Capture of an Intruder by Mobile Agents

Valia Mitsou

NTUA

Graph Searching: The Problem

It was first proposed by Breisch('67) and Parson('76).

- We are given a network whose links are all contaminated by a gas (or an invisible, arbitrarily fast fugitive)
- We are using a set of "searchers" (agents who are trying to clear the network)
- The goal is to use as few searchers as possible.

Motivation

- Maintaining security in network.
- Clearing a pipeline's network.
- Rescuing lost people in underground network.

Variants of the Problem

- Node Search (guard)
- Edge Search (sweep)
- . . . (Mixed Search, t-search)

Node-Search: Legal Operations

A search step:

- Place a searcher on a node
- Remove a searcher from a node

Cleaning an edge

- To clean a contaminated edge (u, v) the two endpoints u, v must be guarded.
- To prevent recontamination of the edge we must seal endpoints incident to contaminated links.

Cleaning an edge

- To clean a contaminated edge (u, v) the two endpoints u, v must be guarded.
- To prevent recontamination of the edge we must seal endpoints incident to contaminated links.

Cleaning an edge

- To clean a contaminated edge (u, v) the two endpoints u, v must be guarded.
- To prevent recontamination of the edge we must seal endpoints incident to contaminated links.

Cleaning an edge

- To clean a contaminated edge (u, v) the two endpoints u, v must be guarded.
- To prevent recontamination of the edge we must seal endpoints incident to contaminated links.

Edge-Search: Legal Operations

A search step:

- Place a searcher on a node
- Remove a searcher from a node
- Move a searcher along a link

Cleaning an edge

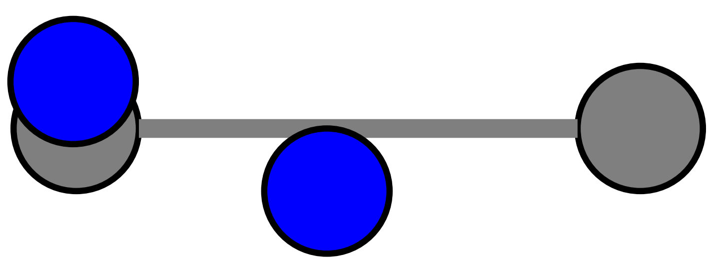
- To clean a contaminated edge (u, v) a searcher must traverse the edge from the one endpoint u to the other v.
- To prevent recontamination of the edge
  - ✓ Another searcher remains on u.
  - ✓ All other links incident to u are clear.

Cleaning an edge

- To clean a contaminated edge (u, v) a searcher must traverse the edge from the one endpoint u to the other v.
- To prevent recontamination of the edge
  - ✓ Another searcher remains on u.
  - ✓ All other links incident to u are clear.

Cleaning an edge

- To clean a contaminated edge (u, v) a searcher must traverse the edge from the one endpoint u to the other v.
- To prevent recontamination of the edge
  - ✓ Another searcher remains on u.
  - ✓ All other links incident to u are clear.



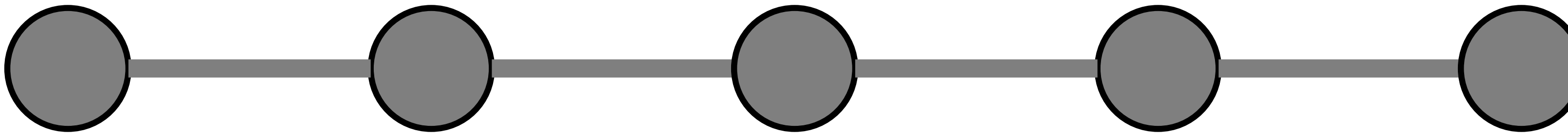
Cleaning an edge

- To clean a contaminated edge (u, v) a searcher must traverse the edge from the one endpoint u to the other v.
- To prevent recontamination of the edge
  - ✓ Another searcher remains on u.
  - ✓ All other links incident to u are clear.

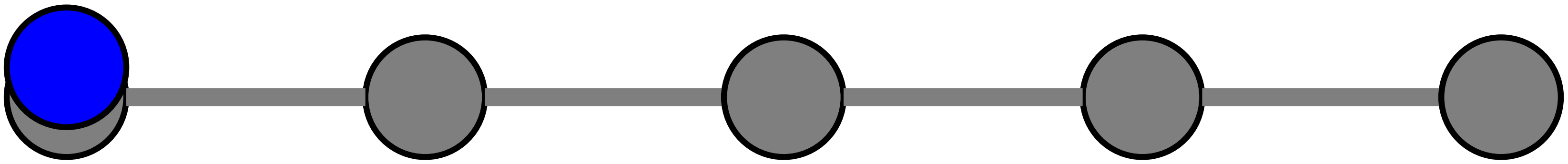
Search Number of a Graph

- *Search number* ($s(G)$) is the smallest number of searchers we can use to clear the network.
- A search strategy that uses $s(G)$ searchers is called *minimal*.

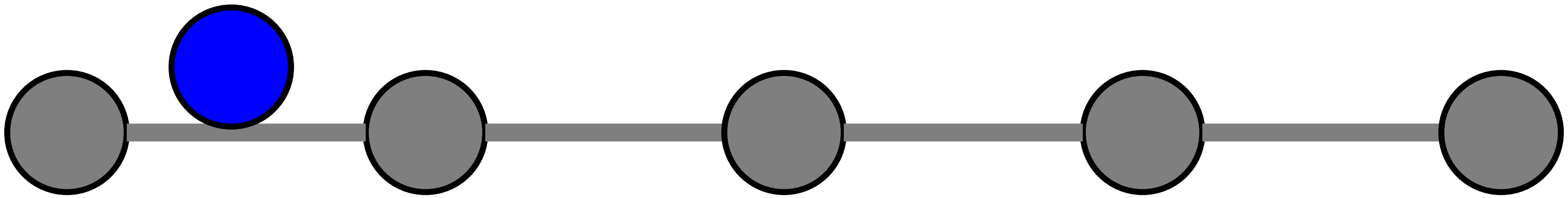
Example: Path



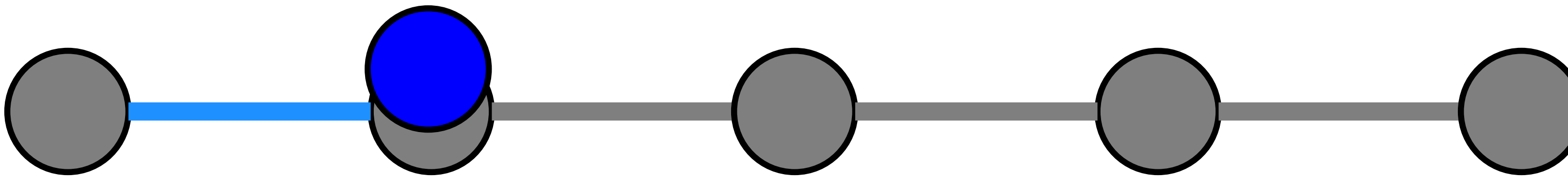
Example: Path



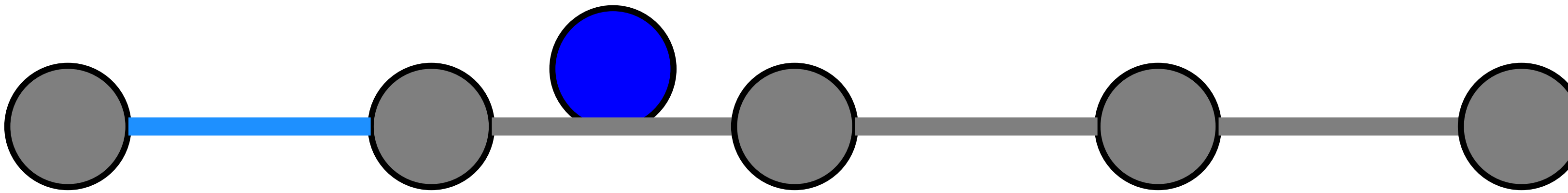
Example: Path



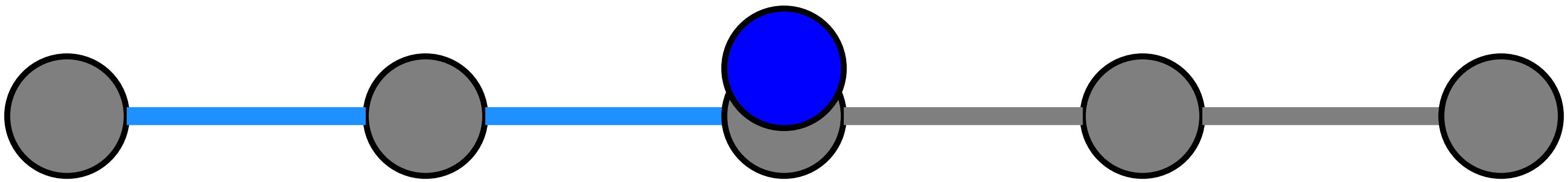
Example: Path



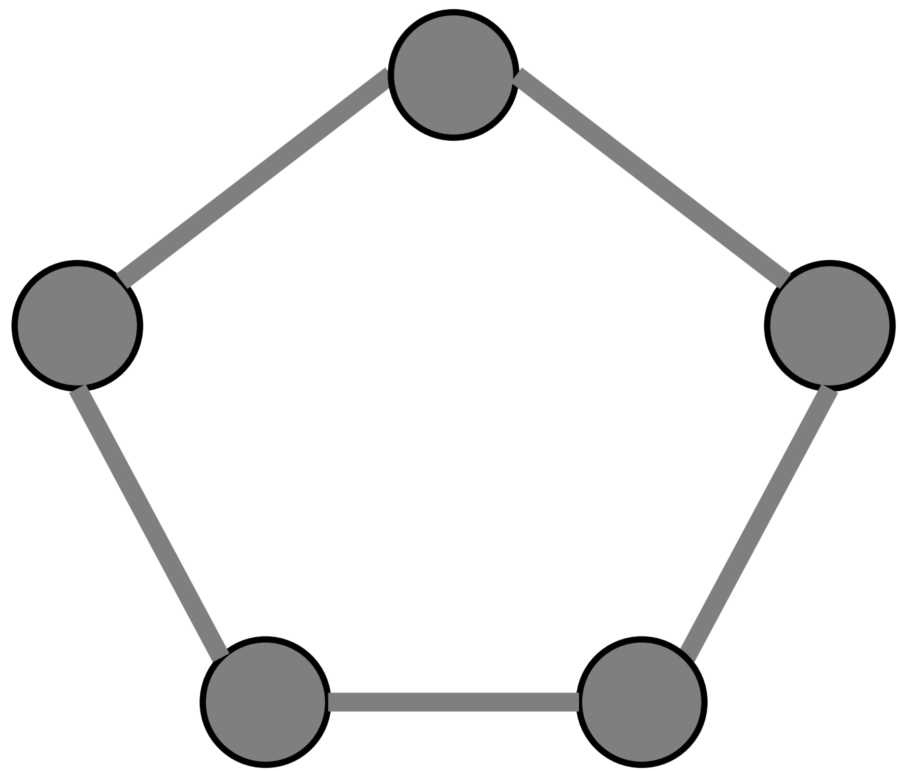
Example: Path



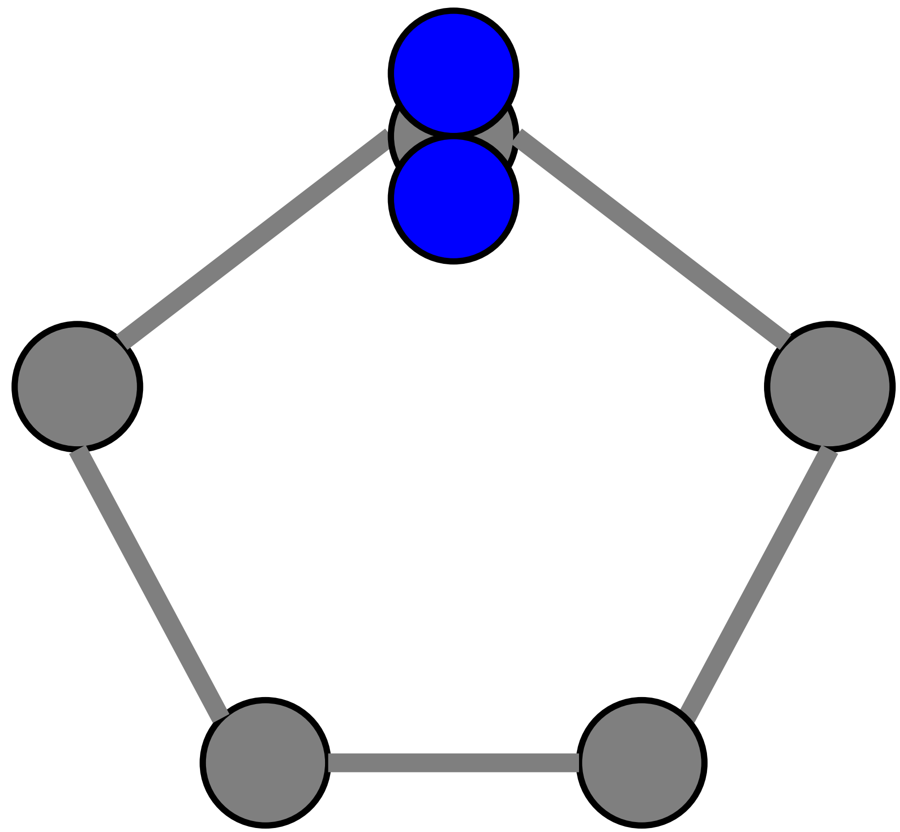
Example: Path



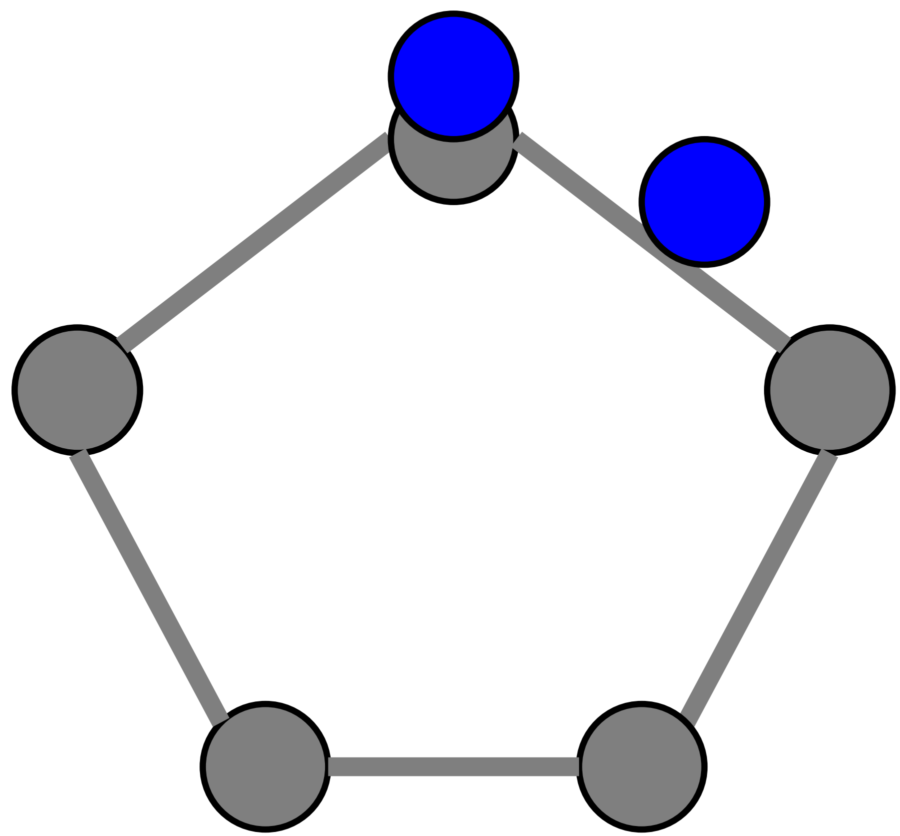
Example: Cycle



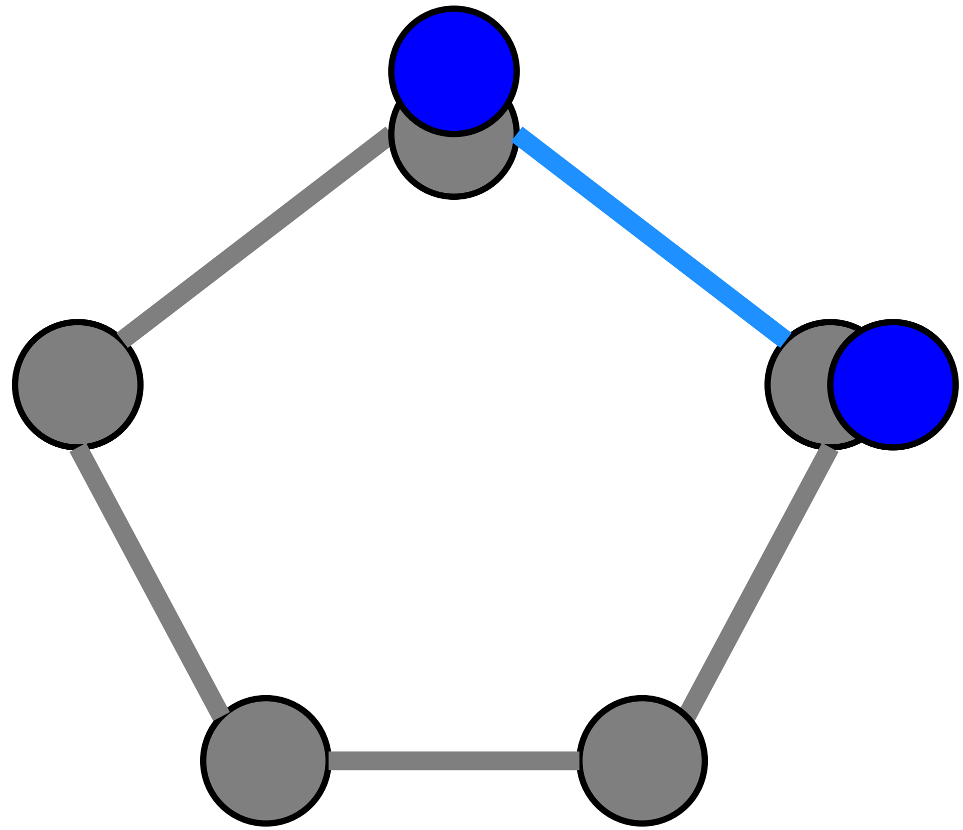
Example: Cycle



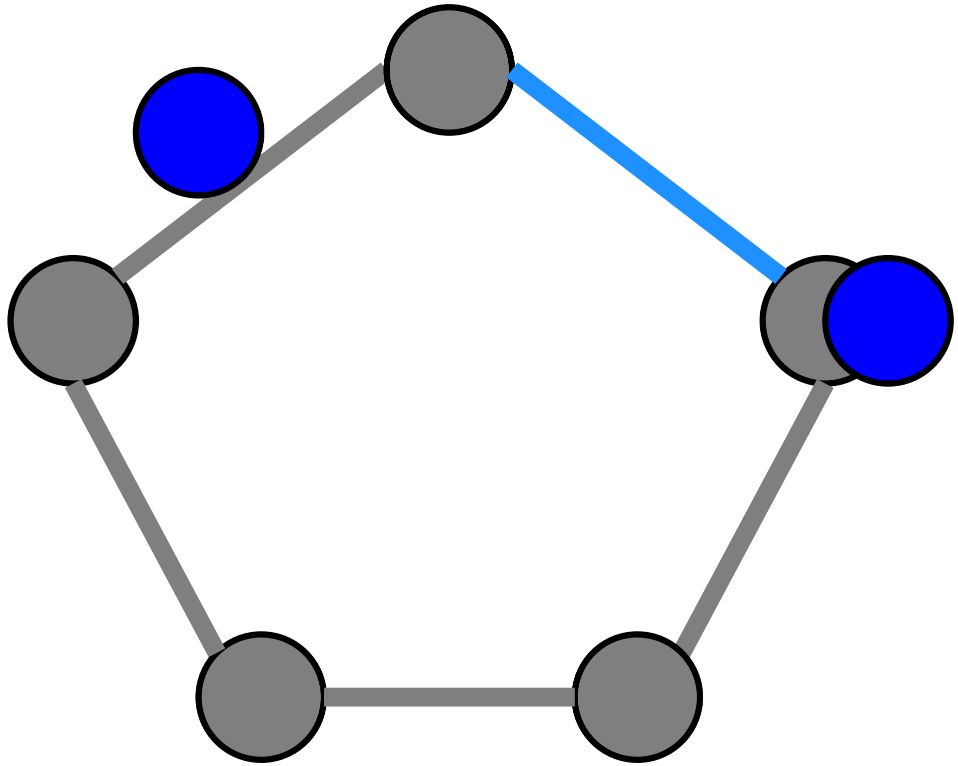
Example: Cycle



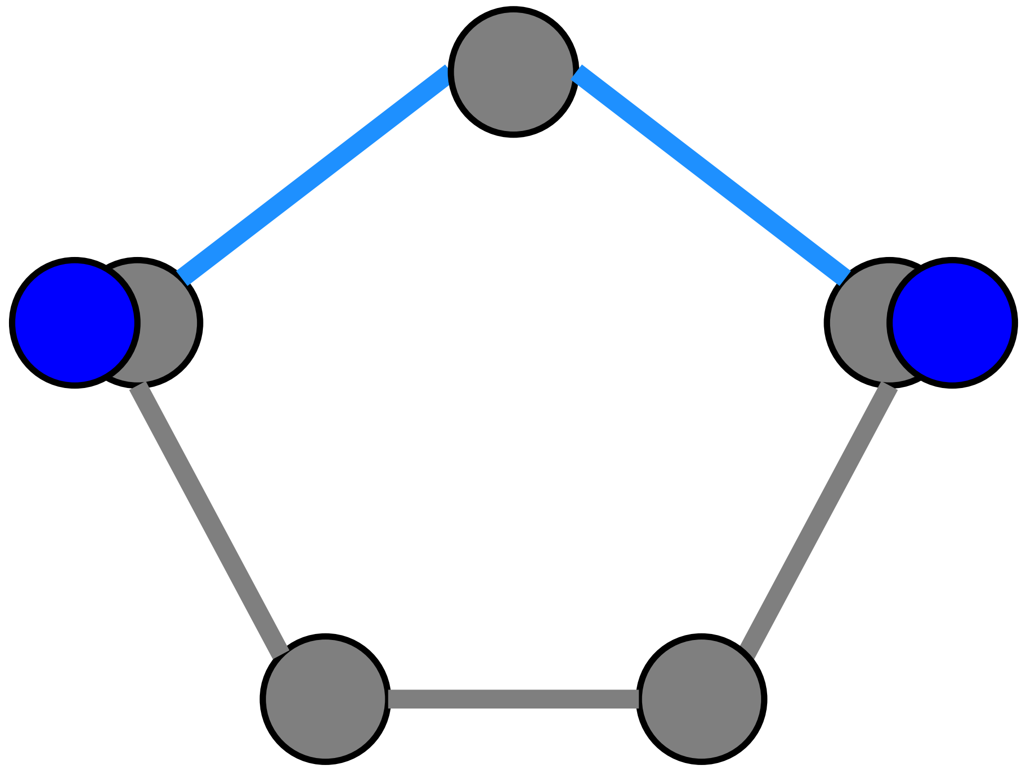
Example: Cycle



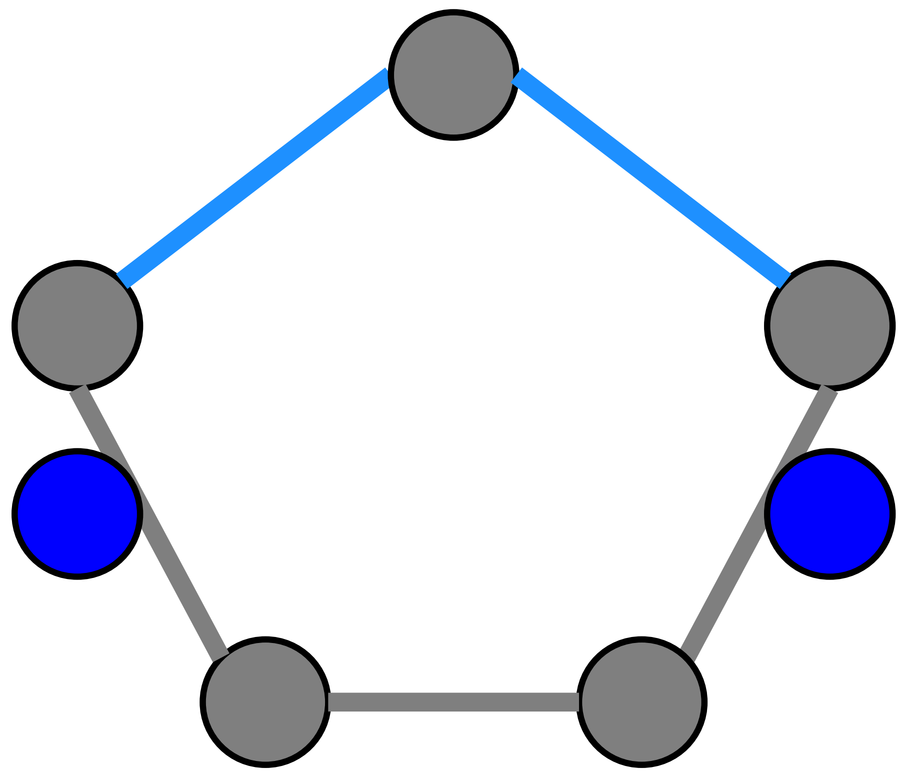
Example: Cycle



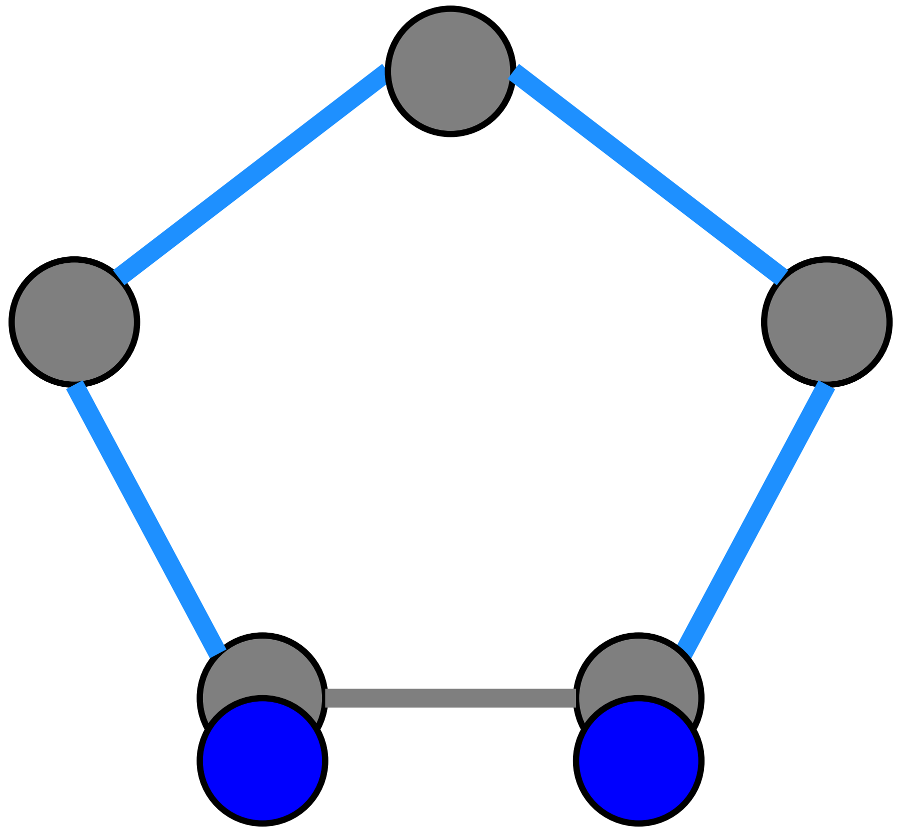
Example: Cycle



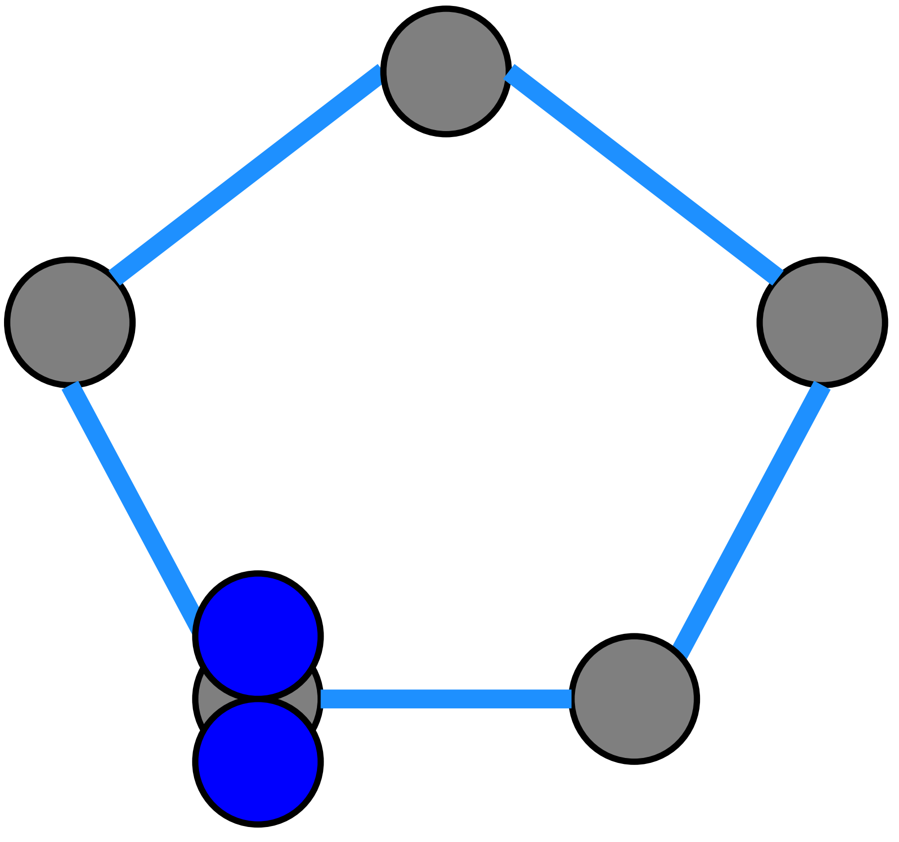
Example: Cycle



Example: Cycle



Example: Cycle



Definitions

- Weighted Search: Vertices and Edges have weights given by a function w.
  - ✓ To seal a node u we need $w(u)$ searchers.
  - ✓ To clean an edge e (edge-search) we need $w(e)$ searchers.
- Contiguous Search: The operation "remove a searcher" is illegal ($cs(G)$).
- Monotonicity: Once a link becomes clear, it cannot be contaminated ever again ($ms(G)$).
- Progressive Search: Exactly one edge becomes clear in every step.

Previous Work

- Megiddo, Hakimi, Garey, Johnson, Papadimitriou:
  - ✓ Graph Searching is NP-Complete.
  - ✓ $O(n)$ time algorithm for $s(T)$ (T is a tree).
  - ✓ $O(nlogn)$ time to find a minimal strategy.
- Barriere, Flocchini, Fraigniaud, Santoro:
  $O(n)$ time to find a minimal strategy (edge-search).
- Thilikos:
  $O(n)$ time to check whether a graph has $s(G) \leq 2$
  (by using graph minors).
- Lapaugh: Recontamination does not help to search a graph. (For any graph G, $ms(G) = s(G)$)

Equivalent Problems

- Node Search $s(G)$
- Path Decomposition $pw(G)$
- Interval Thickness $it(G)$
- Vertex Separation $vs(G)$

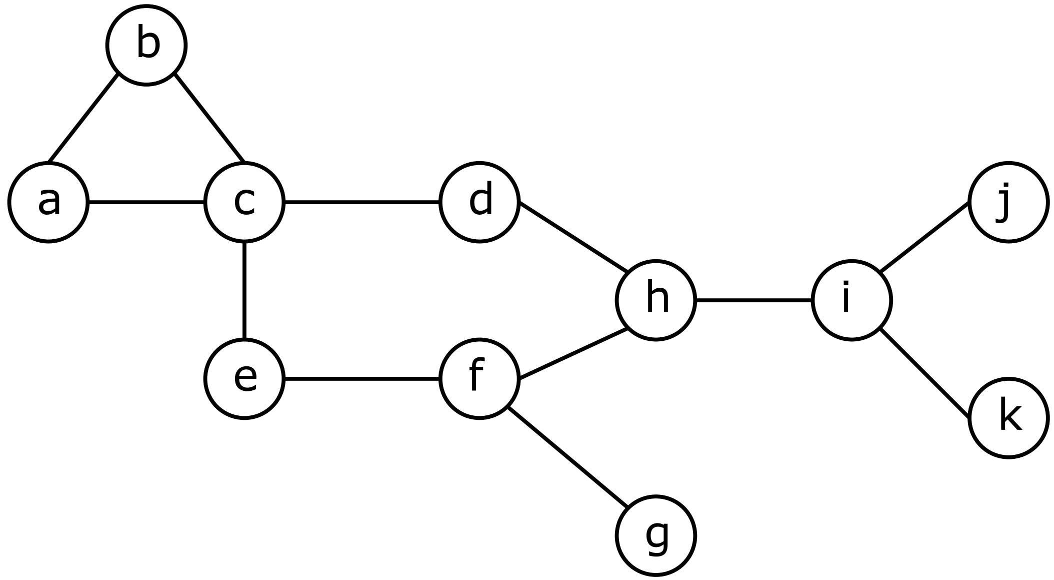
$$\leadsto vs(G) = pw(G) = s(G) - 1 = it(G) - 1$$

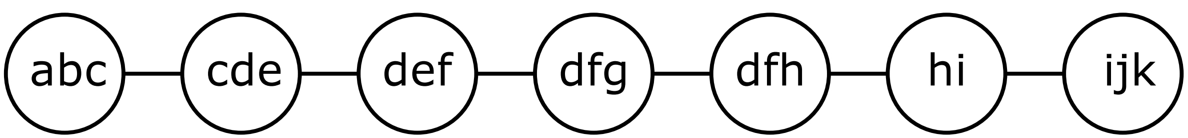
Path Decomposition

Definition: A path decomposition of a graph $G(V, E)$ is a pair $(\mathcal{X}, P)$, where $P(I, F)$ is a path and $\mathcal{X}$ is a family of “bags”.
Every $i \in I$ is associated with a bag $X_i \in \mathcal{X}$.
Every bag contains some nodes of V s.t.:

- $\bigcup_{i \in I} X_i = V$
- for all edges $\{u, w\} \in E$ $\exists i \in I$ with $u \in X_i$ and $w \in X_i$
- $\forall i, j, k \in I :$ if j is on the path from i to k in P, then $X_i \cap X_k \subseteq X_j$

Example for path decomposition



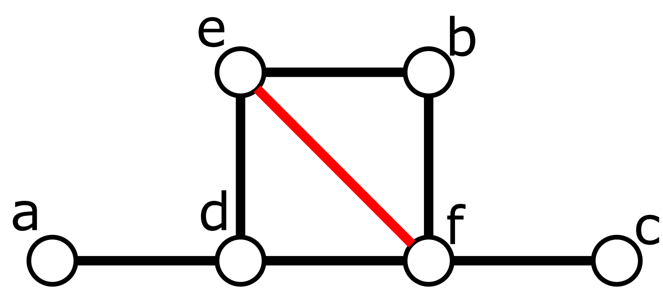


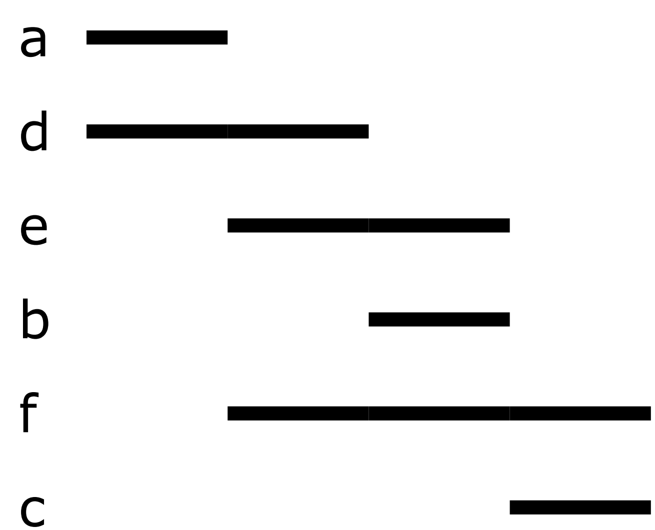
Pathwidth

- The *width* of a path decomposition $((I, F), \{X_i | i \in I\})$ is $\max_{i \in I} |X_i| - 1$.
- The *pathwidth* of a graph G is the minimum width over all tree decompositions of G.

Interval Thickness

Given a graph $G = (V, E)$, find an interval graph $G' = (V, E')$, $E \subseteq E'$, such that the maximum clique of G' is minimum.

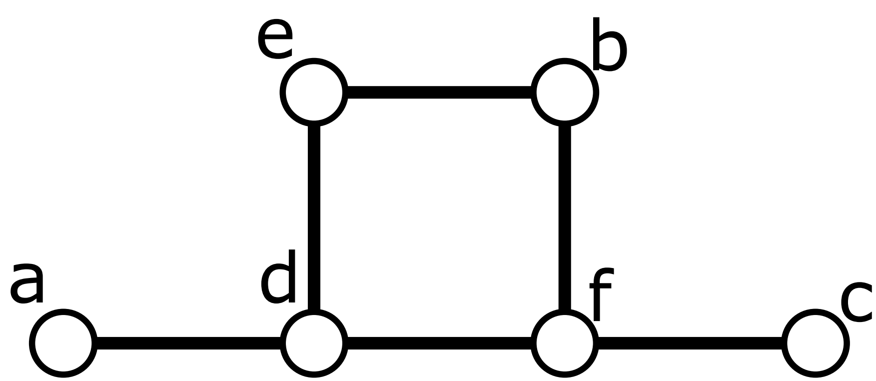


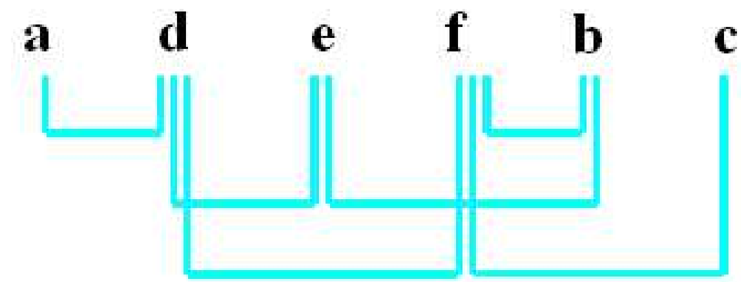


Vertex Separation

- $G = (V, E)$
- Linear layout: $L : V \to \{1, 2, \ldots, |V|\}$
- $V_L(i) = \{u | L(u) \leq i, \exists w, L(w) > i, (u, w) \in E\}$
- $vs_L(G) = max_i |V_L(i)|$
- $vs(G) = min_L vs_L(G)$

Example





- $V(1) = \{a\}$
- $V(2) = \{d\}$
- $V(3) = \{d, e\}$
- $V(4) = \{e, f\}$
- $V(5) = \{f\}$
- $V(6) = \emptyset$

Pathwidth $\approx$ Edge Search Number

- $pw(G) \leq s(G)$

Let ($s_1, s_2, \ldots, s_r$) be a monotone search strategy for a graph G.
Then ($X_1, X_2, \ldots, X_i, \ldots, X_r$) is a path decomposition, where: X_i contains the guarded vertices, and the edge that may have been cleared at step i.

Pathwidth $\approx$ Edge Search Number

- $s(G) \leq pw(G) + 2$

Let $(X_1, X_2, \ldots, X_i, X_{i+1} \ldots, X_r)$ be a path decomposition of a graph G with width $pw(G)$. At step i, the graph induced by $\bigcup_{k<i} X_k$ is cleared;

1. Place (at most) $pw(G) + 1$ searchers on vertices in X_i.
2. One other searcher clears edges in X_i.
3. Remove the searchers on vertices in $X_i \setminus X_{i+1}$.

Barriere, Flocchini, Fraigniaud, Santoro

- Theorem 1: For any tree T there is a monotone contiguous search strategy using $cs(T)$ searchers. Furthermore all searchers can be initially placed in the same node.
- Theorem 2: The contiguous search number and minimal monotone contiguous search for trees can be found serially in $\Theta(n)$ time and distributively with $\Theta(n)$ messages.
- Theorem 3: For every $n > 1$ the largest contiguous search number of n-node trees satisfies $\lfloor \log_2 n \rfloor - 1 \leq cs(n) \leq \lfloor \log_2 n \rfloor$ (In contrast for non-contiguous search $\sim \log_3 n$ searchers suffice).

Theorem 2

- Suppose that the tree T is rooted with root x ($T = T_x$).
- It can be shown that the number of searchers needed for T_x is $cs(T_x) = \max\{cs(T_{x_1}), cs(T_{x_2}) + 1\}$, where $x_1, x_2, \ldots x_k$ are the children of x in decreasing order of $cs(T_{x_i})$.

Compute $cs(T_x)$

- Start from the leaves ($cs(T_l) = 1$)
- Continue to the parents y computing $cs(T_y)$ with the previous type.
- Compute $cs(T_x)$.

Compute $cs(T)$

- Compute $cs(T) = \min_x cs(T_x)$
- This requires $O(n^2)$ time.
- It can be shown that we can compute all $cs(T_x)$ in $O(n)$ time.
- Just find the minimum $cs(T_x)$ (in $O(n)$ time)

Find a Minimal Strategy

- Order the children in the way mentioned before.
- Place $cs(T)$ searchers on x.
- Traverse T_x in pre-order with the simple rule: When moving from a node y to one of its children z (or backwards) transfer $cs(T_z)$ searchers.

Distributed Search

A node can be ready, active or done.

- In the beginning every node is ready.
- Every leaf l sends $cs(T_l) = 1$ to its neighbor and becomes active.
- Every other ready node y waits to receive $d - 1$ messages. Then computes $cs'(T_y)$ and sends it to its parent. Then becomes active.
- Every active node that receives the message from the last neighbor computes the final $cs(T_y)$ and becomes done.

Communication Complexity

- $\Theta(n)$ messages are sent to compute $cs(T_x)$ in every x.
- With a convergecast a middle node computes the minimum among them and sends it back to the other nodes ($\Theta(n)$ messages).

THE END!!!