

# Graph Exploration and Graph Searching

Pierre Fraigniaud

CNRS

LRI, Univ. Paris Sud, Orsay

# The problems

- Graph exploration
  - Mobile entities have to visit all nodes (or to traverse all edges) of a network
- Graph searching
  - Mobile entities have to capture an intruder in a network

# Motivations

- Applications
  - Network security
  - Robotic
- Fundamental aspects
  - Graph minors theory
  - Computational complexity

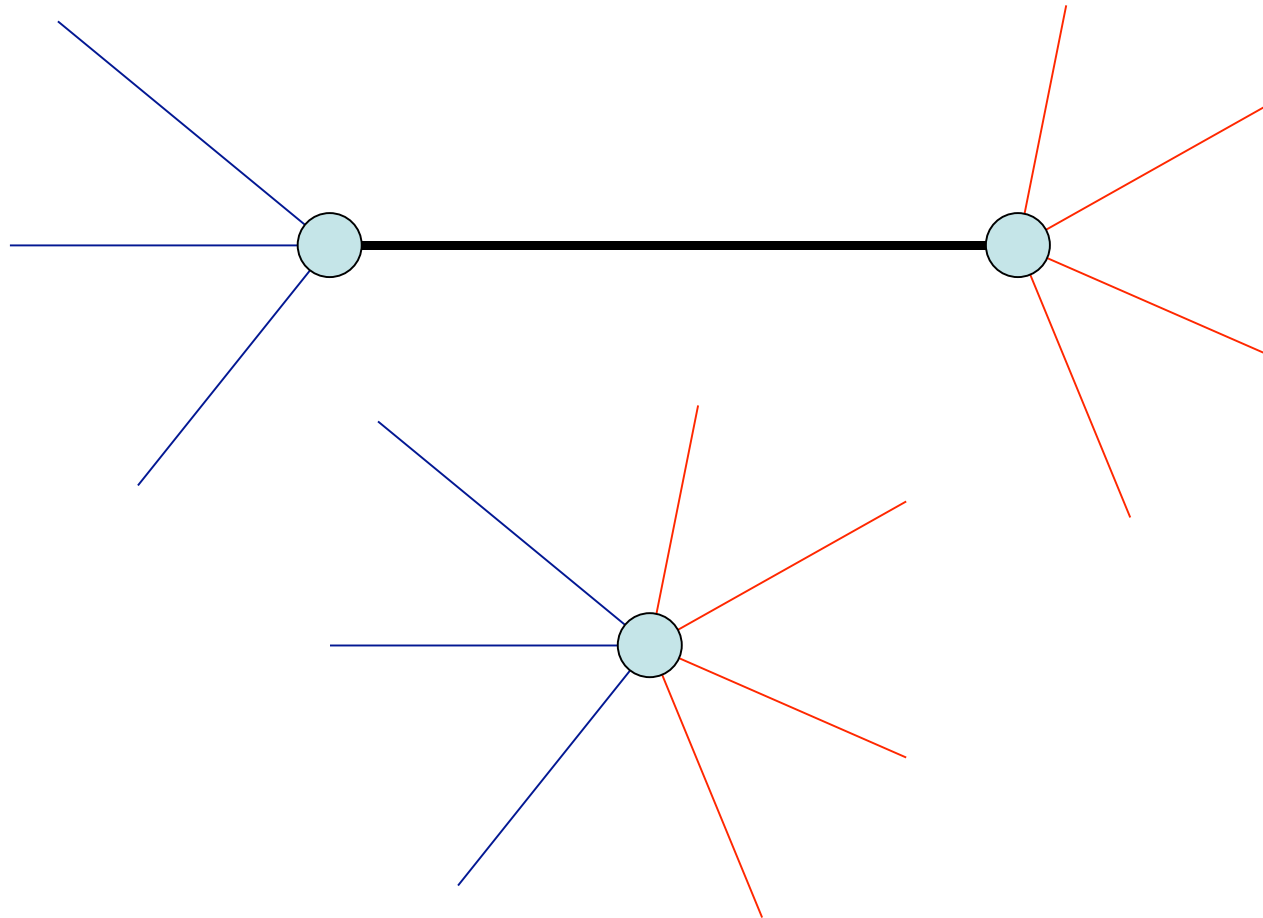
# **I. Fundamental Motivations**

## **I.1. Graph searching**

# Graph minors

- A graph  $H$  is a minor of a graph  $G$  if  $H$  can be obtained from  $G$  by a sequence of the following operations:
  - Removing an edge
  - Removing a node
  - Contracting an edge

# Edge contraction

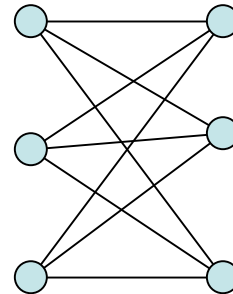
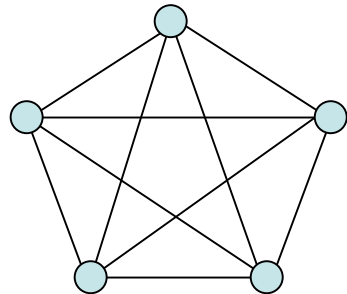


# Minor-closed

- A family  $F$  of graphs is minor-closed if, for any  $G \in F$ , every minor of  $G$  belongs to  $F$
- Examples
  - Planar graphs
  - Forests

# Forbidden minors

- Theorem (Robertson and Seymour)  
Any minor-closed family has a finite set of forbidden minors (Wagner's conjecture)
- Example  
A graph is planar iff it does not has  $K_5$  or  $K_{3,3}$  as minor (Kuratowski's theorem)



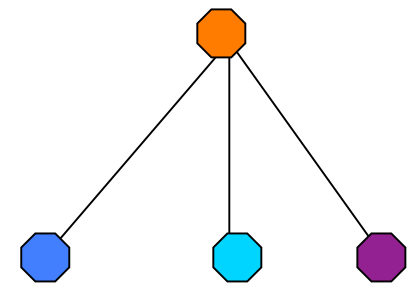
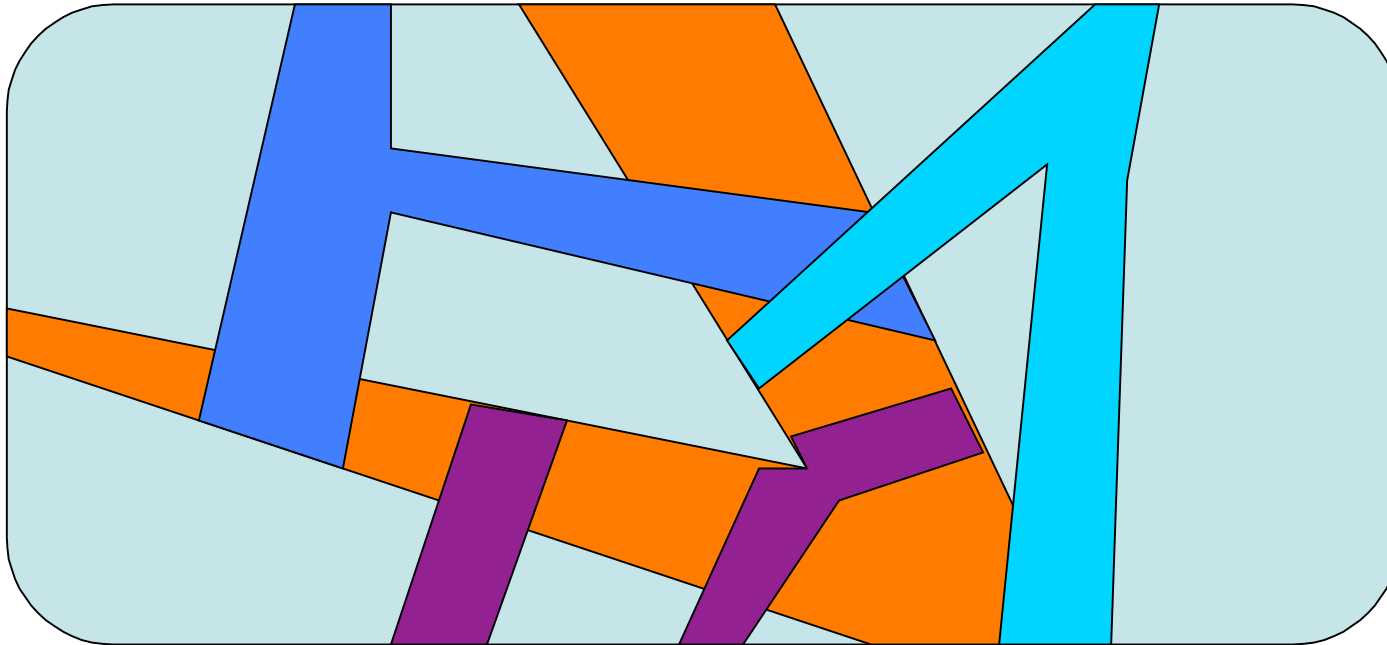
# Corollary

- $F$  a minor-closed family of graphs
- **Theorem** Determining whether  $G \in F$  can be computed in polynomial time
- **Algorithm:**
  - For every forbidden minor  $H$  of  $F$  do:  
check that  $H$  is not a minor of  $G$

# Tree-decomposition

- A tree-decomposition of a graph  $G$  is a tree  $T=(I,F)$  and a collection  $X=\{X_i, i \in I\}$  of subsets of vertices (i.e.,  $X_i \subseteq V(G)$ ) such that:
  - $\forall v \in V(G), \exists i \in I / v \in X_i$
  - $\forall e=\{u,v\} \in E(G), \exists i \in I / u,v \in X_i$
  - $\forall v \in V(G), \{i \in I / v \in X_i\}$  is a subtree of  $T$

# Recursive Separators



# Treewidth

- The **width** of a tree-decomposition is defined as:  $\max_{i \in I} |X_i| - 1$
- The **treewidth**  $tw(G)$  of a graph  $G$  is defined as the minimum width of a tree-decomposition of  $G$ .

# Examples

- For any graph  $G$ ,  $tw(G) \leq n-1$
- For any complete graph:  $tw(K_n) = n-1$
- For any tree  $T$ ,  $tw(T) = 1$
- For any cycle  $C_n$ ,  $tw(C_n) = 2$

# Algorithmic issues

- For any integer  $k \geq 1$ ,  $F_k = \{G, \text{tw}(G) \leq k\}$  is minor-closed.
- Theorem (Courcelle)  
On graphs of treewidth at most  $k$ , where  $k$  is fixed, every decision or optimization problem expressible in monadic second-order logic has a linear algorithm.

# Structural issues

- Determining the forbidden minors of  $F_k = \{G, tw(G) \leq k\}$ .
- Theorem (Robertson and Seymour)  
 $\forall r \geq 1, \exists k$  such that every graph  $G$  with  $tw(G) \geq k$  has a  $r \times r$  mesh as minor.
- Remark: best bound for  $k$ :  $k \leq 2^{O(r^5)}$

# Search games

- Several notions of “width” have an interpretation in term of *search games*
  - treewidth = visible (or inert) search
  - pathwidth (i.e., same as treewidth but restricted to path-decomposition) = node search

# **I. Fundamental Motivations**

## **I.2. Graph exploration**

# Time complexity

- **P** (Polynomial)

Class of problems "solvable" in polynomial time, i.e.,  $\text{time-for-solving}(I) = O(\text{poly}(n))$  where  $n$  is the size of the instance  $I$ .

- **NP** (Non-deterministic polynomial)

Class of problems for which a solution is "checkable" in polynomial time.

- Challenge: is  $P = NP$ ?

# Space complexity

- **L** (log-Space)

Class of problems "solvable" in logarithmic space, i.e.,  $\text{space-for-solving}(I) = O(\log n)$  where  $n$  is the size of the instance  $I$ .

- **NL** (Non-deterministic Log-Space)

Class of problems "solvable" in logarithmic space by a non-deterministic Turing machine.

- Challenges:  $P = L?$     $NL = L?$

# The class SL

- **SL** (Non-deterministic Symmetric Log-Space) (Lewis and Papadimitriou)
  - Class of problems "solvable" in logarithmic space by a non-deterministic symmetric Turing machine:

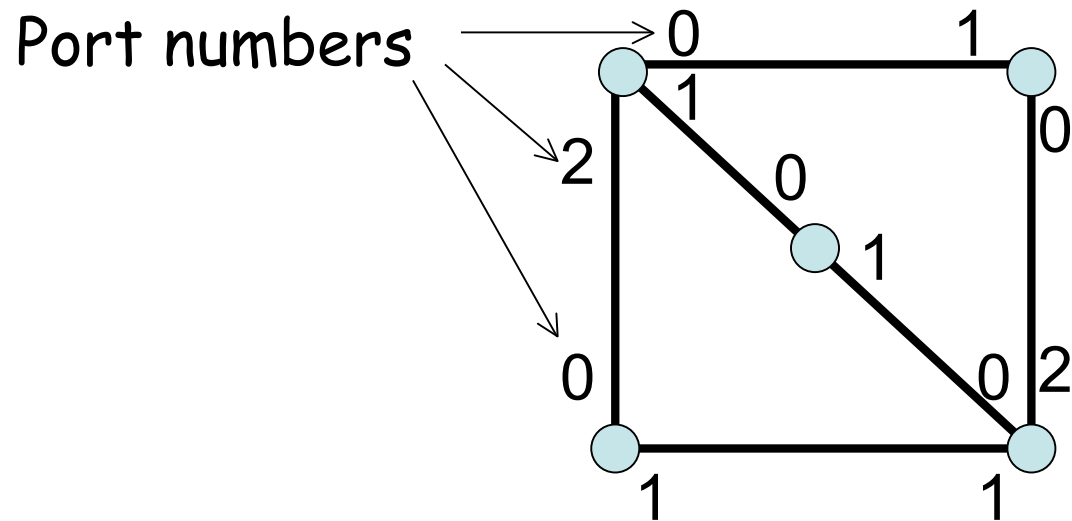
if  $S \rightarrow S'$  then  $S' \rightarrow S$

- fact:  $L \subseteq SL \subseteq NL$
- $SL = L ?$     $SL = NL ?$

# s-t connectivity

- **s-t CON**: Given two nodes **s** and **t** of **G**, determine whether there is a path from **s** to **t** in **G**?
- Two facts:
  - **Theorem** If **G** is directed, then **STCON** is **NL**-complete
  - **Theorem** (Lewis and Papadimitriou) If **G** is undirected, then **USTCON** is **SL**-complete
- **Challenge**: solving **(U)STCON** in log-space.

# Graph exploration



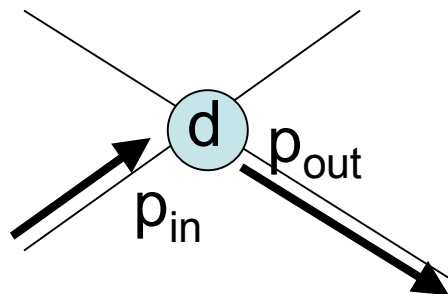
# Finite automata

- Machine with a finite number of states:

$$S = \{s_0, s_1, \dots, s_{k-1}\}$$

- Transition function:

$$f(s_i, p_{in}, d) = (s_j, p_{out})$$



$d = \text{node degree}$

# Questions

- Is there exist a **universal** finite automaton? (i.e., an automaton able to explore all graphs)
- If not, is there a machine exploring all graphs using  $O(\log n)$  memory in a graph of order  $n$ ?

## **II. Graph Searching**

# General objective

Mobile entities have to capture an intruder in a network

# **II. Graph Searching**

## **II.1. Definition**

# Graph searching (Parson)

- Graph searching involves:
  - A set of "searchers"
  - A "fugitive"moving along the edges of a graph
- The fugitive is caught when met by a searcher
- Edge search:
  - Fugitive
    - Arbitrarily fast
    - Permanently know the positions of the searchers
  - Searchers
    - Unaware of the position of the fugitive

# Alternative definition

- Graph searching involves:
  - A set of "searchers"
  - A "contaminated" graph

The traversal of an edge by a searcher **clears** the edge

- The **graph is clear** when all edges have been cleared
- An edge is preserved from **recontamination** if any path between the edge to a contaminated part of the graph contains a searcher.

# Search strategy

- A search strategy is a sequence of:
  - Place a searcher at a node
  - Remove a searcher from a node
  - Move a searcher along an edge
- A search strategy is *winning* if it clears the graph.

# Search number

- The *search number*  $s(G)$  of a graph  $G$  is the minimum number of searchers required by a winning search strategy
- Examples:
  - Paths:  $s(P_n) = 1$
  - Cycles:  $s(C_n) = 2$
  - Cliques:  $s(K_n) = n$  (for  $n \geq 4$ )

# Difficulty of the problem

- Theorem (Megiddo, Hakimi, Garey, Johnson, Papadimitriou)
  - The following decision problem is NP-hard
    - Input: a graph  $G$ , and integer  $k$ ;
    - Question:  $s(G) \leq k$  ?
- Question: is it NP-complete?
- Remark: polynomial for trees

# **II. Graph Searching**

## **II.2. Monotony**

# Monotone strategy

- A search strategy is *monotone* if recontamination never occurs.
- Theorem (Lapaugh; Bienstock and Seymour)  
For any graph  $G$ , there exists a winning monotone search strategy using  $s(G)$  searchers.

# Corollaries

- Recontamination does not help  
⇒ one can restrict the analysis to monotone search strategies
- $s(G) \leq k$  is NP-complete

# Variants (1)

## Node-search (Kirousis and Papadimitriou)

- Two operations:
  - place a searcher at node
  - remove a searcher from node
- An edge is cleared when there are searchers at its two end points.
- $ns(G)$  = minimum number of searchers for a winning node-search strategy.

# Variants (2)

## Mixed-search (Bienstock and Seymour)

- Same operations as edge-search
- An edge is clear
  - either by sliding a searcher
  - or by placing searchers at the two extremities
- $ms(G)$  = minimum number of searchers for a winning mixed-search strategy.

# Proof of the Monotone Theorem

(Bienstock and Seymour)

- $\forall G, ms(G) \leq s(G)$  and  $ms(G) \leq ns(G)$
- $G^-$  is obtained from  $G$  by replacing each edge by two edges in series:



- $G''$  is obtained from  $G$  by replacing each edge by two edges in parallel:

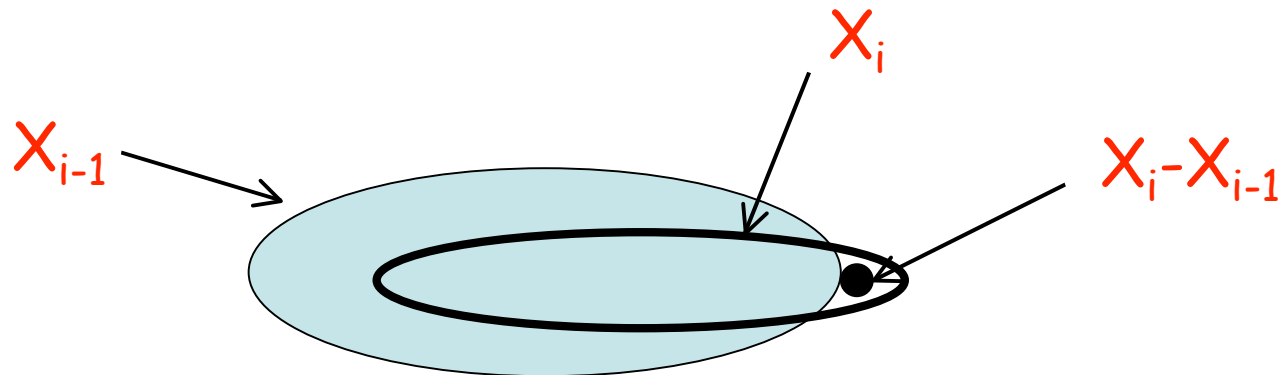


# Monotone mixed search

- $s(G) = ms(G^{--})$
  - $ns(G) = ms(G^{//})$
  - Moreover, a monotone optimal mixed-search strategy in  $G^{--}$  and  $G^{//}$  can be converted to a monotone optimal edge- or node-search strategy, respectively, in  $G$ .
- ⇒ monotonicity for mixed search implies monotonicity for both edge- and node-search.

# Crusades

- A crusade is a sequence  $(X_0, X_1, \dots, X_n)$  of subsets of  $E(G)$  such that:
  - $X_0 = \emptyset$  and  $X_n = E(G)$
  - $|X_i - X_{i-1}| \leq 1$  for all  $i > 0$



# Border

- $G=(V,E), X \subseteq E$
- $\delta(X)$  = set of nodes with incident edges in  $X$  and incident edges in  $E-X$ .
- $|\delta(X)| = |\delta(E-X)|$
- Submodular inequality:  
$$|\delta(X \cap Y)| + |\delta(X \cup Y)| \leq |\delta(X)| + |\delta(Y)|$$

# Progressive crusades

- A crusade  $(X_0, X_1, \dots, X_n)$  is progressive if
  1.  $X_0 \subseteq X_1 \subseteq \dots \subseteq X_n$
  2.  $|X_i - X_{i-1}| = 1$  for all  $i > 0$
- A crusade  $(X_0, X_1, \dots, X_n)$  uses  $k$  guards if  $|\delta(X_i)| \leq k$  for all  $i = 0, \dots, n$

# Proof sketch

- (1) If  $ms(G) \leq k$  then there is a crusade in  $G$  using  $\leq k$  guards
- (2) If there is a crusade in  $G$  using  $\leq k$  guards then there is a **progressive** crusade in  $G$  using  $\leq k$  guards
- (3) (under the hypothesis that every node in  $G$  is incident to at least 2 edges) If there is a progressive crusade in  $G$  using  $\leq k$  guards then there is a **monotone** mixed search strategy for  $G$  using  $\leq k$  searchers

# Proof of (2)

Let  $(X_0, X_1, \dots, X_n)$  be a crusade using  $\leq k$  guards such that:

(a)  $\sum_i (|\delta(X_i)| + 1)$  is minimum

(b)  $\sum_i |X_i|$  is minimum subject to (a)

$|X_i - X_{i-1}| = 1$  because otherwise

$$(X_0, X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n)$$

is a crusade contradicting (a)

$|\delta(X_{i-1} \cup X_i)| \geq |\delta(X_i)|$  because otherwise

$$(X_0, X_1, \dots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \dots, X_n)$$

is a crusade contradicting (a)

# Proof of (2) -- cont'd

By submodularity inequality:

$$|\delta(X_{i-1} \cap X_i)| \leq |\delta(X_{i-1})|$$

Hence

$$(X_0, X_1, \dots, X_{i-2}, X_{i-1} \cap X_i, X_i, X_{i+1}, \dots, X_n)$$

is a crusade using  $\leq k$  guards.

By (b) we get:

$$|X_{i-1} \cap X_i| \geq |X_{i-1}|$$

Hence

$$X_{i-1} \subseteq X_i$$

# **II. Graph Searching**

## **II.3. Search and width**

# Equivalence theorem

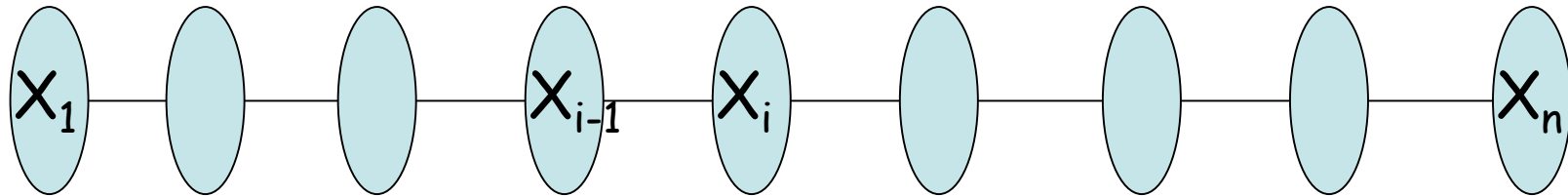
- Theorem
  - The node-search number of a graph is equal to its pathwidth plus one:
- Theorem (Seymour and Thomas; Dendriss, Kirousis and Thilikos)
  - The visible-search number of a graph is equal to its treewidth plus one:

$$ns(G) = pw(G) + 1$$

$$vs(G) = tw(G) + 1$$

# Proof of $ns(G) = pw(G) + 1$

An optimal path-decomposition of  $G$



Lemma:  $X_{i-1} \cap X_i$  is a separator of the graph

Node search strategy:

Place a searcher at every node of  $X_1$

For  $i=2$  to  $n$  do

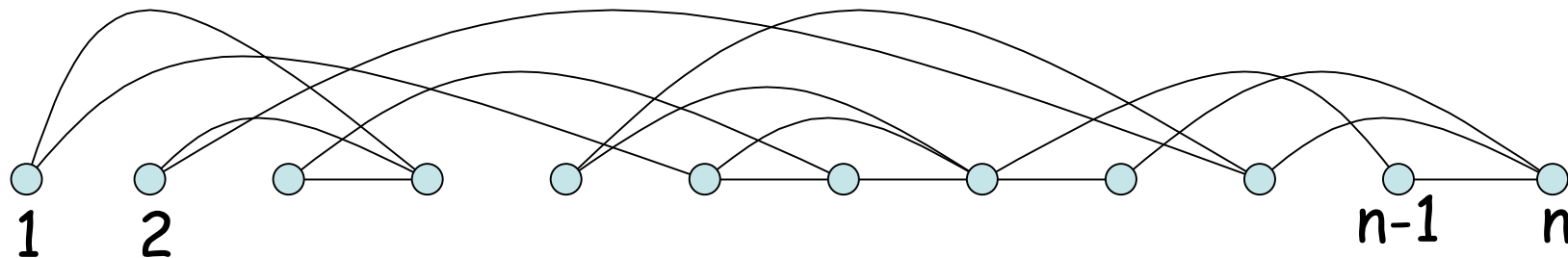
1) remove all searchers in  $X_{i-1} - (X_{i-1} \cap X_i)$

2) Place a searcher at every node of  $X_i - (X_{i-1} \cap X_i)$

# Linear layouts

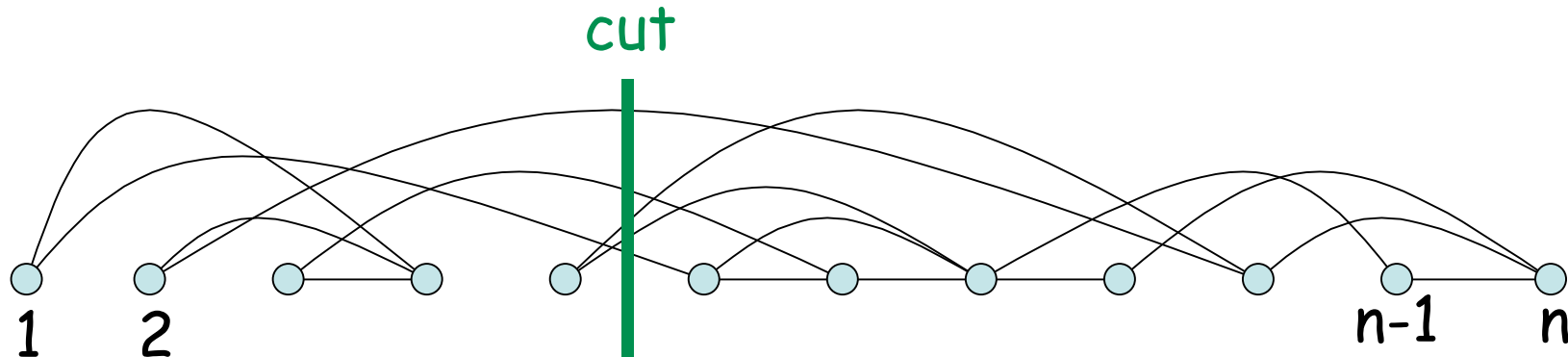
- Definition

- Label the nodes of  $G$  from 1 to  $n=|V(G)|$



- Layouts are characterized by different parameters (corresponding to as many width notions, defined by cuts)

# Vertex separation



- $\text{size}(\text{cut}) = \# \text{node on the left side of the cut, with a neighbor on the right side of the cut}$
- $\text{width}(\text{layout}) = \max_{\text{all cuts}} \text{size}(\text{cut})$
- $\text{vs}(G) = \min_{\text{all layouts}} \text{width}(\text{layout})$

# Some results

- Theorem (Ellis, Sudborough, Turner)

$$ns(G) = vs(G) + 1$$

- Corollary

$$vs(G) \leq s(G) \leq vs(G) + 2$$

- Proof:

$$ns(G) - 1 \leq s(G) \leq ns(G) + 1$$

# **III. Connected Search**

# Objective

Searching a graph in a connected way

# **III. Connected Search**

## **III.1. Motivation and definition**

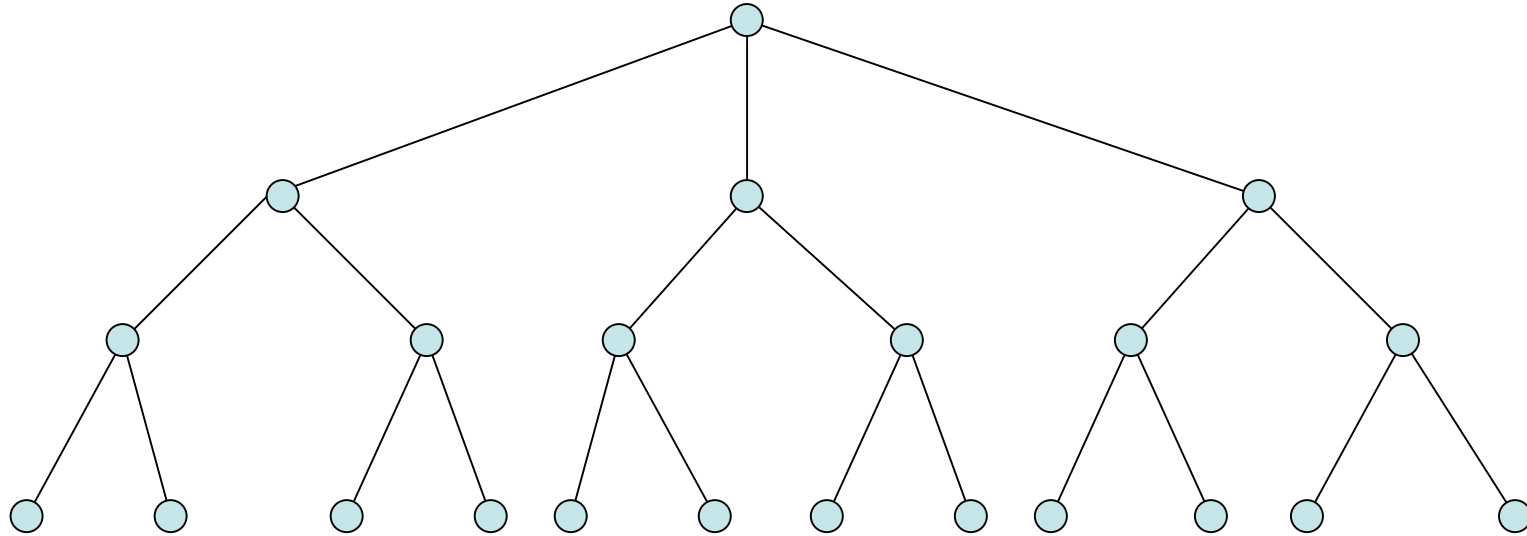
# Motivation

- The operation “place a searcher” is unfeasible in many contexts (networks, speleology, etc.)  
→ **internal** search
- **Secure moves** and **secure communications** between searchers require that searchers should better not be split into several groups  
→ **connected** search

# Definition

- A search strategy is **connected** if at any step the subgraph induced by the clear edges is connected.
- The minimum number of searchers required to clear a connected graph  $G$  with a connected search strategy is denoted by  $cs(G)$
- Obviously:  $cs(G) \geq s(G)$

$\exists G$  with  $cs(G) > s(G)$



- $s(B_3) = 3$

- $cs(B_3) = 4$

$s(T) \leq \log_3 n$  whereas  $\exists T / cs(T) \approx \log_2 n$

# Difficulties

- **Bad news 1:** the family  $F_k = \{G / cs(G) \leq k\}$  is not minor-closed
- **Bad news 2 (Dyer):**  
A **monotone** optimal connected search strategy does not necessarily exist
- **Consequence:** The problem  $cs(G) \leq k ?$  is NP-hard, but it is not known whether there exists a yes-certificate checkable in polynomial time.

# **III. Connected Search**

## **III.2. The case of trees**

# Monotony

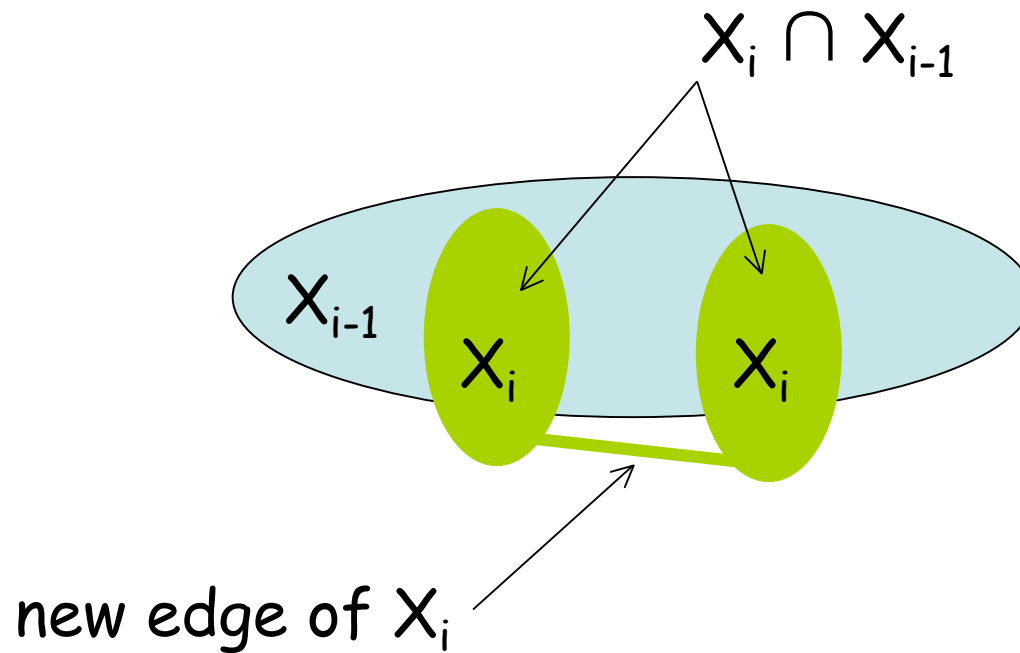
- **Connected** crusade: every  $X_i$  is connected
- Going back to Bienstock and Thomas proof:

$$(X_0, X_1, \dots, X_{i-2}, X_{i-1} \cap X_i, X_i, X_{i+1}, \dots, X_n)$$

is not necessarily a connected crusade because  $X_{i-1} \cap X_i$  may not be connected

- Theorem (Barrière, Flocchini, F., Santoro)  
For any tree  $T$ , there exists a monotone connected search strategy using  $cs(T)$  searchers.

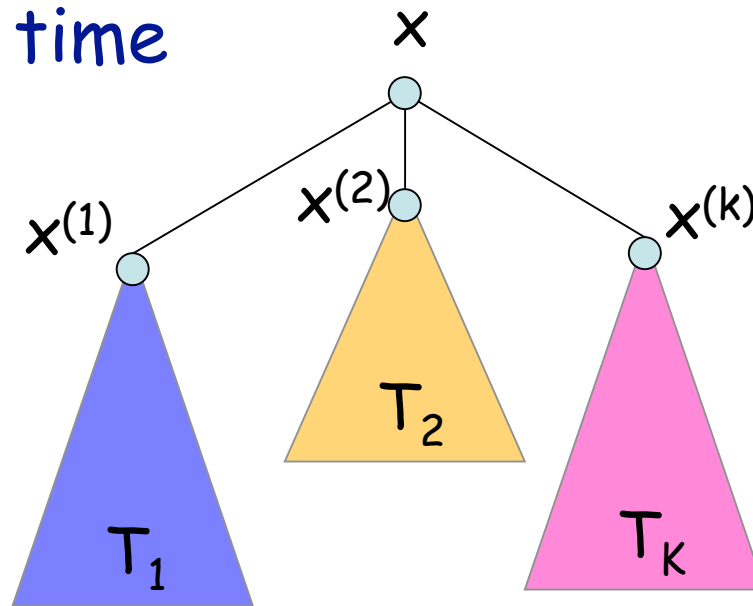
$X_{i-1} \cap X_i$  is connected in trees



# Linear-time algorithm for trees

- Theorem (Barrière, Flocchini, F., Santoro)

For any tree  $T$ ,  $cs(T)$  can be computed in linear time



$$n_i = cs_{x^{(i)}}(T_i)$$

$$n_1 \geq n_2 \geq \dots \geq n_k$$

$$n_1 = n_2 \Rightarrow cs_x(T) = n_1 + 1$$

$$n_1 > n_2 \Rightarrow cs_x(T) = n_1$$

# Obstructions for trees

- **Theorem** (Parson; Takahashi, Ueno, Kajitani)  
The number of forbidden minors for  
 $\{\text{tree } T / s(T) \leq k\}$  is at least  $(k!)^2$
- **Theorem** (Barrière, F., Santoro, Thilikos)  
The  $k$ -caterpillar is the unique forbidden  
minor for  $\{\text{tree } T / cs(T) \leq k\}$

# k-caterpillars

- A spine is a path
- Caterpillars:
  - A 0-caterpillar  $D_0$  is a path, and is its own spine
  - A tree  $T$  is a  $k$ -caterpillar  $D_k$  with spine  $P$  if, for every component  $T'$  of  $T-P$ 
    - There is a path  $P'$  such that  $T'$  is a  $(k-1)$ -caterpillar  $D_{k-1}$  with spine  $P'$
    - One of the two extremities of  $P'$  is adjacent to  $P$

# **III. Connected Search**

## **III.3. Ratio $cs/s$**

# Problem

- For any connected graph  $G$ ,  $cs(G) \geq s(G)$
- Can we bound the ratio  $cs(G)/s(G)$ ? I.e., what is the "price of connectedness"?

# cs/s for trees

- Theorem (Barrière, F., Santoro, Thilikos)  
For any tree  $T$  such that  $s(T) \geq 2$  we have  
 $cs(T) \leq 2s(T) - 2$

# General case

- Theorem (Fomin, F., Thilikos)

For any  $m$ -edge connected graph  $G$ :

$$cs(G)/s(G) \leq \lceil \log m \rceil + 1$$

Tool: carvings (connected branchwidth)

- Theorem (F. and Nisse)

For any  $n$ -node connected graph  $G$ :

$$cs(G)/s(G) \leq \log n + 1$$

Tool: connected treewidth

# Connected tree-decomposition

- Let  $(T, X)$  be a tree-decomposition of  $G$ , and  $e$  an edge of  $T$ . The  $e$ -cut is the two subtrees of  $T - \{e\}$
- $(T, X)$  is connected if, for every  $e$ -cut  $\{T_1, T_2\}$ , the two subgraphs  $G_1$  and  $G_2$  of  $G$  induced by the bags of  $V(T_1)$  and  $V(T_2)$ , respectively, are connected.

# Connected treewidth

- $ctw(G)$  = minimum width of a connected tree-decomposition of  $G$
- Theorem (Golombic; Parra and Scheffler)  
For any connected graph  $G$ ,  $ctw(G) = tw(G)$

# Proof of $cs/s \leq O(\log n)$

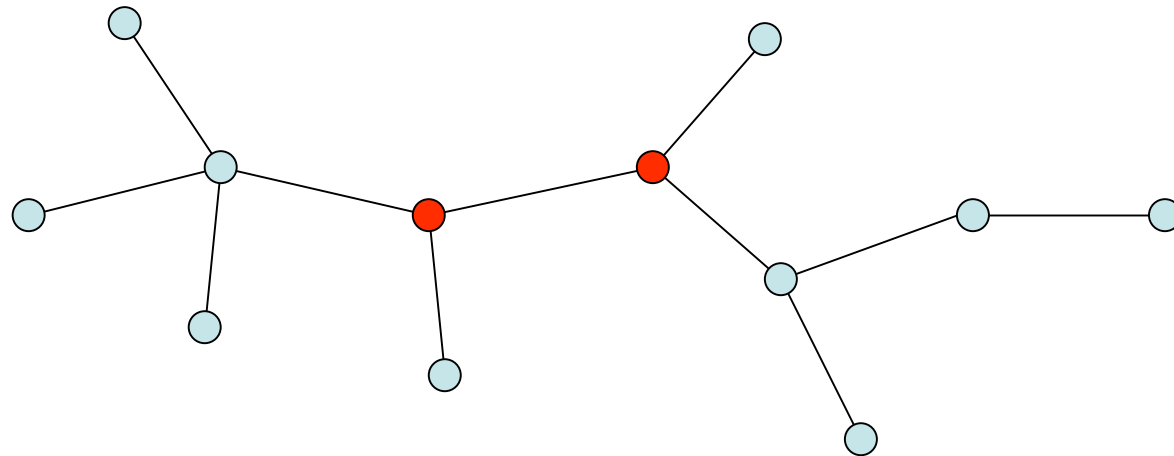
Lemma:  $cs(G) \leq O(\log n) tw(G)$  starting from any node

Proof:

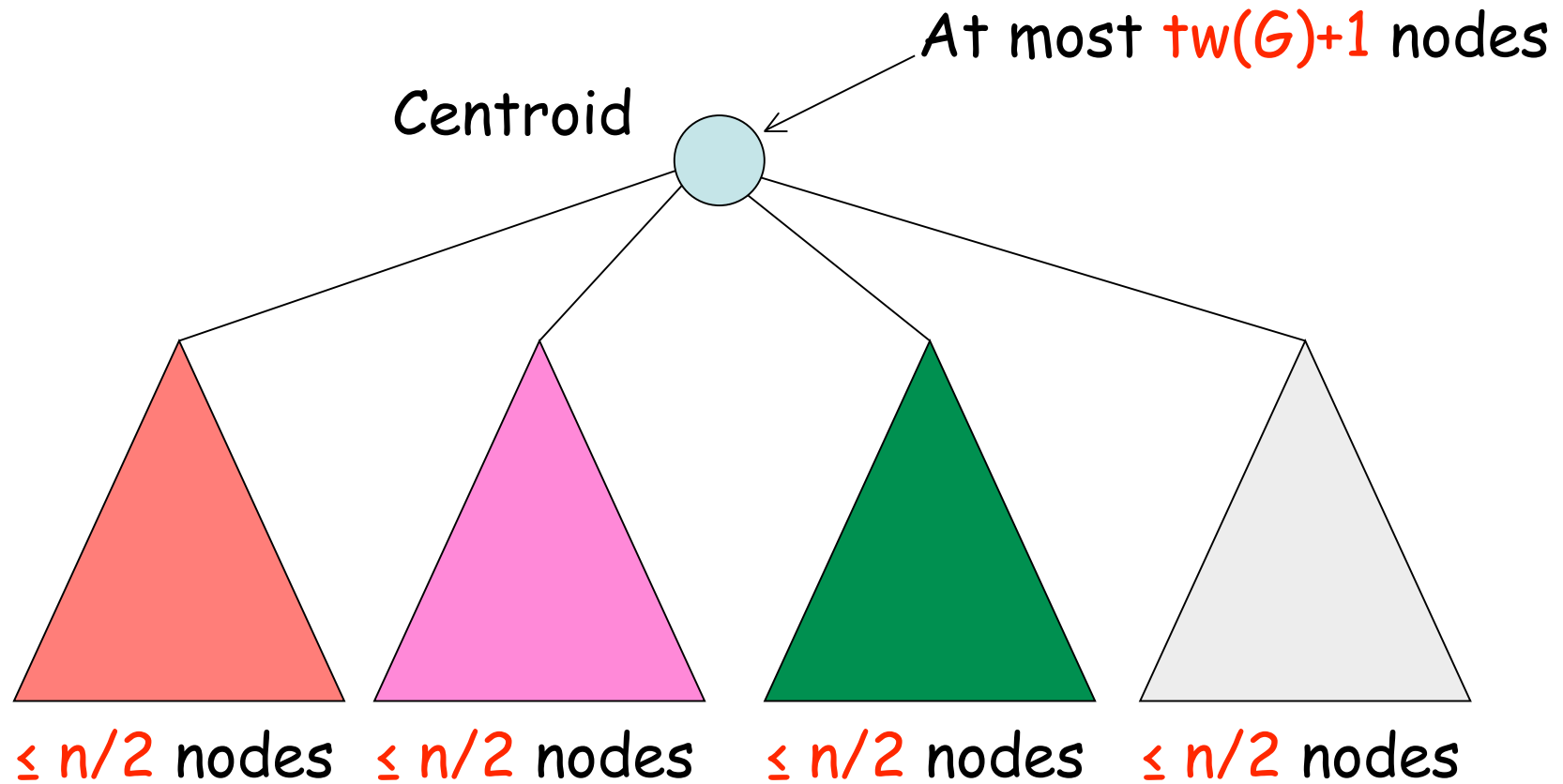
- Recall  $(X_0, X_1, \dots, X_n)$  is a connected crusade if  $X_i$  induce a connected subgraph for any  $i$
- One can show that if there exists a connected crusade in  $G$  using  $\leq k$  guards then  $cs(G) \leq k+1$
- **Claim:**  $\forall$  connected graph  $G$ , and  $\forall e \in E(G)$ , there exists a connected crusade  $(X_0, X_1, \dots, X_n)$  using  $\leq tw(G) \log n$  guards, and such that  $X_1 = \{e\}$ .

# Centroid

A centroid of an  $n$ -node tree  $T$  is a vertex  $v$  such that  $T - \{v\}$  is a forest of trees, each of at most  $n/2$  vertices.



# Centroids of Tree-decomposition



QED

# Conjecture

For every connected graph  $G$ , we have

$$cs(G)/s(G) \leq 2$$

# **IV. Graph exploration**

# General objective

Mobile entities have to traverse all  
edges of an unknown graph

# **IV. Graph exploration**

## **IV.1. Single finite automaton**

# Moore vs. Mealy automata

- Moore automaton

- The out-going port number is part of the automaton's state:

$$f(s_i, p_{in}, d) = s_j \text{ and } p_{out} = \lambda(s_j)$$

- Mealy automaton

- The transition function computes the out-going port number and the new state:

$$f(s_i, p_{in}, d) = (s_j, p_{out})$$

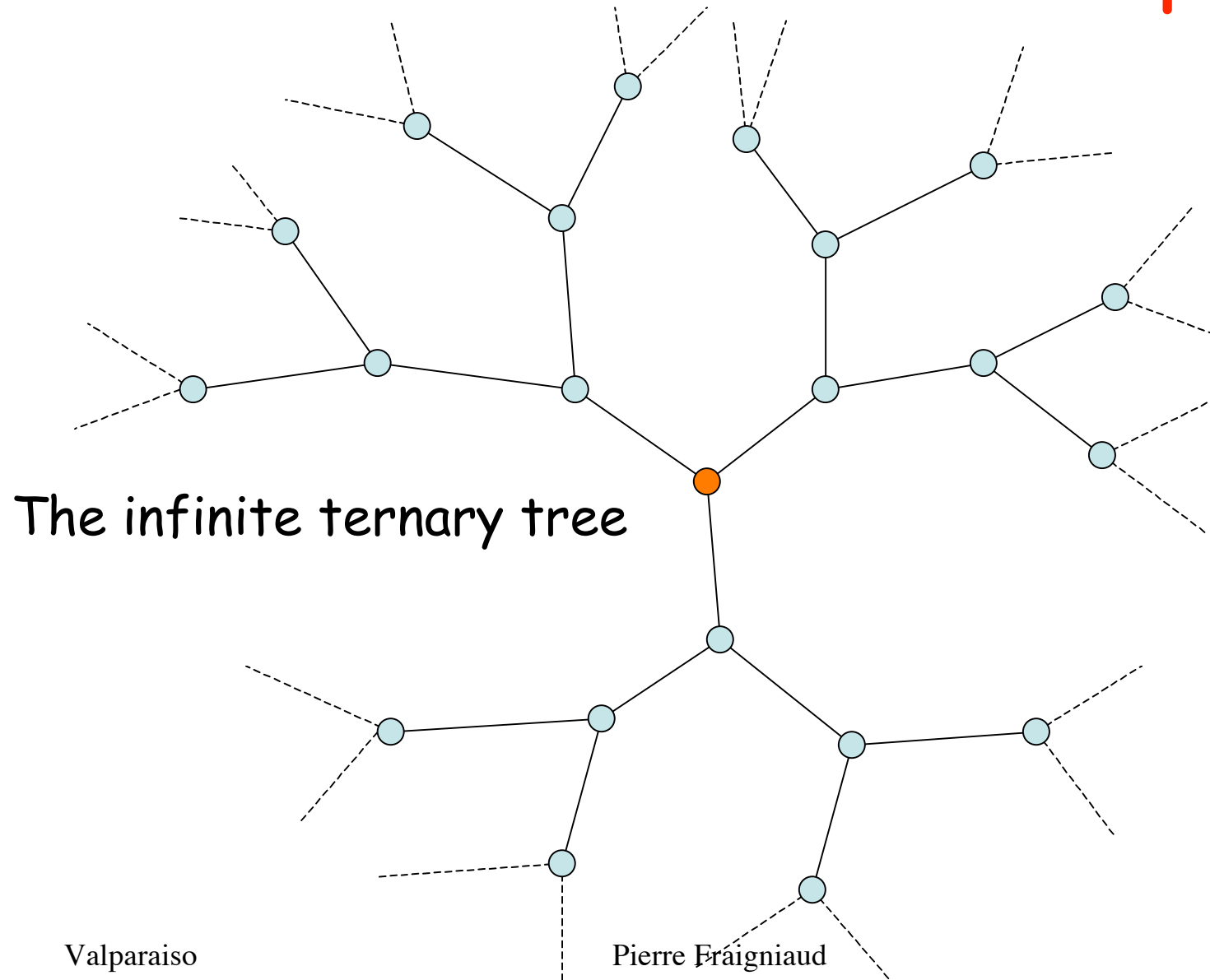
# Traps

- A pair  $(G, v)$  is a trap for an automaton  $A$  if  $A$  does not explore  $G$  starting from  $v$ .
- Theorem (Budach)
  - Any finite automaton has a trap

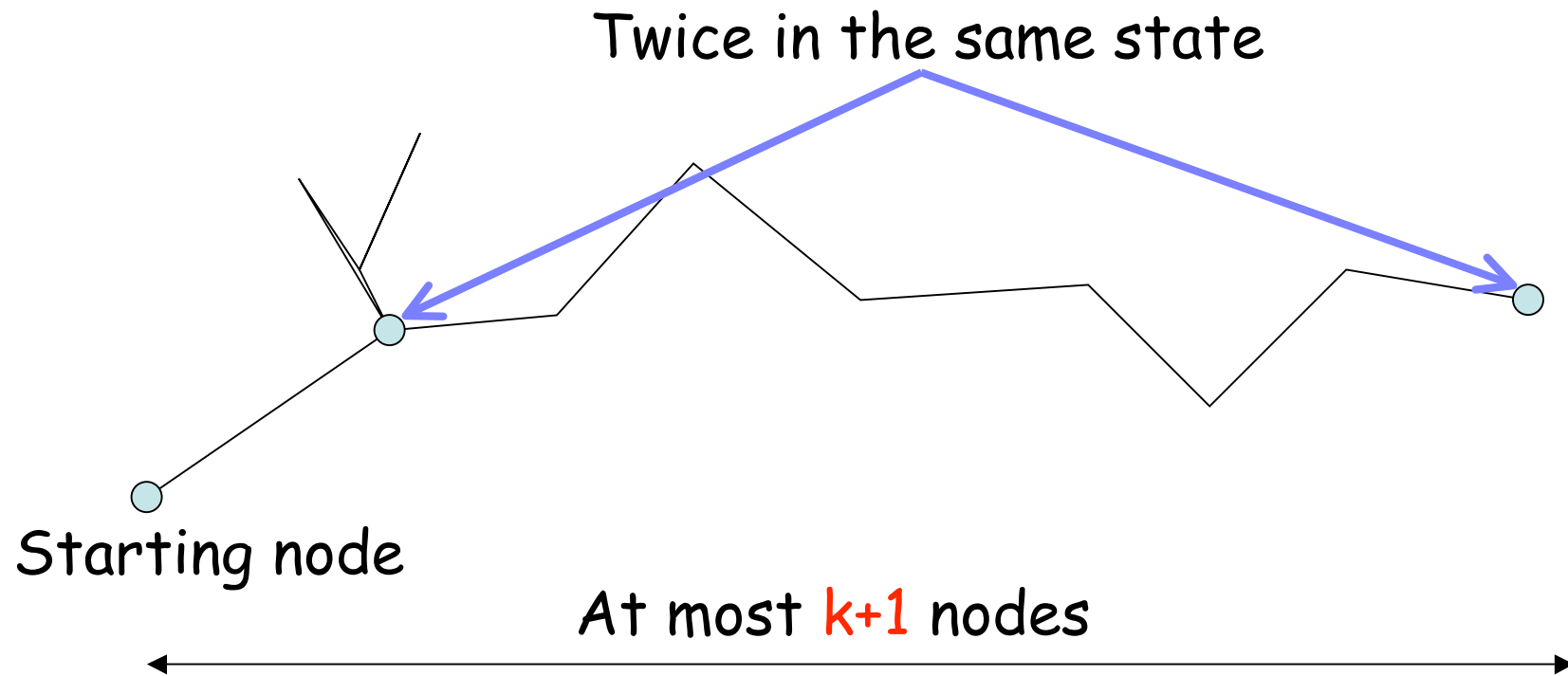
# Impossibility results

- Theorem (F., Ilcinkas, Peer, Pelc, Peleg)  
Any  $k$ -state automaton has a trap of size  $\leq k+1$
- Corollary  
A machine exploring all graphs of order  $\leq n$  needs  $\Omega(\log n)$  bits of memory.

# Construction of a trap



# Automaton trajectory



# DFS is space-optimal

- By adding an tree-superstructure on top of the previous construction, one gets the space bound  $\Omega(D \log \Delta)$  bits for exploring all graphs of diameter  $\leq D$  and maximum degree  $\leq \Delta$
- DFS at increasing depth gives a space bound  $O(D \log \Delta)$  bits.
- Theorem (F., Ilcinkas, Peer, Pelc, Peleg)  
Graph exploration of all graphs of diameter  $\leq D$  and maximum degree  $\leq \Delta$  requires  $\Theta(D \log \Delta)$  bits.

# Open problem

- Is there an automaton using space  $O(\log n)$  bits for exploring all graphs of at most  $n$  nodes?

# **IV. Graph exploration**

## **IV.2. More powerful finite machines**

# Team of non-cooperative automata

- A **team** of non-cooperative automata is a finite set  $\{A_1, \dots, A_k\}$  of finite automata, each executing its own program independently from the others
- **Theorem**  
Any team of non-cooperative finite automata can be trapped.

# Automaton using pebbles

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

- Theorem

Any finite automaton with a finite number of (possibly different) pebbles can be trapped.

# Team of cooperative automata

- A **team** of cooperative automata is a finite set  $\{A_1, \dots, A_k\}$  of finite automata, pairwise interacting when meeting at same node (i.e., exchanging states)
- Theorem (Rollik)  
Any team of cooperative finite automata can be trapped.

# Methodology

- Non-cooperative automata
  - Put a trap for the  $k-1$  automata  $A_1, \dots, A_i$  in every edge of a trap for the automaton  $A_{i+1}$
- Cooperative automata
  - Either they are separated, and then they act as non-cooperative automata,
  - Or they remain together, and then they act as a "big" automaton

# Hydra

- A  $k$ -head hydra is a set  $\{A_1, \dots, A_k\}$  of  $k$  finite automata, permanently interacting
- Alternatively, a hydra is a multi-head automaton
- Theorem (F., Ilcinkas, Pelc)  
Any 3-head hydra can be trapped.

# JAGs

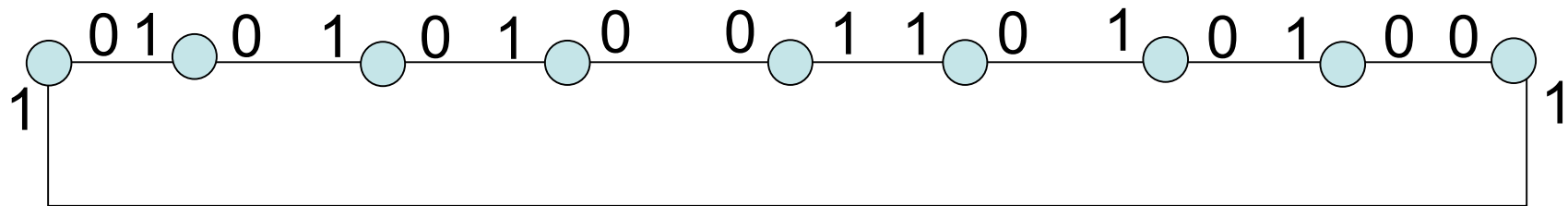
- Jumping Automaton for Graphs
- A JAG is:
  - A hydra
  - Any head can "jump" at a node occupied by another head
- Theorem (Cook and Rackoff)  
Any JAG can be trapped.

# **IV. Graph exploration**

## **IV.3. Universal Traversal Sequences**

# The graph family $G_{n,d}$

- $G_{n,d} = \{d\text{-regular } n\text{-node connected graphs}\}$
- Arbitrarily labeled



# Definition (Cook)

- For fixed  $n$  and  $d$ , a universal traversal sequence (UTS) is a sequence  $S=(p_1,p_2,\dots,p_r)$ ,  $0 \leq p_i \leq d-1$  such that any arbitrarily labeled graph of  $G_{n,d}$  is explored by  $S$
- Exploration algorithm:
  - at step  $i$ , take port  $p_i$  to leave the current node

# Two major issues

- The **length** of an UTS
- The **space** complexity for constructing a UTS
- Existential result: For any  **$n$**  and  **$d$** , there exists a UTS for  $G_{n,d}$  of length  **$O(d n^3 \log n)$**

# Universal Exploration Sequences (Koucky)

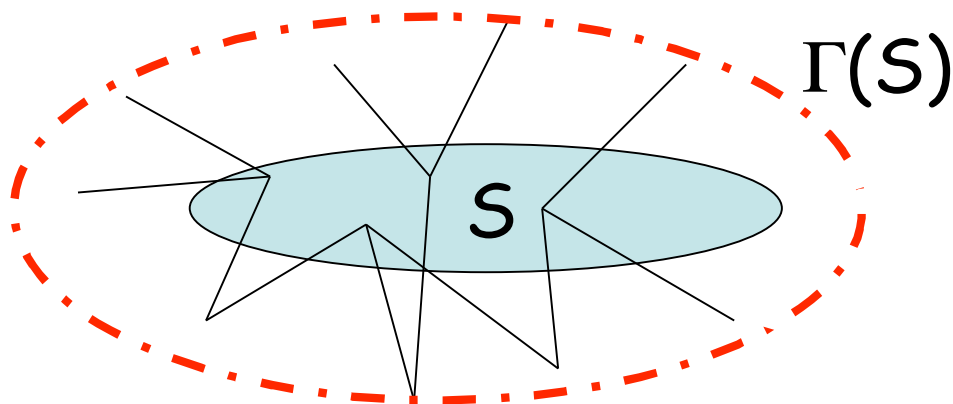
- For fixed  $n$  and  $d$ , a universal exploration sequence (UXS) is a sequence  $S = (p_1, p_2, \dots, p_r)$ ,  $0 \leq p_i \leq d-1$  such that any arbitrarily labeled graph of  $G_{n,d}$  is explored by  $S$
- Exploration algorithm:
  - at step  $i$ , leave the current node by port  $q_i + p_i \bmod d$  where this node was entered by port  $q_i$

# Some results

- Existential result: For any  $n$  and  $d$ , there exists a UXS for  $G_{n,d}$  of length  $O(d^2 n^3 \log n)$
- The sequence  $1^n$  is a UXS for  $G_{n,2}$
- The sequence  $1^{2n}$  is a UXS for  $n$ -node trees
- UXS allow backtracking:  $p_i = 0$

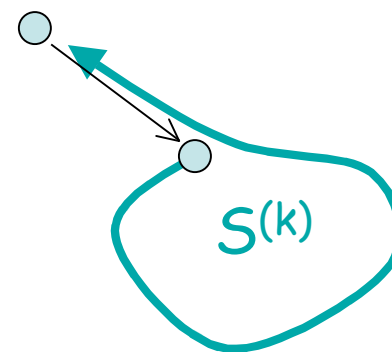
# Expanders

- A graph  $G$  is an expander if there exists  $\beta > 1$  such that, for any  $S \subset V(G)$  (with  $|S|$  "not too big") the following holds:  $|S \cup \Gamma(S)| \geq \beta |S|$



# UXS for DFS

- Theorem (Koucky)
  - $S^{(0)} = 0$
  - $S^{(k)} = (1S^{(k-1)})^{d-1}$
  - DFS at depth  $h$ :  $(1S^{(h-1)})^d$
- Corollary
  - There exists a log-space constructible UXS for all expanders.



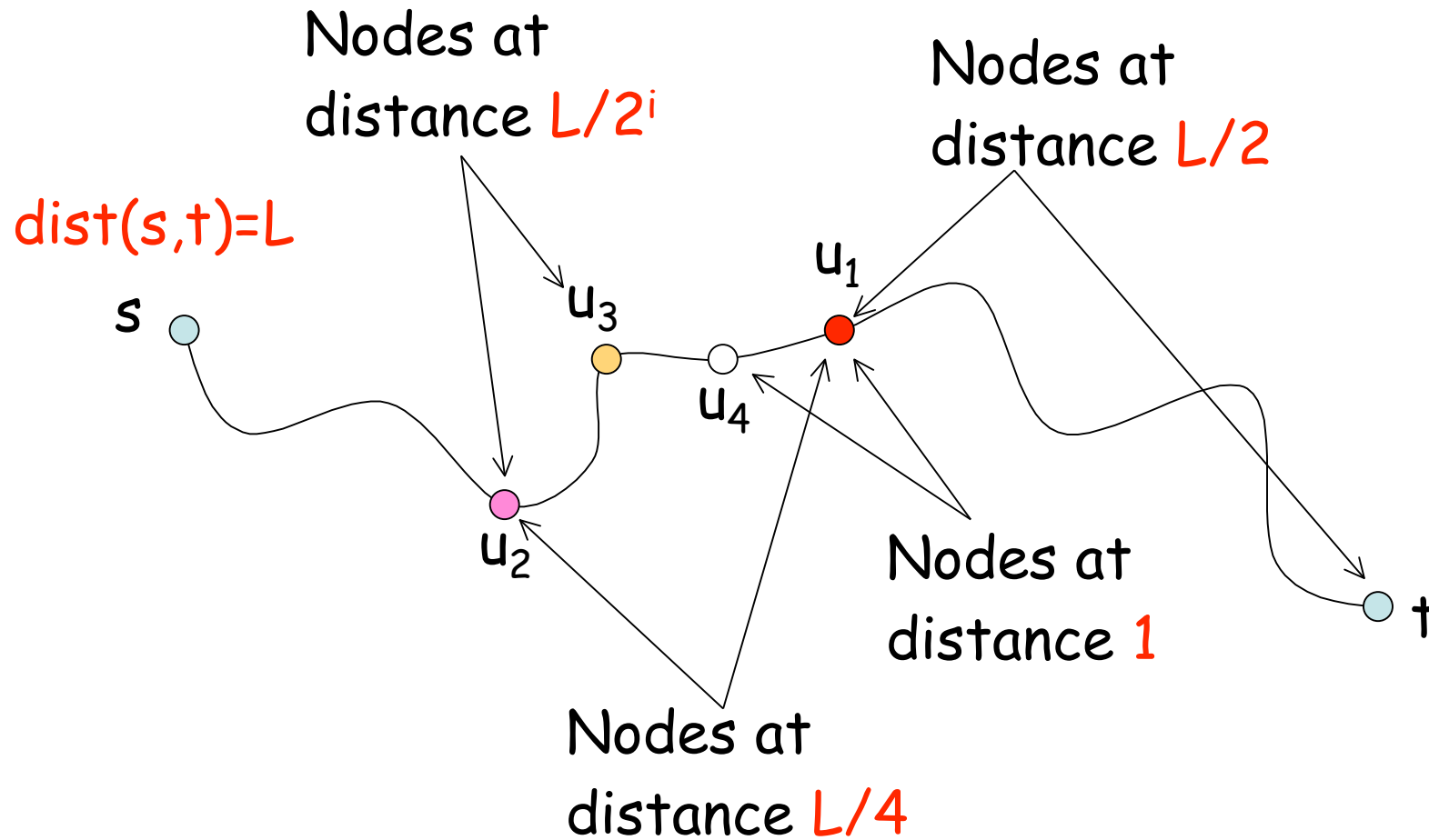
# **IV. Graph exploration**

## **IV.4. SL vs. L**

# Savitch's theorem

- USTCON:  
Is there a path from  $s$  to  $t$  in graph  $G$ ?
- Theorem (Savitch '70)  
USTCON can be solved deterministically  
in space  $O(\log^2 n)$
- Corollary:  $SL \subseteq L^2$

# Proof sketch



# Algorithm (rough)

Procedure:  $\text{path}(u,v,L)$

- if  $L=1$  then check whether  $\{u,v\} \in E(G)$
- else for  $w=1$  to  $n$  do
  - $\text{path}(u,w,L/2)$
  - $\text{path}(w,v,L/2)$

Program: call  $\text{path}(s,t,n)$

Analysis:

- depth of the recursion  $\leq \lceil \log_2 n \rceil$
- each level of recursion requires storing  $w \in \{1, \dots, n\}$ , on  $O(\log n)$  bits.

$$SL = L$$

- Nissan, Szemerédi, Wigderson '89:

$$SL \subseteq L^{3/2}$$

- Armoni, Ta-Shma, Wigderson, Zhou '00:

$$SL \subseteq L^{4/3}$$

- Reingold '04:

$$SL = L$$

# $\pi$ -consistent labeling

- Rotation map:  $\text{Rot}_G(v,i) = (w,j)$  iff



- Let  $\pi$  be a permutation of  $\{1, \dots, d\}$
- $\text{Rot}_G$  is  $\pi$ -consistent if:

$$\text{Rot}_G(v,i) = (w,j) \implies j = \pi(i)$$

# UTS for $\pi$ -consistent labeling

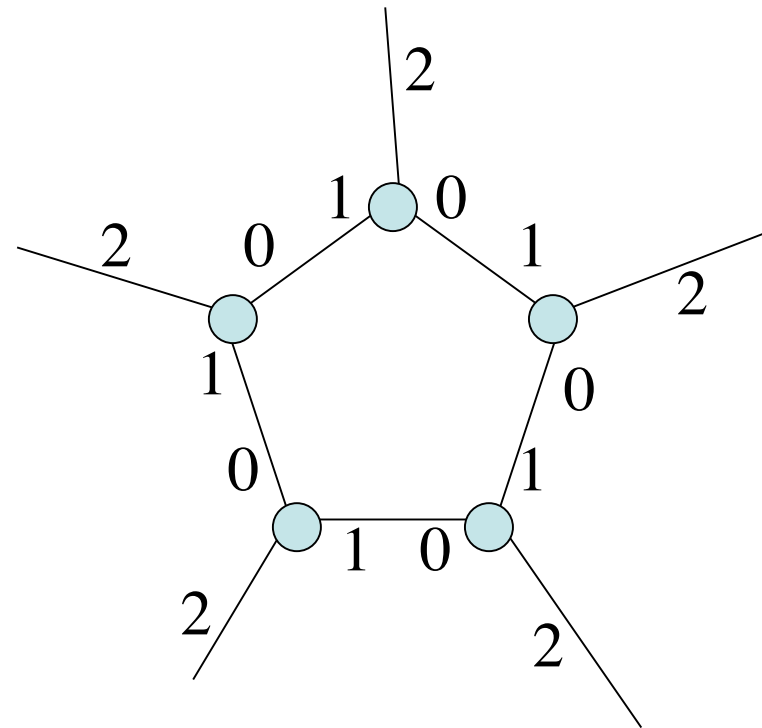
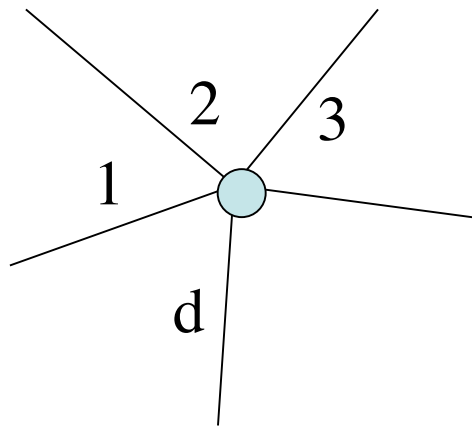
- Theorem (Reingold '04)

There exists a  $\log$ -space algorithm constructing UTS for  $\pi$ -consistently labeled graphs of  $G_{n,d}$

- Theorem (Koucky '03)

For any UTS of  $G_{nd,3}$  there exists a UXS for  $G_{n,d}$ , of same length, and constructible in  $\log$ -space.

# From $G_{n,d}$ to $G_{nd,3}$



# Proof of Koucky's theorem

- Let  $S=(p_1,p_2,\dots,p_r)$  be UTS for  $G_{nd,3}$

$$S=(S_1,2,S_2,2,S_3,2,\dots,2,S_k)$$

with  $S_i \in \{0,1\}^*$

- For  $U \in \{0,1\}^*$  let:

$$\text{eval}_d(U) = (|\{j/U_j=1\}| - |\{j/U_j=0\}|) \text{ mod } d$$

- Claim:  $(\text{eval}_d(S_1), \text{eval}_d(S_2), \dots, \text{eval}_d(S_{k-1}))$  is a UXS for  $G_{n,d}$  QED

# log-space constructible UXS

- Corollary

There exists a log-space algorithm that, given  $1^n$  and  $1^d$ , constructs a UXS for  $G_{n,d}$

- Proof:

Proof of Koucky's theorem uses graphs with  $\pi$ -consistent labelings

# **V. Mobile computing issues**

# Mobile computing

- Distributed computing:
  - Several (often very many) computing entities that can communicate to achieve some task (cf. Sergio's lecture)
- Mobile computing:
  - Several (often very many) **mobile** computing entities that can communicate to achieve some task

# **V. Mobile computing issues**

## **V.1. Terminating exploration**

# Terminating exploration

- The mobile machine has to traverse graphs **and stop** when the exploration is achieved
- In arbitrary graphs, terminating exploration requires a marker (e.g., a pebble)
  - because two cycles of different sizes cannot be distinguished

# Some results

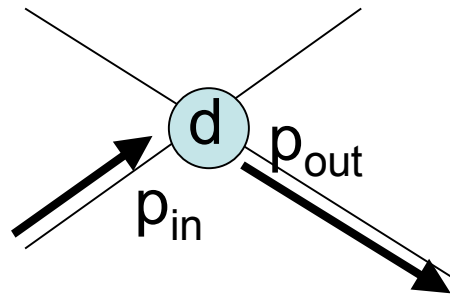
- Theorem (F., Ilcinkas, Rajsbaum, Tixeuil '05)  
A machine (provided with a pebble) achieving terminating exploration of all graphs of order  $\leq n$  needs  $\Omega(\log n)$  bits of memory.
- Theorem (Diks, F., Pelc, Kranakis '02)  
In trees, a machine (without pebble):
  - can explore all trees with constant memory
  - cannot perform terminating exploration of all trees of order  $\leq n$  with less than  $\Omega(\log \log \log n)$  bits of memory.
  - can perform exploration and return in all trees of order  $\leq n$  with  $O(\log^2 n)$  bits of memory (and needs  $\Omega(\log n)$  bits of memory for this task).

# **V. Mobile computing issues**

## **V.2. Exploring colored graphs**

# Exploration scheme

- An exploration scheme is a pair  $(\chi, A)$  where:
  - $\chi$  is a coloring algorithm for all graphs
  - $A$  is a finite automaton whose transition function takes into account the color of the nodes
- Transition function:  $f(s_i, p_{in}, d, c) = (s_j, p_{out})$   
where  $c = \chi(G, u)$  and  $u$  is the current node

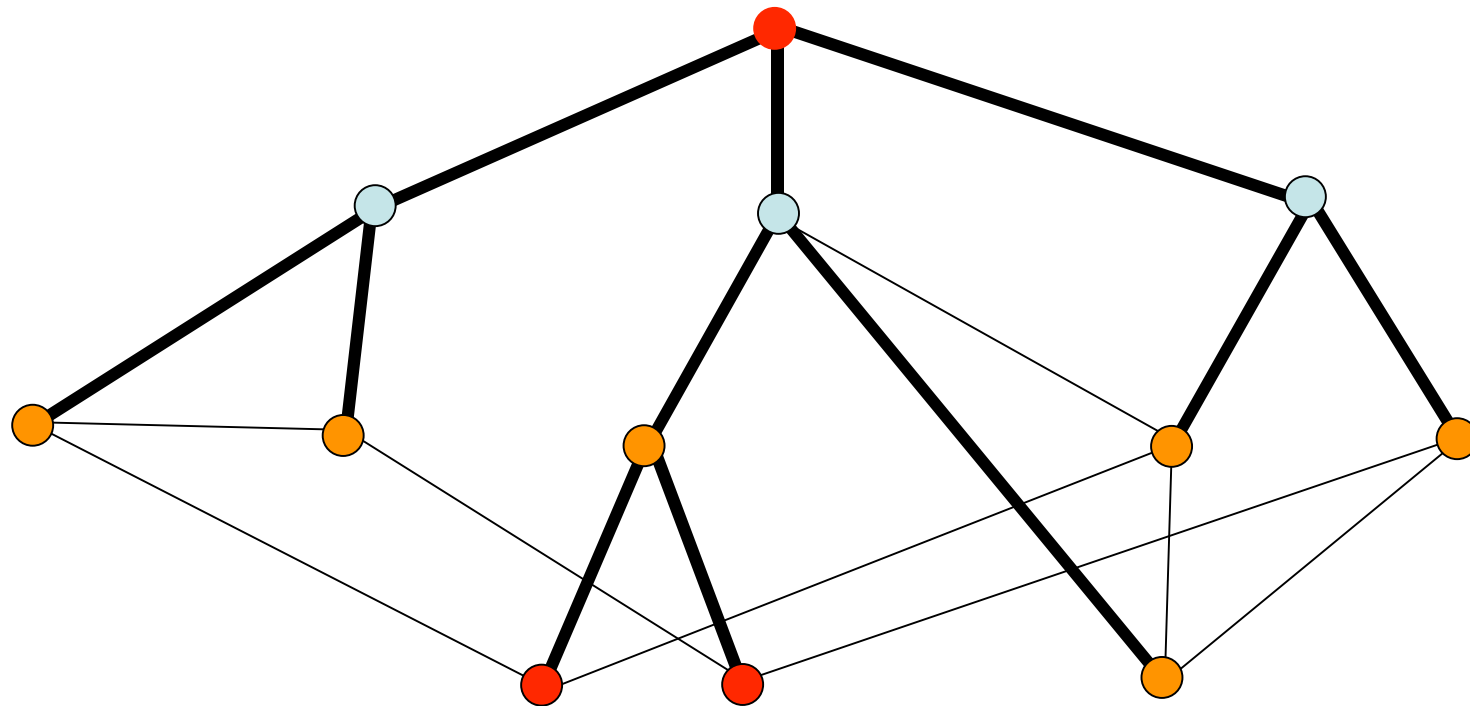


$d$  = node degree  
 $c$  = node color

# Universal exploration scheme

- A **universal** exploration scheme is an exploration scheme that explores all graphs
- Theorem (Cohen, F., Ilcinkas, Korman, Peleg '05)  
There exists a universal exploration scheme using three colors
- Remark: **1** color is not enough (cf. Rollik, Budach, etc.), and **3** colors are sufficient (cf. the theorem above).
- Open problem: what about **2** colors?

# Proof sketch



# **V. Mobile computing issues**

## **V.3. Black hole search**

# Black hole

- **Motivation:** modeling a harmful hostile program located at a node of a network
- **Definition:** A **black hole** is a node of an asynchronous network that destroys any mobile agent visiting it, without letting any trace of this destruction
- **Objective:** locating the black hole
- **Model:** **whiteboards** available at nodes, where agents can read and write messages

# Caution walk

- Three types of edges
  - **Unexplored**: no agent ever tried to traverse the edge
  - **Dangerous**: an agent is currently trying to traverse the edge, but it did not yet return
  - **safe**: an agent traversed the edge, returned, and has written "safe" on the whiteboard
- Caution walk:
  - Never traverse a dangerous edge

# Measures

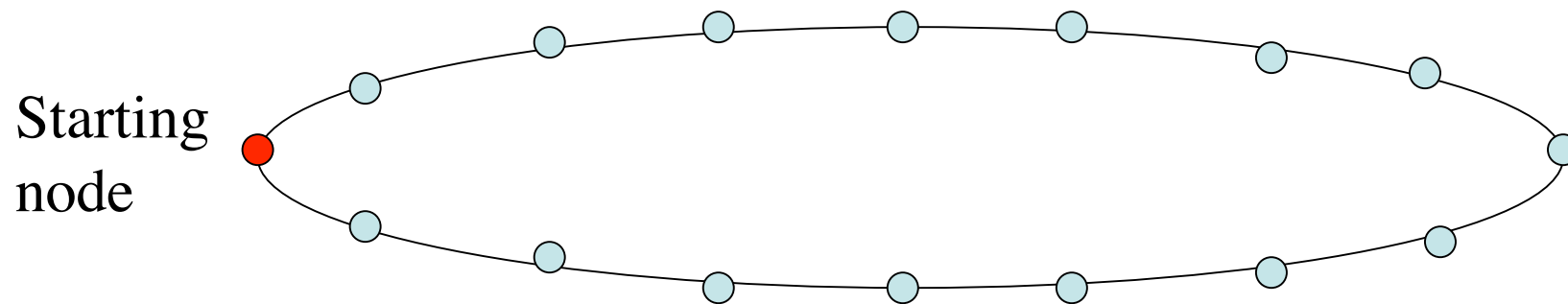
- Number of agents
- Number of moves (i.e., edge traversals)
- Different assumptions:
  - The agents have a map of the graph
  - They do not have a map but the edge-labeling has sens of direction
  - The graph is unknown, and the labeling is arbitrary

# Main result

**Theorem** (Dobrev, Flocchini, Prencipe, Santoro '02)

In a 2-connected network, two agents with a map of the network can find the black hole in  $O(n \log n)$  moves.

Proof: Case of the ring



Remark:  $\Omega(n \log n)$  moves are required

# **V. Mobile computing issues**

## **V.4. Micelaneous**

# Rendezvous

- Agents, initially located at different nodes of a network, aims at meeting at a same node (a.k.a. the *gathering* problem)
- Measures:
  - feasibility
  - number of moves (when feasible)

# Pattern formation

- A set of mobile entities spread out in the plane aim at forming some pattern
- Measures:
  - feasibility
  - time (when feasible)

# **VI. Open problems**

# Open problems

## Graph Searching

- For every connected graph,  $cs(G)/s(G) \leq 2$ ?
- Design a distributed algorithm for graph searching
- Is the decision problem " $cs(G) \leq k$ ?" in NP?

## Graph exploration

- Graph exploration with a  $\log$ -space memory machine
- Design a  $\log$ -space constructible **UTS**
- What are the relative powers of **UTS** and **UXS**?
- Space complexity of terminating exploration in trees
- Are **3** colors necessary to explore all graphs with a finite automaton?

**Thank you!**

Contact:

[pierre@lri.fr](mailto:pierre@lri.fr)

<http://www.lri.fr/~pierre>