Graph Searching and related problems.

Graph Minors, connected search strategies, and distributed approach

Nicolas Nisse

DIM, Universidad de Chile, Project Anillo en Redes.

October the 12th 2007

Outline

General problem

Context

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

Goal

To design a strategy that capture **any** fugitive using the **fewest searchers as possible**.

Initially studied by Breish (67) and Parson (76)

To save a speleologist lost in a caves'network.
Equivalently, to clear a contaminated pipelines'network.

General problem

Context

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

Goal

To design a strategy that capture **any** fugitive using the **fewest searchers as possible**.

Initially studied by Breish (67) and Parson (76)

To save a speleologist lost in a caves'network.
Equivalently, to clear a contaminated pipelines'network.

Motivations

Layout Problems

Numerical analysis, VLSI design, etc.
Related to: bandwidth, cutwidth, profile, etc.

Computational Complexity

Relationship between graph searching and pebble games [Kirousis and Papadimitriou 86]
Tradeoff space/time complexity of computation.

Graph Minors Theory

Robertson and Seymour [JCTB, 1983-]
Tree decomposition, Path decomposition

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

The fugitive

- occupies vertices of the graph;
- is arbitrary fast;
- can move from one vertex to another by following a path in G, as long as it does not cross any vertex occupied by a searcher.

The searchers

- can jump from one vertex to any other vertex of the graph;
- are visible.

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

The fugitive

- occupies vertices of the graph;
- is arbitrary fast;
- can move from one vertex to another by following a path in G, as long as it does not cross any vertex occupied by a searcher.

The searchers

- can jump from one vertex to any other vertex of the graph;
- are visible.

Search Strategy, Parson. [GTC,1978] Model 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.

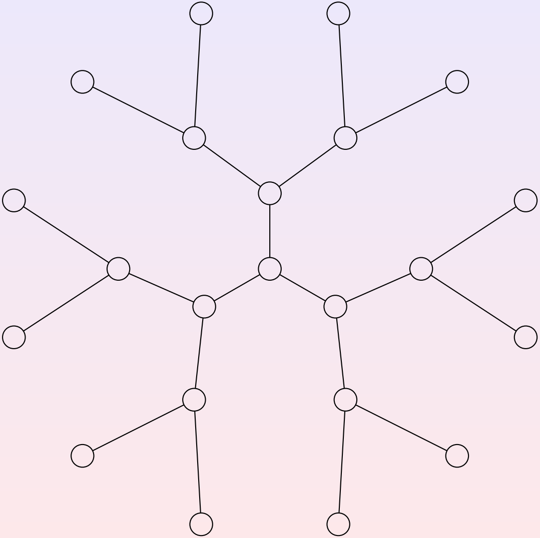
. . . that must result in catching the fugitive

The fugitive is caugth when it occupies the same vertex as a searcher and it cannot move away.

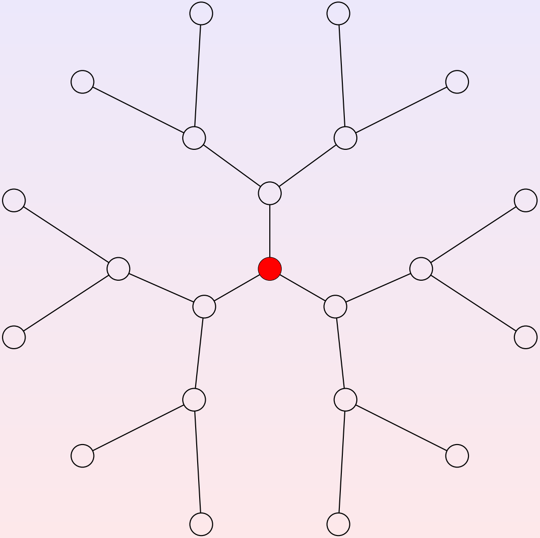
The node-search number

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

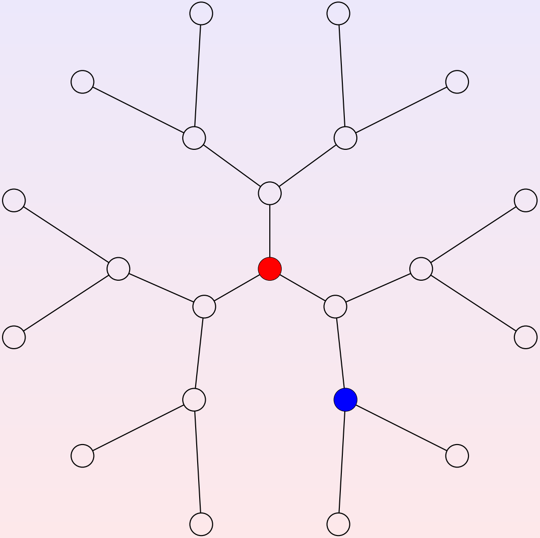
Simple example: A ternary tree



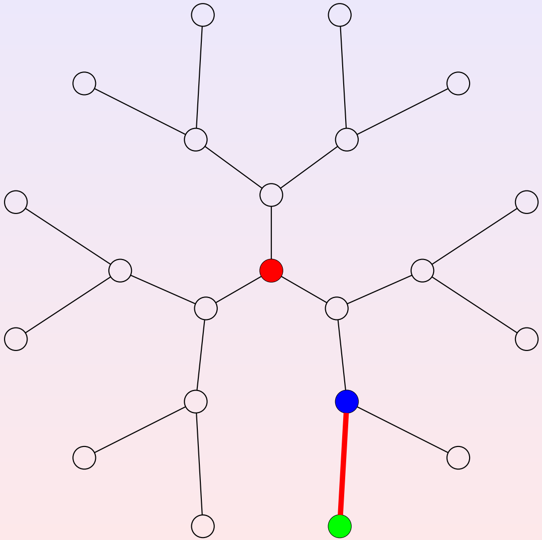
Simple example: A ternary tree



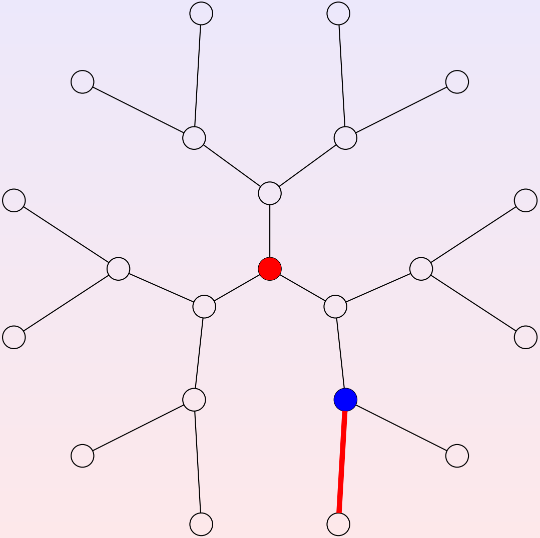
Simple example: A ternary tree



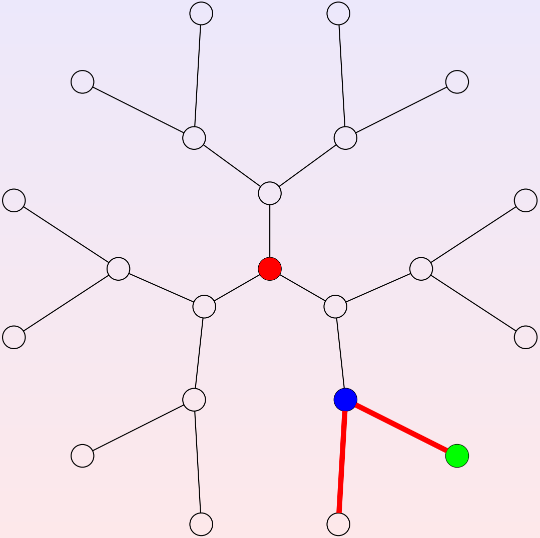
Simple example: A ternary tree



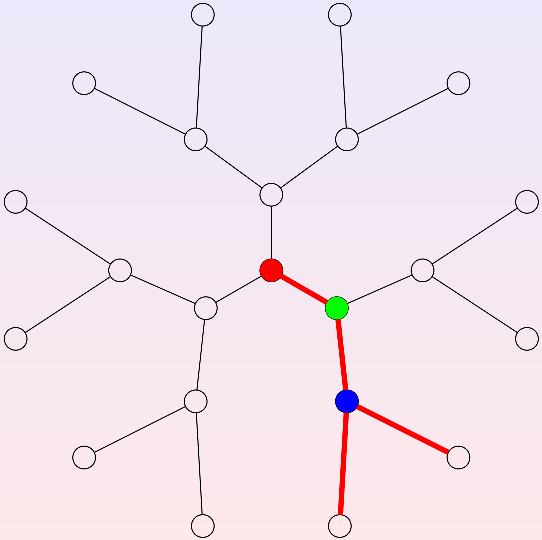
Simple example: A ternary tree



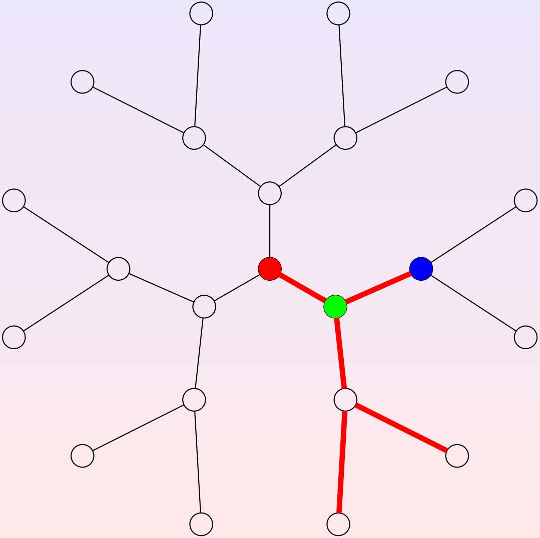
Simple example: A ternary tree



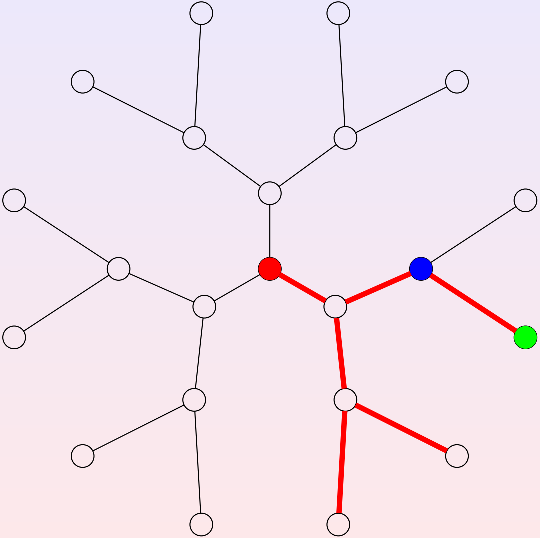
Simple example: A ternary tree



Simple example: A ternary tree

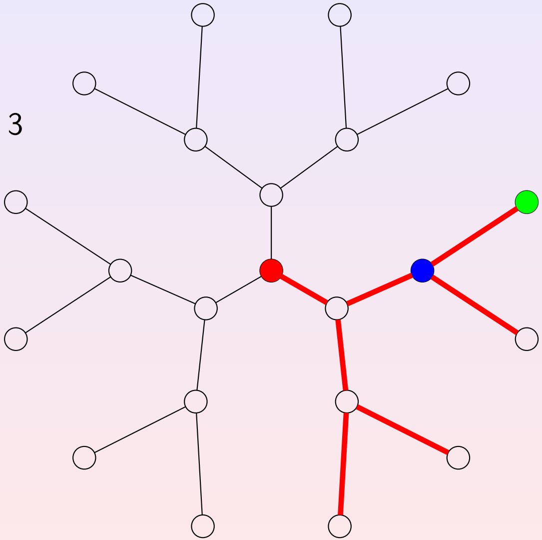


Simple example: A ternary tree



Simple example: A ternary tree

$\mathbf{s}(T) = 3$



Visibility of the fugitive

Visible fugitive

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

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

Invisible fugitive vs. visible fugitive in trees

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

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

Visibility of the fugitive

Visible fugitive

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

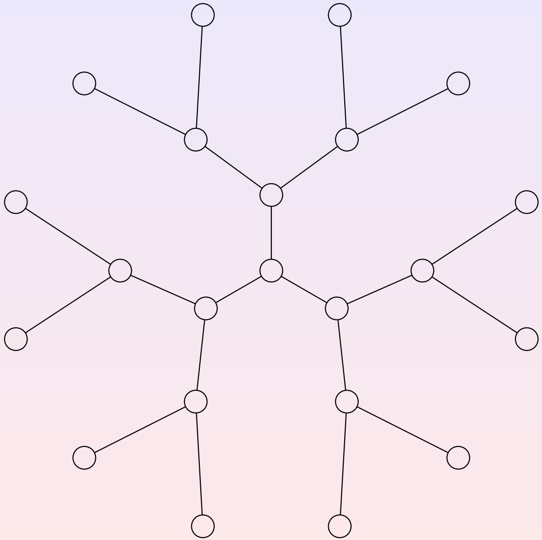
Obviously, for any graph G, $\mathbf{vs}(G) \leq \mathbf{s}(G)$.

Invisible fugitive vs. visible fugitive in trees

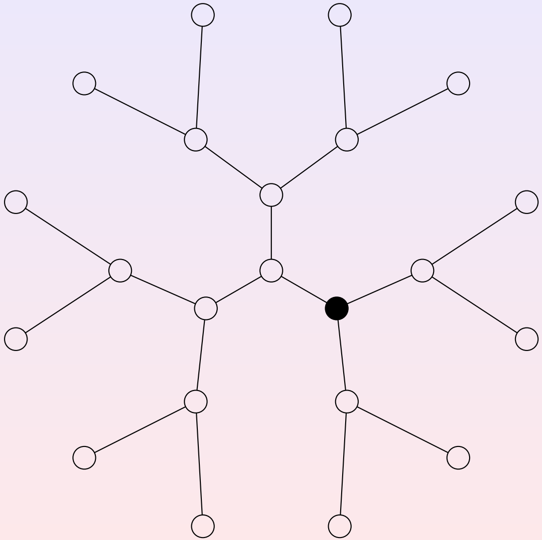
For any n-nodes tree T, $\mathbf{s}(T) \leq 1 + \log_3(n-1)$ (tight)
Megiddo *et al.* [JACM 88]

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

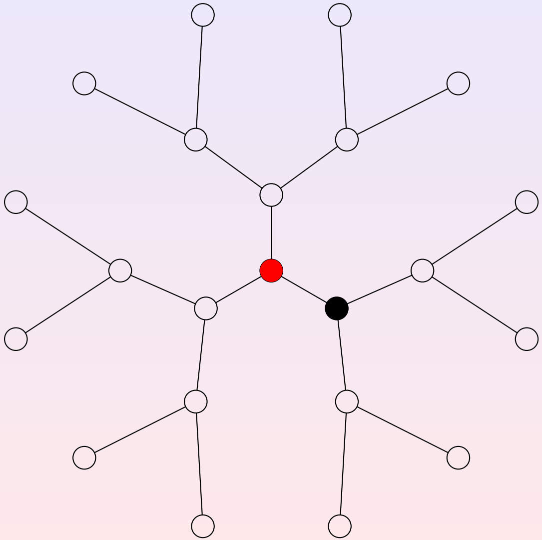
Visible graph searching in a tree



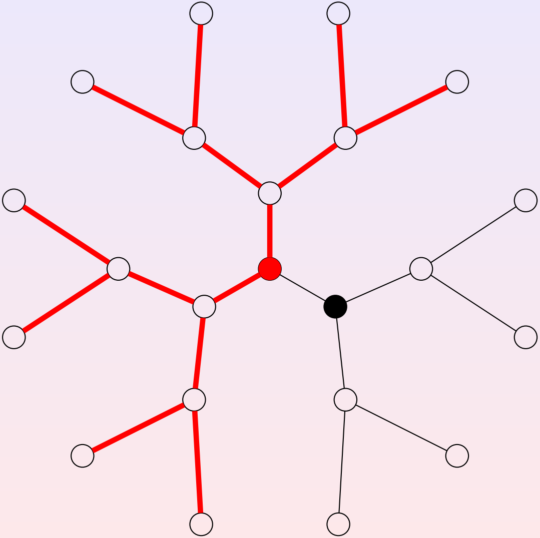
Visible graph searching in a tree



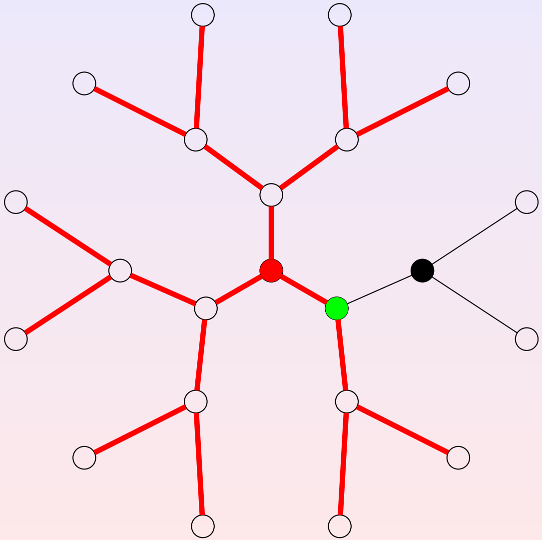
Visible graph searching in a tree



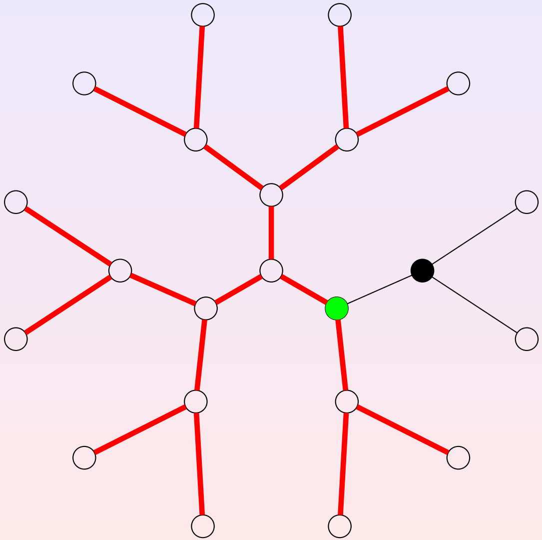
Visible graph searching in a tree



Visible graph searching in a tree

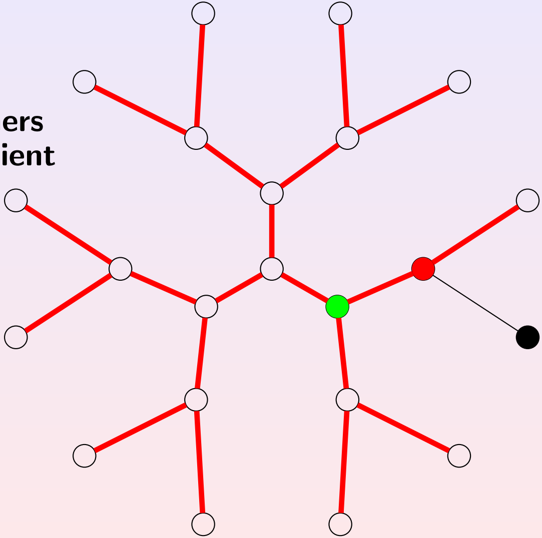


Visible graph searching in a tree



Visible graph searching in a tree

2 searchers are sufficient



NP-hardness

The following problems are NP-hard

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

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

NP-membership? Certificate?

A strategy is a certificate, but what is its size ?

NP-hardness

The following problems are NP-hard

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

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

NP-membership? Certificate?
A strategy is a certificate, but what is its size ?

Monotonicity

Monotone strategies

A vertex v is recontaminated if the fugitive can move to v after v has been occupied by a searcher.
A search strategy is monotone if no recontamination ever occurs. That is, a vertex is occupied by a searcher only once.

A monotone strategy consists of a polynomial number of steps

Question: Does recontamination help?

In other words, for any graph, does there always exist a monotone strategy using the smallest number of searchers?

Monotonicity

Monotone strategies

A vertex v is recontaminated if the fugitive can move to v after v has been occupied by a searcher.
A search strategy is monotone if no recontamination ever occurs. That is, a vertex is occupied by a searcher only once.

A monotone strategy consists of a polynomial number of steps

Question: Does recontamination help?

In other words, for any graph, does there always exist a monotone strategy using the smallest number of searchers?

Monotonicity

Monotone strategies

A vertex v is recontaminated if the fugitive can move to v after v has been occupied by a searcher.
A search strategy is monotone if no recontamination ever occurs. That is, a vertex is occupied by a searcher only once.

A monotone strategy consists of a polynomial number of steps

Question: Does recontamination help?

In other words, for any graph, does there always exist a monotone strategy using the smallest number of searchers?

Recontamination does not help

Case of an invisible fugitive

- **Bienstock and Seymour**, J.of Alg., 1991
  Monotonicity in graph searching.
- **LaPaugh**, J.of ACM, 1993
  Recontamination does not help to search a graph.

Constructive proofs: Any strategy using k searchers can be turned into a *monotone* strategy using $\leq k$ searchers.

Case of a visible fugitive

- **Seymour and Thomas**, J. of Comb. Th., 1993.
  Graph searching and a min-max theorem for tree-width

Non constructive proof.

Corollary: To compute $\mathbf{s}(G)$ (resp., $\mathbf{vs}(G)$) is NP-complete.

Recontamination does not help

Case of an invisible fugitive

- **Bienstock and Seymour**, J.of Alg., 1991
  Monotonicity in graph searching.
- **LaPaugh**, J.of ACM, 1993
  Recontamination does not help to search a graph.

Constructive proofs: Any strategy using k searchers can be turned into a *monotone* strategy using $\leq k$ searchers.

Case of a visible fugitive

- **Seymour and Thomas**, J. of Comb. Th., 1993.
  Graph searching and a min-max theorem for tree-width

Non constructive proof.

Corollary: To compute $\mathbf{s}(G)$ (resp., $\mathbf{vs}(G)$) is NP-complete.

Recontamination does not help

Case of an invisible fugitive

- **Bienstock and Seymour**, J.of Alg., 1991
  Monotonicity in graph searching.
- **LaPaugh**, J.of ACM, 1993
  Recontamination does not help to search a graph.

Constructive proofs: Any strategy using k searchers can be turned into a *monotone* strategy using $\leq k$ searchers.

Case of a visible fugitive

- **Seymour and Thomas**, J. of Comb. Th., 1993.
  Graph searching and a min-max theorem for tree-width

Non constructive proof.

Corollary: To compute $\mathbf{s}(G)$ (resp., $\mathbf{vs}(G)$) is NP-complete.

Search numbers and graphs'decompositions

Thanks to the monotonicity, we get:

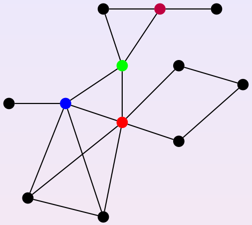
Search number and Pathwidth (**pw**)

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

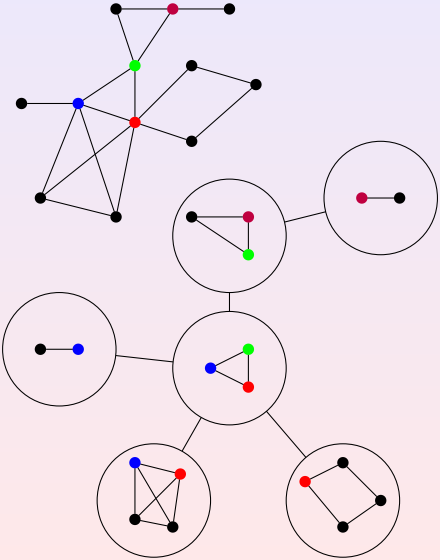
Visible search number and Treewidth (**tw**)

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

Tree and Path Decompositions

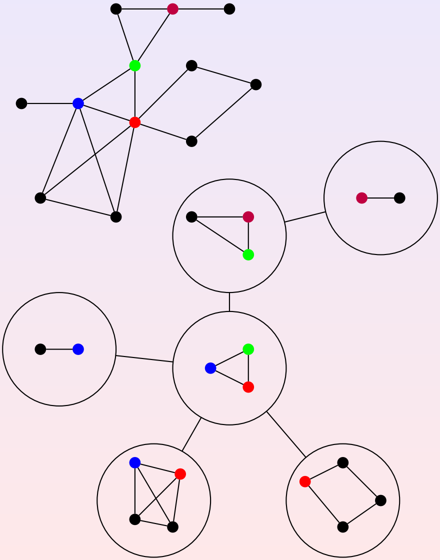


Tree and Path Decompositions



a tree T and bags $(X_t)_{t \in V(T)}$

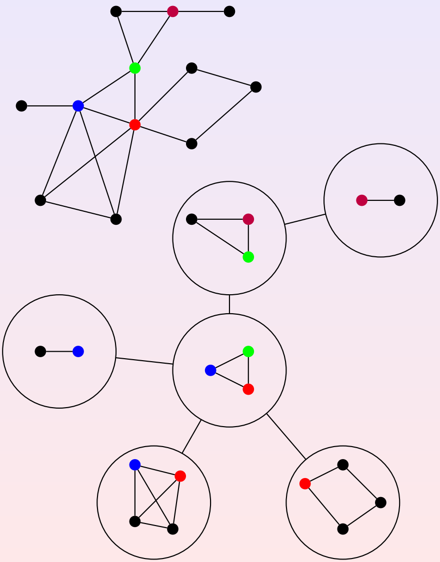
Tree and Path Decompositions



a tree T and bags $(X_t)_{t \in V(T)}$

- every vertex of G is at least in one bag;

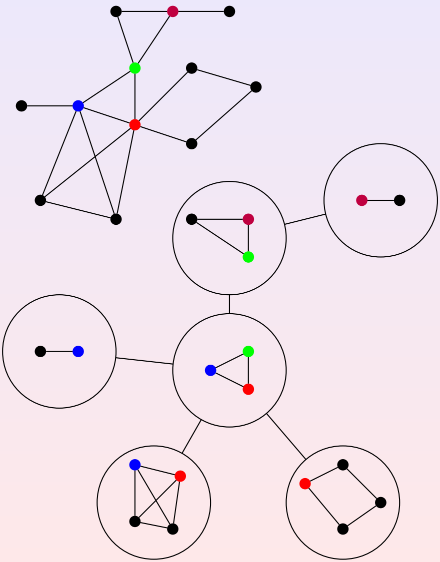
Tree and Path Decompositions



a tree T and bags $(X_t)_{t \in V(T)}$

- every **vertex** of G is at least in one bag;
- both ends of an edge of G are at least in one bag;

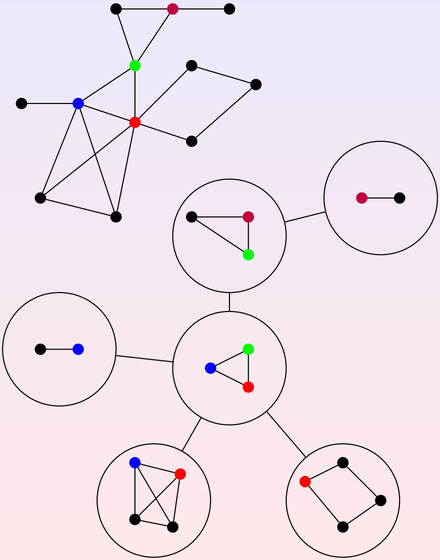
Tree and Path Decompositions



a tree T and bags $(X_t)_{t \in V(T)}$

- every **vertex** of G is at least in one bag;
- both ends of an **edge** of G are at least in one bag;
- For any vertex of G, all bags that contain it, form a subtree.

Tree and Path Decompositions

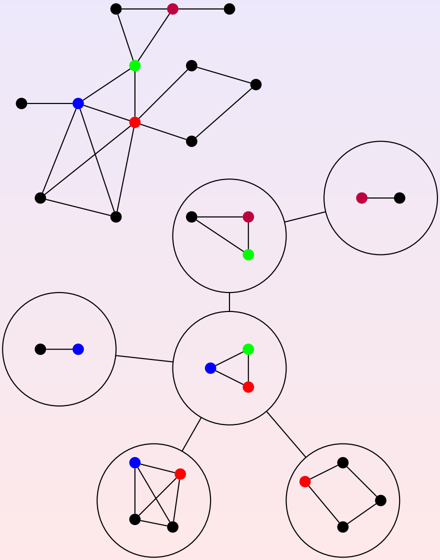


a tree T and bags $(X_t)_{t \in V(T)}$

- every **vertex** of G is at least in one bag;
- both ends of an **edge** of G are at least in one bag;
- For any vertex of G, all bags that contain it, form a **subtree**.

Width = Size of largest Bag -1

Tree and Path Decompositions



a tree T and bags $(X_t)_{t \in V(T)}$

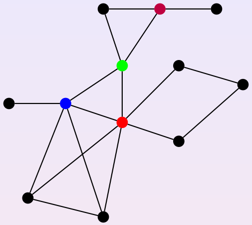
- every **vertex** of G is at least in one bag;
- both ends of an **edge** of G are at least in one bag;
- For any vertex of G, all bags that contain it, form a **subtree**.

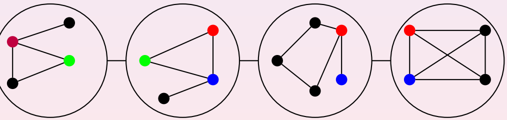
Width = Size of largest Bag -1

treewidth of G

tw(G), minimum width among any tree-decomposition

Tree and Path Decompositions





a path P and bags $(X_t)_{t \in V(P)}$

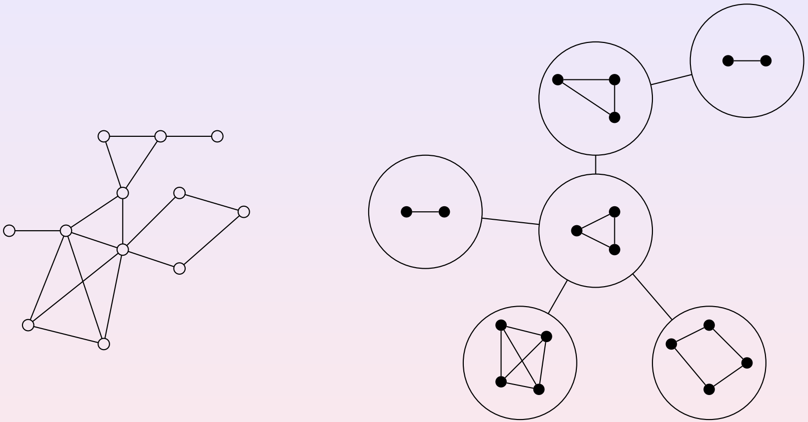
- every **vertex** of G is at least in one bag;
- both ends of an **edge** of G are at least in one bag;
- For any vertex of G, all bags that contain it, form a **subpath**.

Width = Size of largest Bag -1

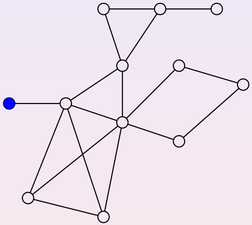
pathwidth of G

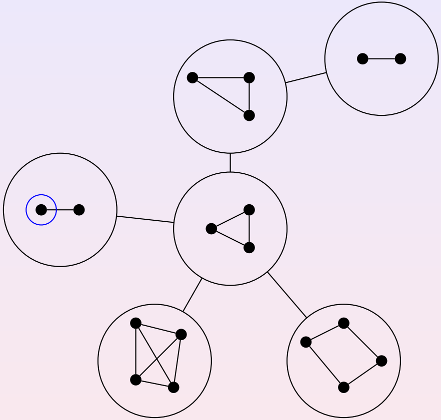
pw(G), minimum width among any path-decomposition

Monotone Visible Search Number $=$ Treewidth $+1$

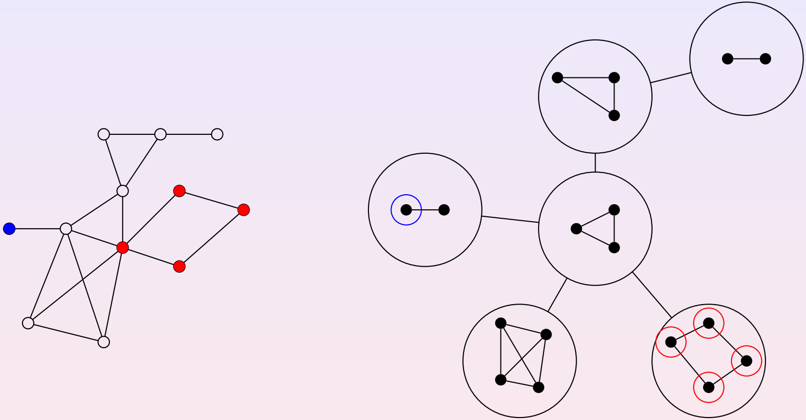


Monotone Visible Search Number = Treewidth +1

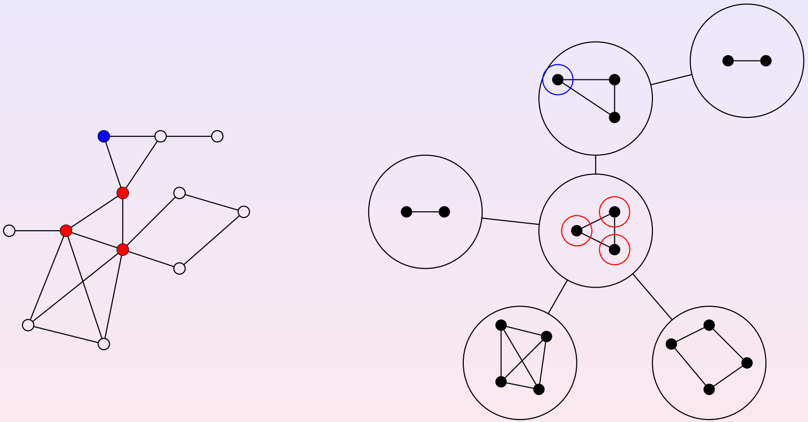




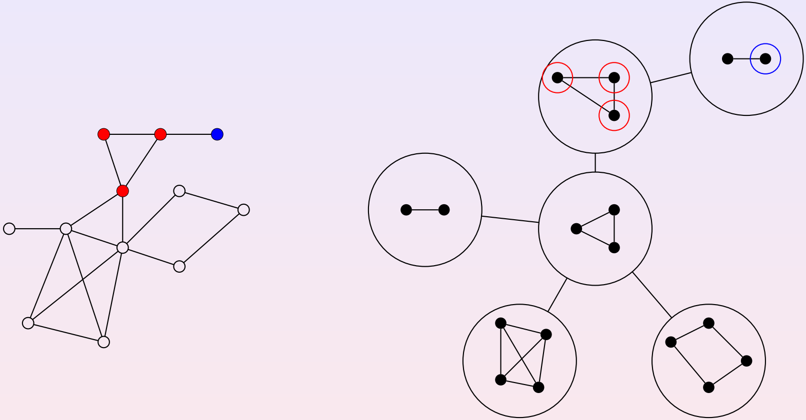
Monotone Visible Search Number = Treewidth +1



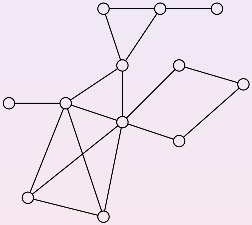
Monotone Visible Search Number = Treewidth +1

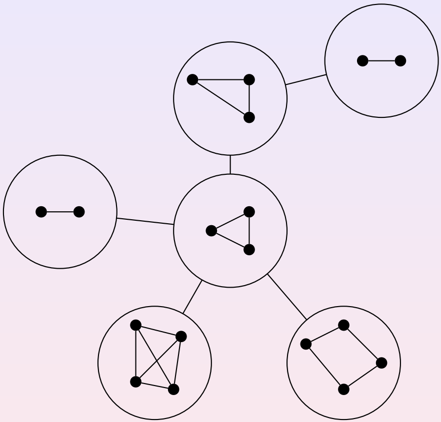


Monotone Visible Search Number = Treewidth +1

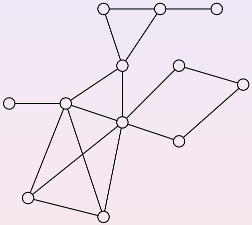


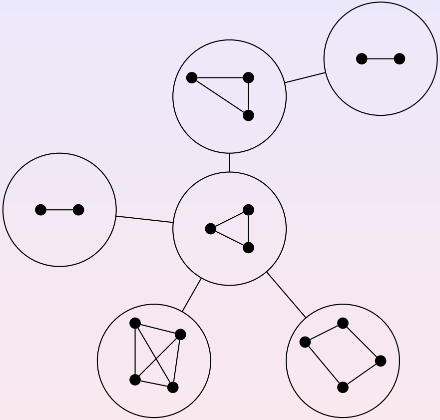
Monotone Visible Search Number = Treewidth +1





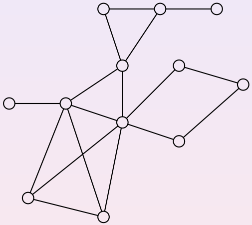
Monotone Visible Search Number = Treewidth +1

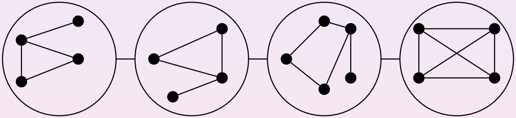




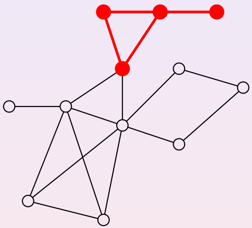
Thus, $\mathbf{vs(G)} \leq \mathbf{tw(G)} + 1$

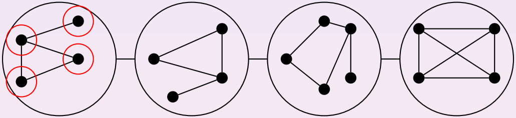
Monotone Search Number = Pathwidth +1



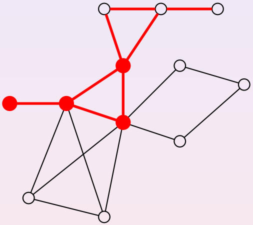


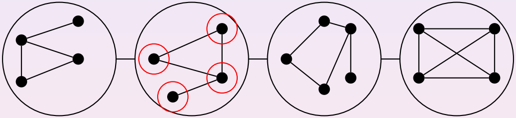
Monotone Search Number = Pathwidth +1



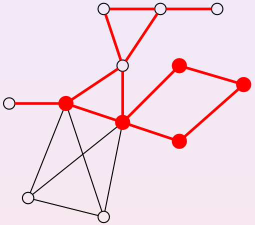


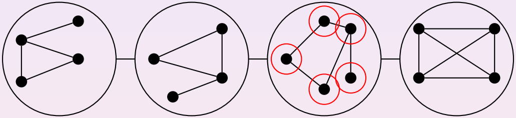
Monotone Search Number = Pathwidth +1





Monotone Search Number = Pathwidth +1





Thus, $\mathbf{s(G)} \leq \mathbf{pw(G)} + \mathbf{1}$

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

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

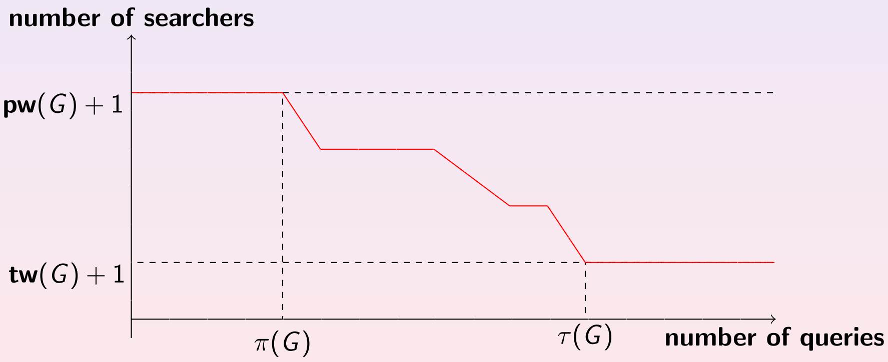
1. Place a searcher at a vertex of the graph;
2. Remove a searcher from a vertex of the graph;
3. Perform a query to the Oracle.

Tradeoff number of searchers / number of queries

Controlled Amount of Nondeterminism

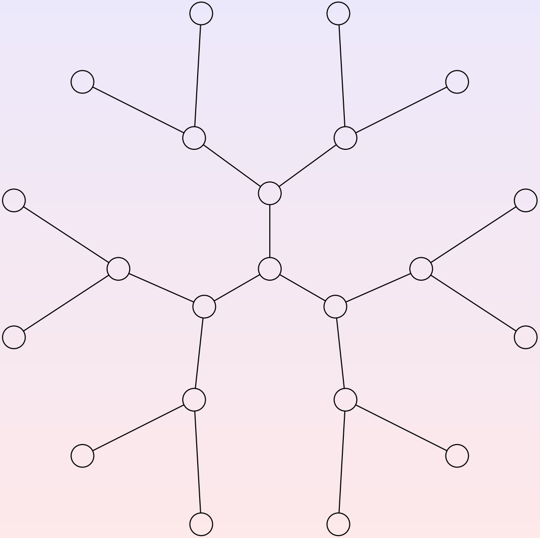
q-limited (non-deterministic) search number, $s_q(G)$

- $\mathbf{s}_0(G) = \mathbf{pw}(G) + 1$, invisible search number of G;
- $\mathbf{s}_\infty(G) = \mathbf{tw}(G) + 1$, visible search number of G.



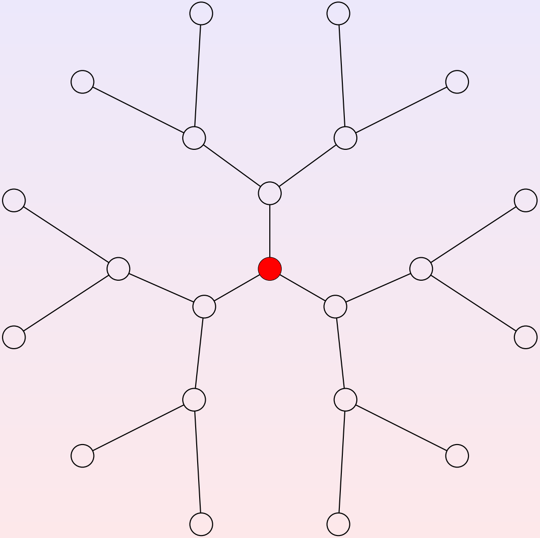
Still the same ternary tree

$s_0(T)=3$



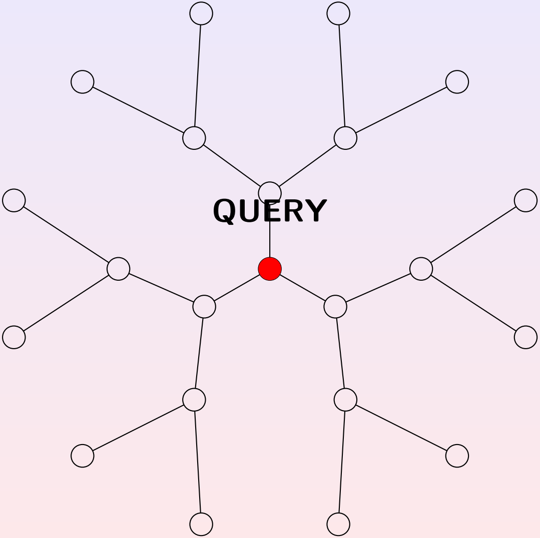
Still the same ternary tree

$s_0(T)=3$



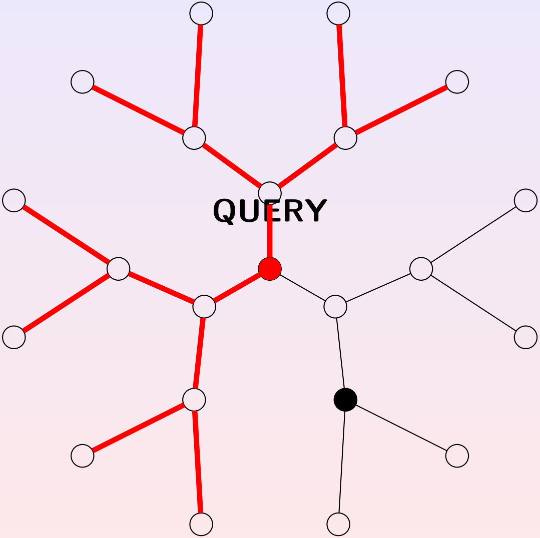
Still the same ternary tree

1 query



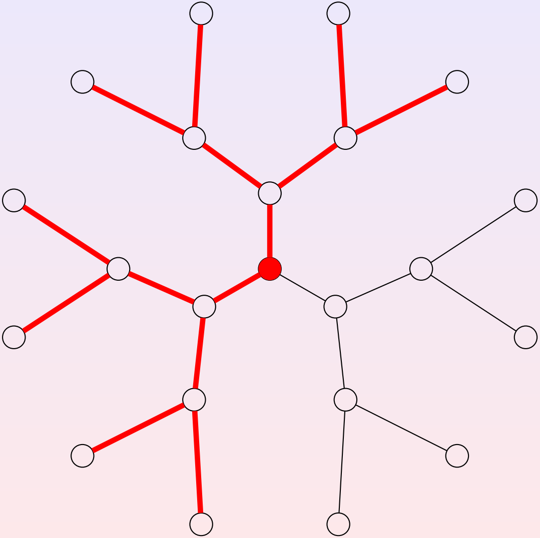
Still the same ternary tree

1 query



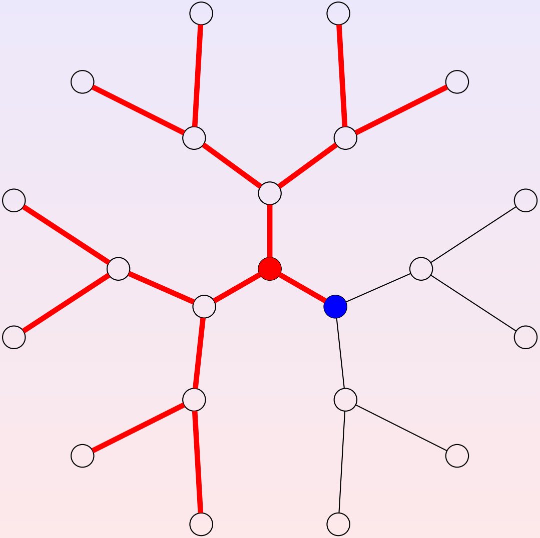
Still the same ternary tree

1 query



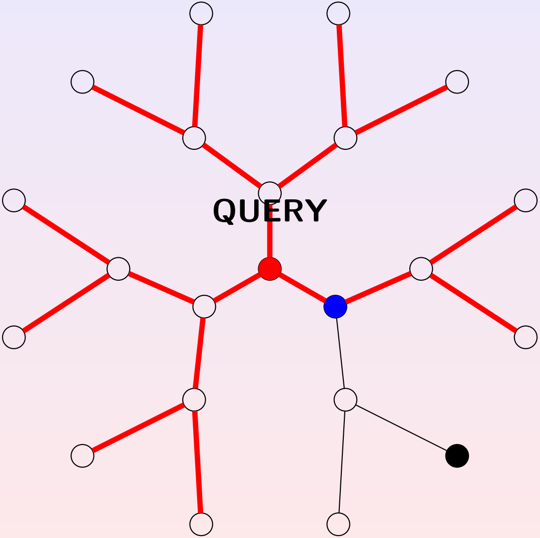
Still the same ternary tree

1 query



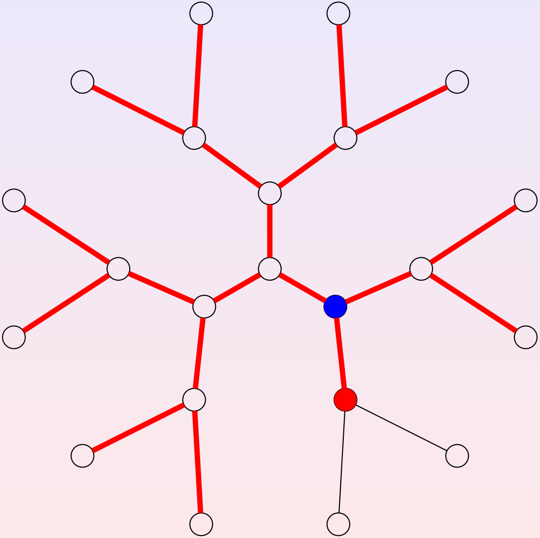
Still the same ternary tree

2 queries



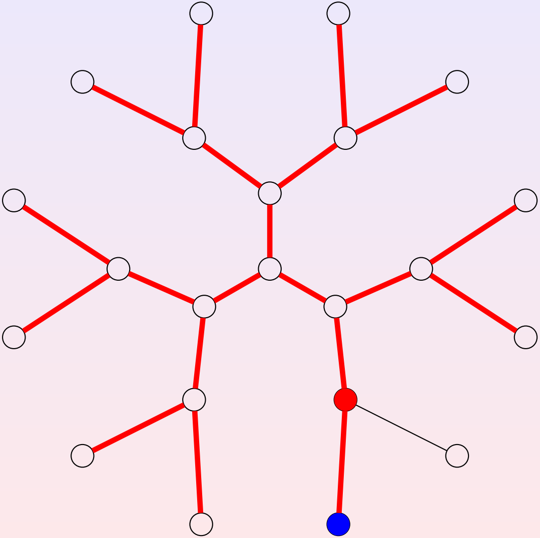
Still the same ternary tree

2 queries



Still the same ternary tree

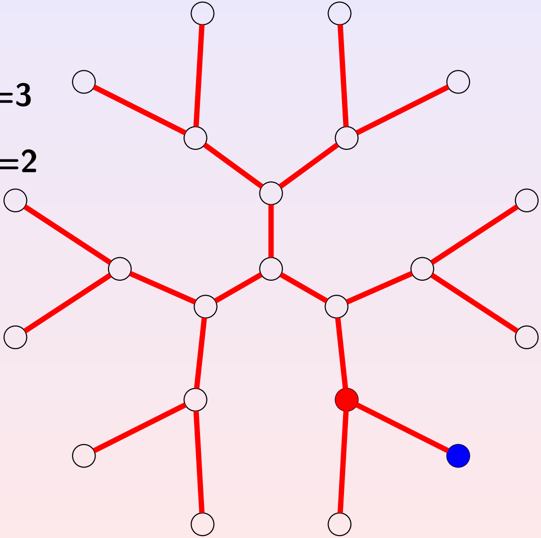
2 queries



Still the same ternary tree

$s_0(T)=s_1(T)=3$

$s_2(T)=s_\infty(T)=2$



Results

Monotonicity [Mazoit and N.]

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 $q = \infty$).

Graph's Decompositions [Fomin, Fraigniaud and N.]

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

Monotonicity: Search-tree

Auxiliary structure inspired by the tree-labelling [Robertson and Seymour, Graph Minor X]:

Search-tree = A rooted tree T labelled with subsets of $E(G)$

For any vertex $v \in V(T)$ incident to $e_1, \ldots, e_p$:

- label of v: $\ell(v) \subseteq E(G)$
- label of e_i: $\ell_v(e_i) \subseteq E(G)$

Any edge has two labels: one for each extremity.

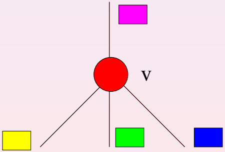
Two Properties

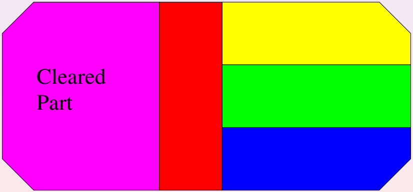
1. $\{\ell(v), \ell_v(e_1), \ell_v(e_2), \ldots, \ell_v(e_p)\}$ partition of $E(G)$;
2. $\forall e = \{u, v\} \in E(T)$, $\ell_v(e)$ and $\ell_u(e)$ are disjoint.

Monotonicity: Search-tree

Non-deterministic search strategy $\Rightarrow$ Search-tree

- placement of searchers $\Rightarrow$ vertex of T
- query $\Rightarrow$ fork (vertex of T with more than one child)
- removal of searchers $\Rightarrow$ edge of T

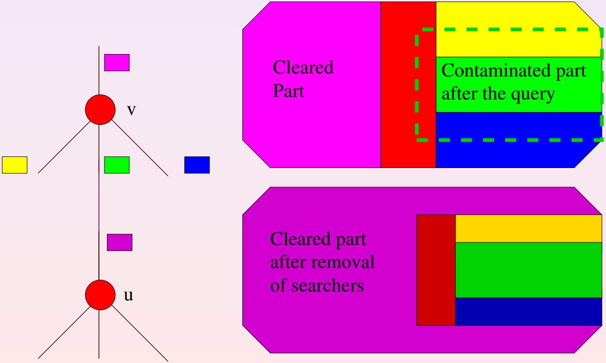




Monotonicity: Search-tree

Two Properties

1. $\{\ell(v), \ell_v(e_1), \ell_v(e_2), \dots, \ell_v(e_p)\}$ partition of $E(G)$;
2. $\forall e = \{u, v\} \in E(T)$, $\ell_v(e)$ and $\ell_u(e)$ are disjoint.



Monotonicity

Search-tree = a rooted tree T labelled with subsets of $E(G)$.

Remark: a search tree $\approx$ relaxed tree decomposition

Sketch of the proof

- (possibly non monotone) strategy $\Rightarrow$ Search-tree
- weight function over the search-trees
- minimal search-tree $\Rightarrow$ monotone strategy
- local optimization without increasing neither the number of searcher, nor the number of queries.

Recent Results and Open Problems

Class of trees

- polynomial time algorithm to compute $\mathbf{s}_1(T)$ and polynomial time 2-approximation to compute $\mathbf{s}_q(T)$, for any $q \geq 0$. [Amini, Coudert, and N.]
- $\exists$? polynomial (linear?) time algorithm to compute $\mathbf{s}_q(T)$, for any tree T, and $q \geq 0$.

Search-tree

- Generalization of the min-max theorem for treewidth [Amini, Mazoit, N. and Thomassé]
- (FPT) Algorithms? Excluding Minor Theorem?
- Application to matroids.

Outline

Connected Graph Searching

Limits of the Parson's model

- Searchers cannot move at will in a real network;
- Secured communications.

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

At any step, the cleared part of the graph must induce a connected subgraph.
Let $\mathbf{cs}(G)$ be the connected search number of the graph G.

Two main questions

What is the cost of connectivity? ratio $\mathbf{cs}/\mathbf{s}$?
Monotonicity property of connected graph searching?

Connected Graph Searching

Limits of the Parson's model

- Searchers cannot move at will in a real network;
- Secured communications.

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

At any step, the cleared part of the graph must induce a connected subgraph.
Let $\mathbf{cs}(G)$ be the connected search number of the graph G.

Two main questions

What is the cost of connectivity? ratio $\mathbf{cs}/\mathbf{s}$?
Monotonicity property of connected graph searching?

Connected Graph Searching

Limits of the Parson's model

- Searchers cannot move at will in a real network;
- Secured communications.

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

At any step, the cleared part of the graph must induce a connected subgraph.
Let $\mathbf{cs}(G)$ be the connected search number of the graph G.

Two main questions

What is the cost of connectivity? ratio $\mathbf{cs}/\mathbf{s}$?
Monotonicity property of connected graph searching?

The cost of connectedness

In terms of number of searchers

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

For any connected graph G, $\mathbf{cs}(G) \leq \mathbf{s}(G)\ (2 + \log |E(G)|)$.
Fomin, Fraigniaud, and Thilikos [Tech. Rep. 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

Using the concept of *connected* tree-decomposition.

Cost of connectivity [Fraigniaud and N.]

For any n-node connected graph G, $\mathbf{cs}(G)/\mathbf{s}(G) \leq \log n$.

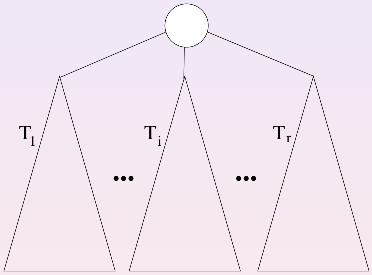
Graphs with bounded chordality k [N.]

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

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

Sketch of proof: $\mathbf{cs}(G) \leq \mathbf{s}(G)\ \log n$

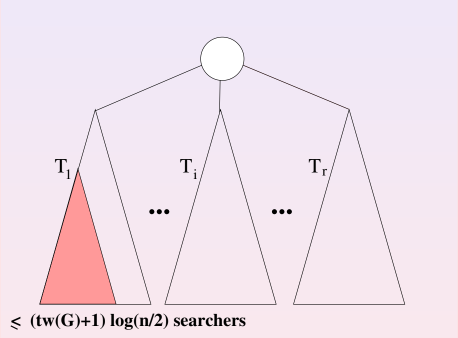
Proof by induction on n: $\mathbf{cs}(G) \leq (\mathbf{tw}(G)+1)\ \log n$



For any $1 \leq i \leq r$, $G[T_i]$ is a connected subgraph with at most $n/2$ vertices.

Sketch of proof: $\mathbf{cs}(G) \leq \mathbf{s}(G)\ \log n$

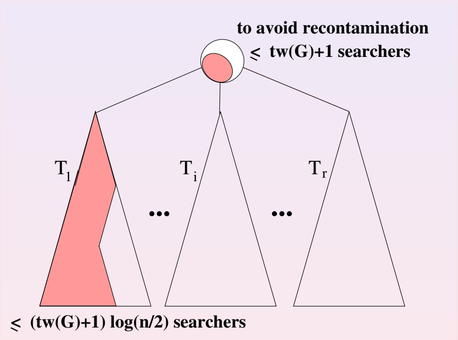
Proof by induction on n: $\mathbf{cs}(G) \leq (\mathbf{tw}(G)+1)\ \log n$



There is a connected search strategy for $G[T_1]$, using at most $(\mathbf{tw}(G)+1)\log(n/2)$ searchers.

Sketch of proof: $\mathbf{cs}(G) \leq \mathbf{s}(G)\ \log n$

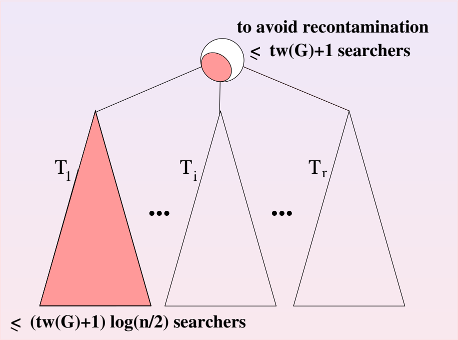
Proof by induction on n: $\mathbf{cs}(G) \leq (\mathbf{tw}(G)+1)\ \log n$



At most $\mathbf{tw}(G)+1$ searchers are required to protect $G[T_1]$ from recontamination from the remaining part of G.

Sketch of proof: $\mathbf{cs}(G) \leq \mathbf{s}(G)\ \log n$

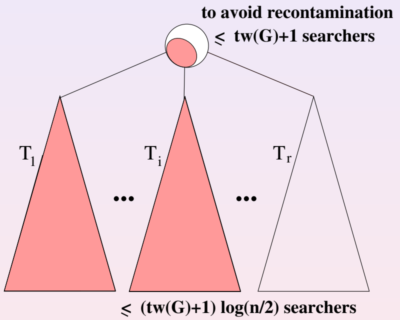
Proof by induction on n: $\mathbf{cs}(G) \leq (\mathbf{tw}(G)+1)\ \log n$



Then, we can terminate the clearing of $G[T_1]$.

Sketch of proof: $\mathbf{cs}(G) \leq \mathbf{s}(G)\ \log n$

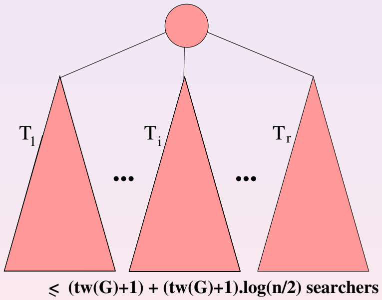
Proof by induction on n: $\mathbf{cs}(G) \leq (\mathbf{tw}(G)+1)\ \log n$



The $(\mathbf{tw}(G)+1)\log(n/2)$ searchers can be used to clear another subgraph $G[T_i]$, and so on...

Sketch of proof: $\mathbf{cs}(G) \leq \mathbf{s}(G)\ \log n$

Proof by induction on n: $\mathbf{cs}(G) \leq (\mathbf{tw}(G)+1)\ \log n$



Connected search strategy using at most $(\mathbf{tw}(G)+1)\log n$ searchers. Thus, $\mathbf{cs}(G) \leq \mathbf{s}(G)\log n$

Results: Case of a visible fugitive

Cost of connectivity [Fraigniaud and N.]

For any n-node graph G, $\mathbf{cvs}(G)/\mathbf{vs}(G) \leq \log n$
tight for monotone strategies: $\mathbf{mcvs}(G)/\mathbf{vs}(G) \geq \Omega(\log n)$.

Non Monotonicity [Fraigniaud and N.]

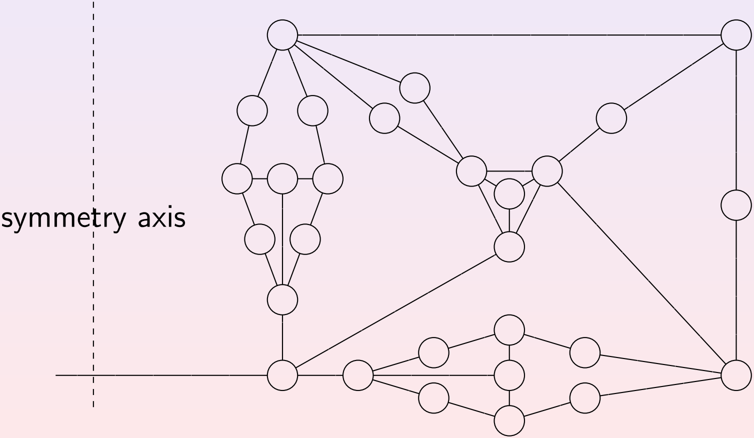
In visible connected graph searching, recontamination helps

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

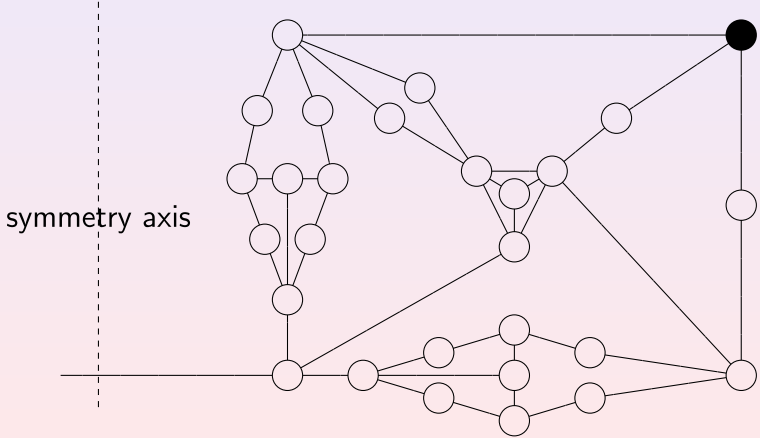
Let G be the graph below: **mcvs**$(G) >$ **cvs**$(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

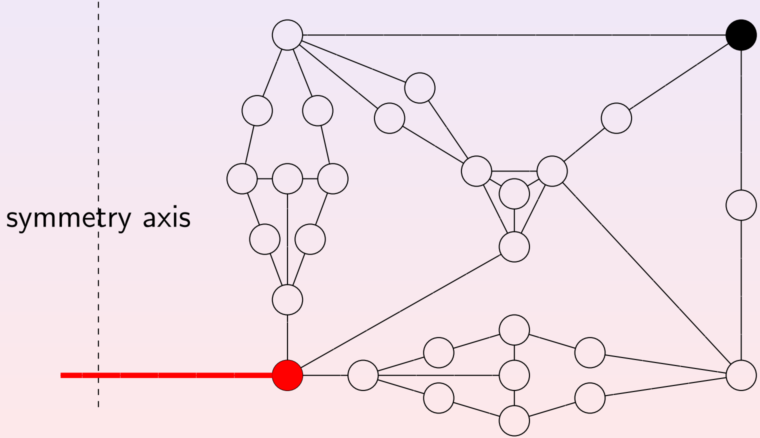
Let G be the graph below: $\textbf{mcvs}(G) > \textbf{cvs}(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

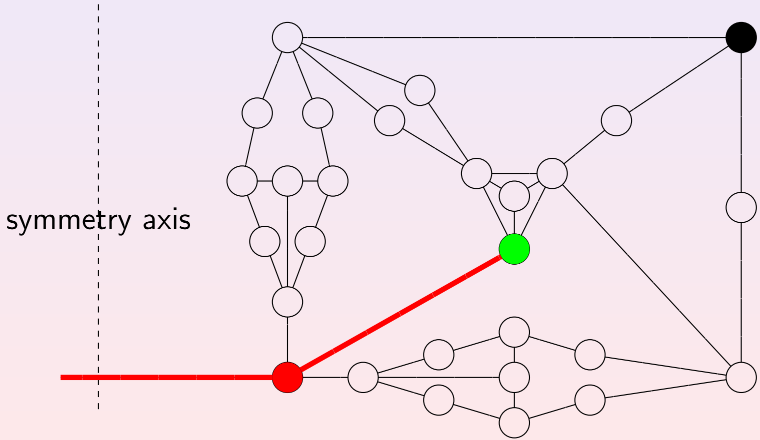
Let G be the graph below: **mcvs**$(G) >$ **cvs**$(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

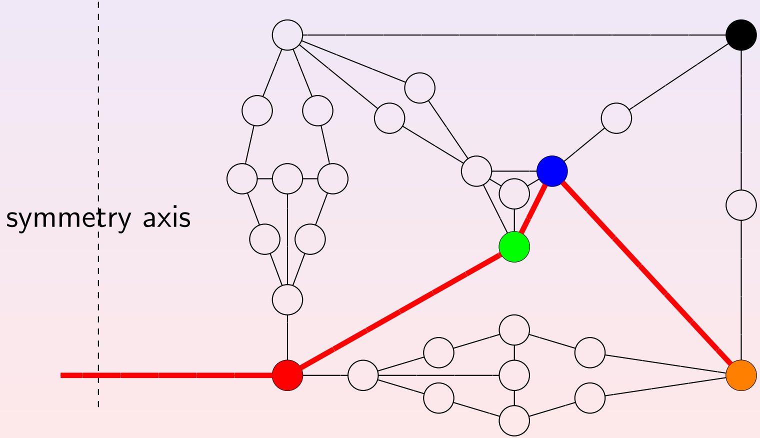
Let G be the graph below: $\textbf{mcvs}(G) > \textbf{cvs}(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

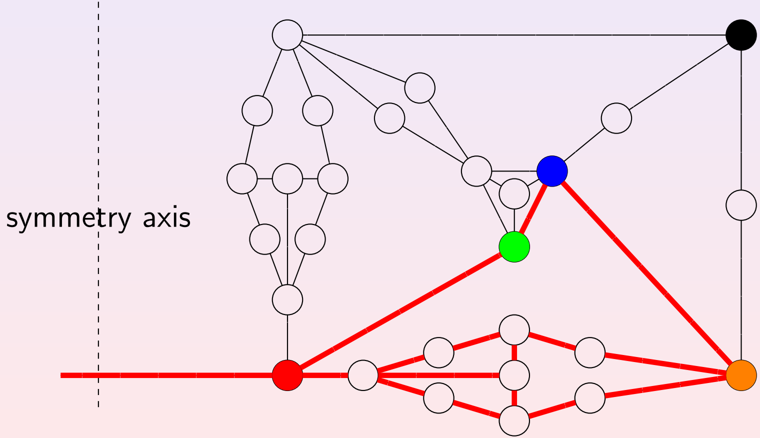
Let G be the graph below: $\mathbf{mcvs}(G) > \mathbf{cvs}(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

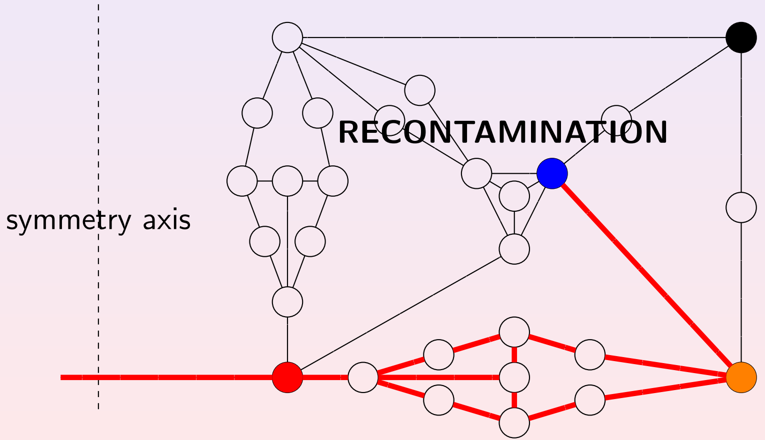
Let G be the graph below: $\mathbf{mcvs}(G) > \mathbf{cvs}(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

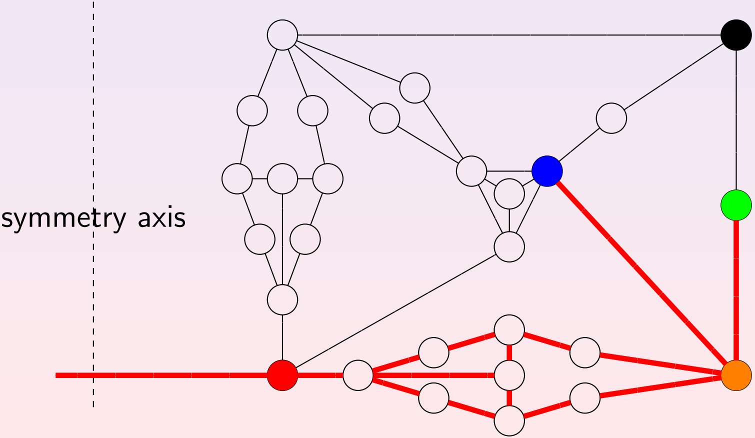
Let G be the graph below: **mcvs**$(G) >$ **cvs**$(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

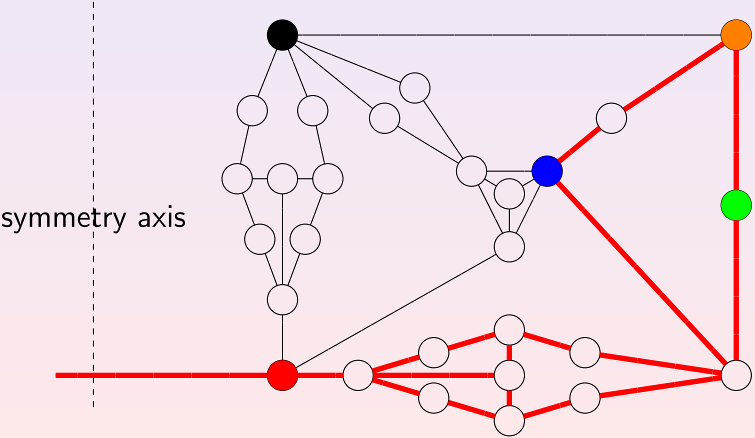
Let G be the graph below: $\mathbf{mcvs}(G) > \mathbf{cvs}(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

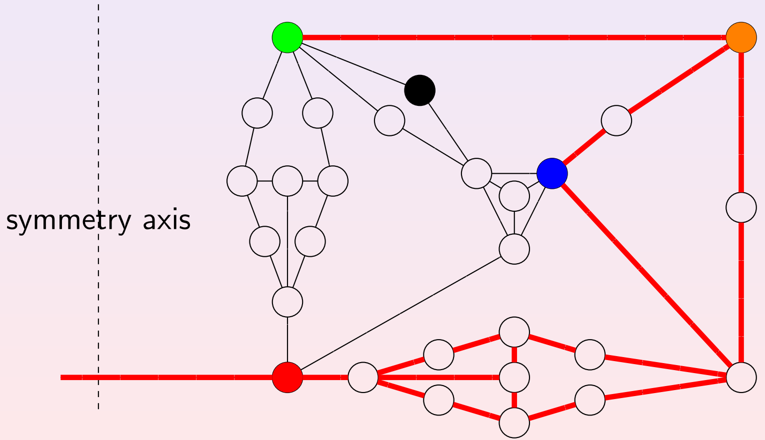
Let G be the graph below: **mcvs**$(G) >$ **cvs**$(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

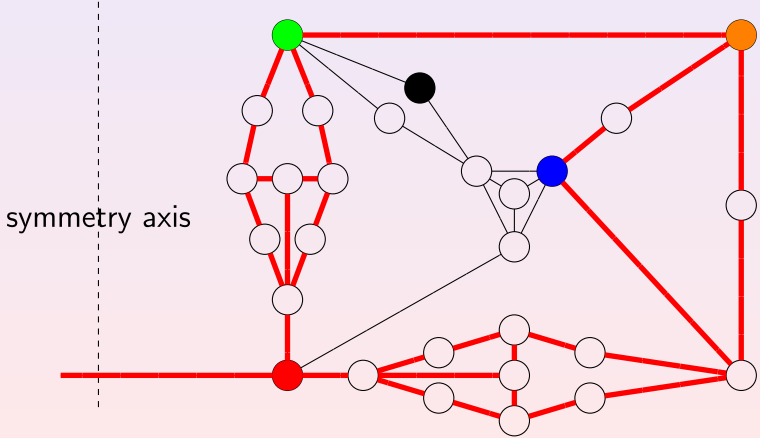
Let G be the graph below: **mcvs**$(G) >$ **cvs**$(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

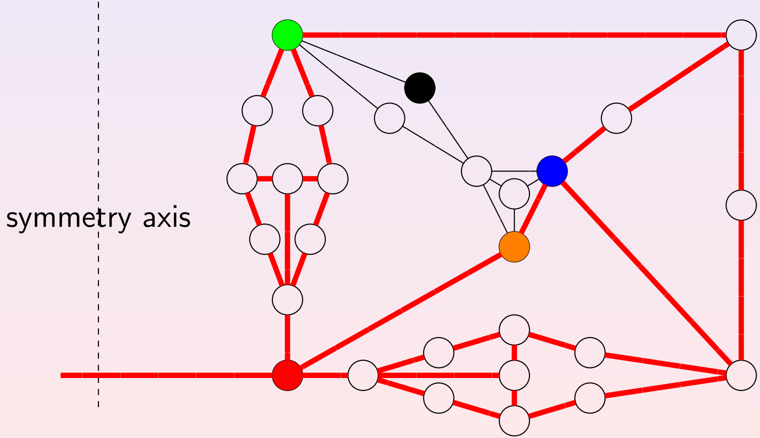
Let G be the graph below: **mcvs**$(G) >$ **cvs**$(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

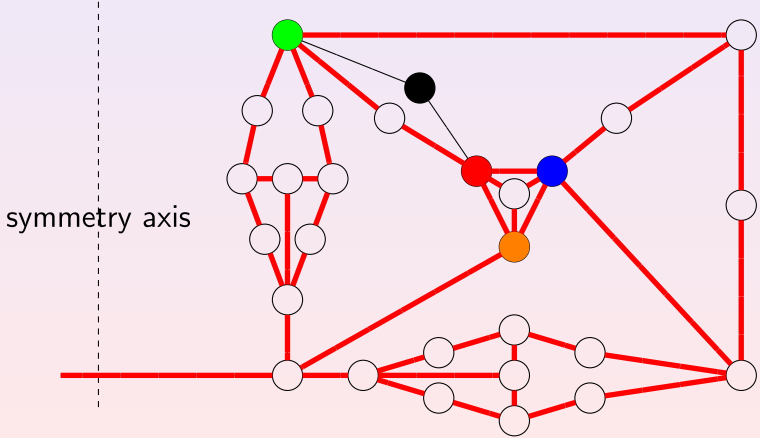
Let G be the graph below: $\mathbf{mcvs}(G) > \mathbf{cvs}(G) = 4$.



Non-monotonicity

Recontamination helps in visible connected graph searching

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



Connected Graph Searching: Open Problems

Cost of connectivity

Conjecture: For any graph G, $\mathbf{cs}(G)/\mathbf{s}(G) \leq 2$ [Barrière *et al.*]

FPT Algorithm

FPT algorithm to compute $\mathbf{cs}(G)$?

NP membership

Bound on the number of recontamination-steps?

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.

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

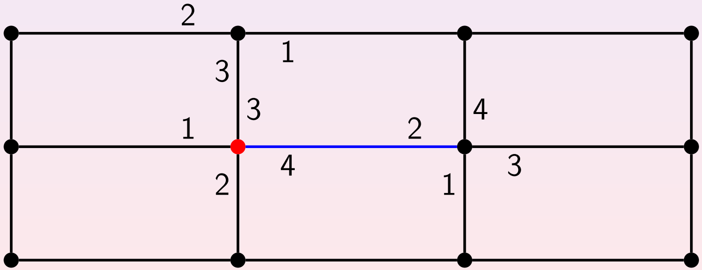
To design a *distributed protocol* that enables the *minimum number* of searchers to clear the network.
The searchers must compute themselves a strategy.

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: the network

- undirected connected graph;
- local orientation of the edges;
- whiteboards on vertices;
- 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 priori 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]
- **Sierpinski's graph**. Luccio. [FUN 07]

A monotone connected strategy is performed using **mcs** $+ 1$ searchers, in polynomial time.

Remark:

The extra searcher is due to the asynchronicity of the network and it is necessary [CIC 05].

Distributed graph searching: related work

The searchers have a priori 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]
- **Sierpinski's graph**. Luccio. [FUN 07]

A monotone connected strategy is performed using **mcs** $+ 1$ searchers, in polynomial time.

Remark:

The extra searcher is due to the asynchronicity of the network and it is necessary [CIC 05].

Results

Distributed algorithm [Blin, Fraigniaud, N. and Vial]

Distributed protocol that enable **$\mathbf{mcs}(G) + 1$** searchers to clear an unknown graph G in a connected way

Sketch of the algorithm

For $k \geq 0$ to n **do**

Test all connected search strategies using $\leq k$ searchers.
(Difficulty: to ensure that all strategies will be checked)
If this fails, another searcher is called (k++).

Drawback: the strategy is not monotone and may be performed in exponential time.

Results

Distributed algorithm [Blin, Fraigniaud, N. and Vial]

Distributed protocol that enable $\mathbf{mcs}(G) + 1$ searchers to clear an unknown graph G in a connected way

Sketch of the algorithm

For $k \geq 0$ to n **do**

Test all connected search strategies using $\leq k$ searchers.
(Difficulty: to ensure that all strategies will be checked)
If this fails, another searcher is called (k++).

Drawback: the strategy is not monotone and may be performed in exponential time.

Results

Unless P=NP, there is no hope to clear an unknown graph in a monotone way, using the optimal number of searchers.

Provide knowledge about the graph [N. and Soguet]

$\Theta(n \log n)$ bits of information must be provided to the $\textbf{mcs}(G)$ searchers to clear a unknown graph G in a monotone connected way.

Use more searchers [Ilcinkas, N. and Soguet]

$\Theta(\frac{n}{\log n})\textbf{mcs}(G)$ searchers are necessary and sufficient to clear any unknown graph G in a connected *monotone* way.

Distributed Graph Searching: Open Problems

Tradeoff between the number of searchers and the amount of information about the graph.

Better algorithm to clear a graph, possibly computing non-monotone search strategies?

Outline

Further Works

Directed graph searching

Several recent works [Obdrzalek, Hunter *et al.*, Adler, etc.] using "directed" tree decompositions.
Duality therorem?
(FPT) Algorithms?

Cost of connectivity

Conjecture: For any graph G, $\mathbf{cs}(G) \leq 2\mathbf{s}(G)$ [Barrière *et al.*]

Search-tree

- Generalization of the min-max theorem for treewidth [Amini, Mazoit, N. and Thomassé]
- (FPT) Algorithms? Excluding Minor Theorem?
- Application to matroids.