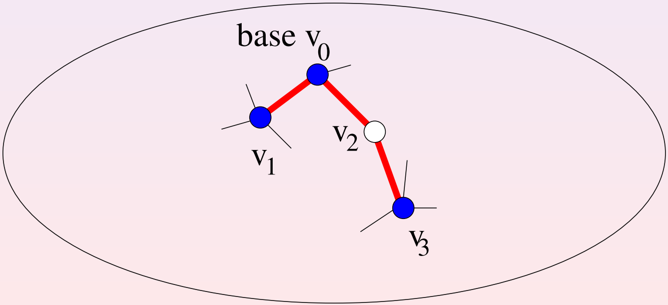


Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

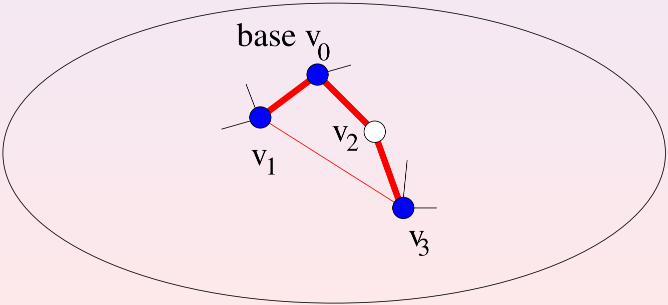


Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

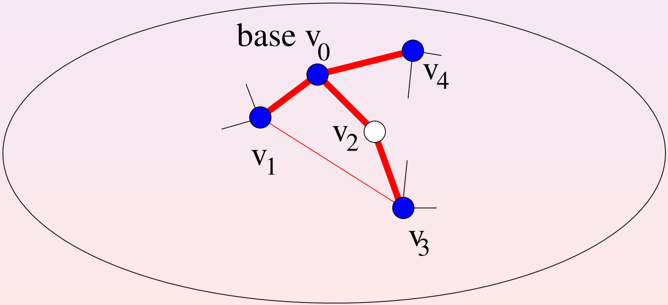


Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

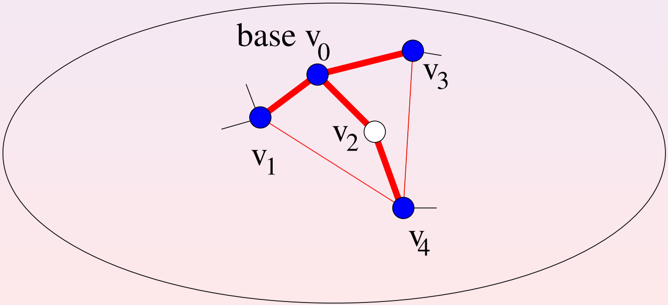


Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \longrightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.

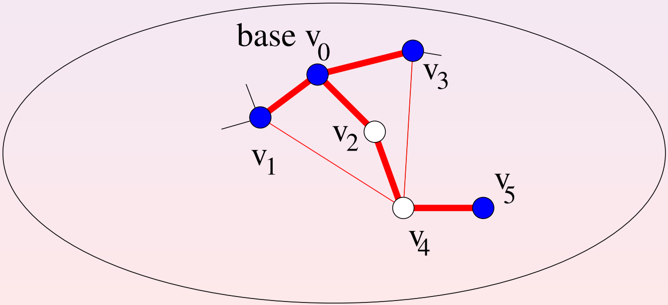


Idea of the upper bound: $O(n \log n)$

Let G be a graph, and $v_0 \in V(G)$

Let S be a monotone connected and optimal strategy for G.
Our oracle "encodes" S on the vertices of G.

$S \rightarrow$ a vertex-ordering $\{v_0, v_1, \cdots, v_{n-1}\}$,
and n trees $T_0 \subset \cdots \subset T_{n-1}$ such that T_i spans $\{v_0, \cdots, v_i\}$.



Idea of the upper bound: the Oracle

Our protocol is divided in **n+1 phases**.
Any vertex v_i, 3 types of edges:

1. the edges of the spanning tree T_n
2. the edge by which the searcher will leave v_i;
3. the others.

Moreover the oracle provides 2 phase numbers: **The** phase when the edges of type 3 can be cleared and **the** phase when v_i can be left.

Size of advice: coding a spanning tree + 2 phase numbers for any vertex = $O(n \log n)$ bits of advice.

Idea of the upper bound: the Protocol

Phase i of the protocol ($0 \leq i \leq n$):

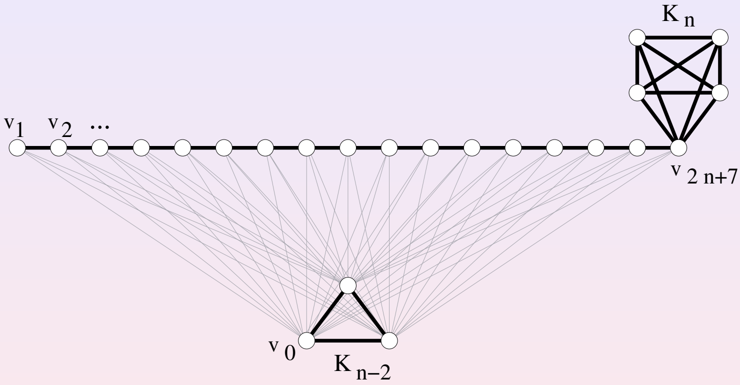
At the beginning of the phase i:

- T_i + some edges between the vertices $\{v_0, \cdots, v_i\}$ are cleared.
- Any vertex of $\{v_0, \cdots, v_i\}$ is guarded by a searcher if it is incident to a contaminated edge.

Idea of the protocol:

1. Any free searcher performs a DFS of T_i.
2. If it meets a vertex such that the phase to clear the edges of type 3 is i, then it clears these edges.
3. At the end of the phase, the edge that enables to move on v_{i+1} is discovered and cleared.

The lower bound: $\Omega(n \log n)$



Class of graphs $(G_n)_{n \in N}$ (The figure corresponds to G_5).
All the monotone connected and optimal strategies in this class are **strongly constrained**.

Outline

1. Introduction
2. Non-deterministic Graph Searching
3. Connected Graph Searching
4. Distributed Graph Searching
5. Conclusion and Further Works

Summary of the results

Non-deterministic graph searching

A unified approach of visible and invisible graph searching
Unified proof of monotonicity.

Connected graph searching

Upper bounds for the ratio cs/s
Case of a visible fugitive

Distributed graph searching

Distributed protocol to clear an unknown graph
Amount of information required for monotonicity

Open Problems

Non-deterministic graph searching

FPT Algorithm?
Polynomial-time algorithm in trees?

Connected graph searching

cs/s ? FPT Algorithm?
NP-membership?

Distributed graph searching

Tradeoff between amount of information and number of searchers?

Ad'

IMAGINE: First International workshop on Mobility, Algorithms and Graph theory In dynamic NEtworks.

Collocated with DISC 2007 in Cyprus (the day after)

http://www.lifl.fr/IMAGINE2007