Connected Treewidth and Connected Graph Searching

Pierre Fraigniaud[1] Nicolas Nisse[2]

CNRS, LRI, Université Paris-Sud, France.

LRI, Université Paris-Sud, France.

LATIN 05, March 21*th*, 2006

Graph Searching

Goal

In a contaminated network,

- an invisible omniscient arbitrary fast fugitive ;
- a team of searchers ;

We want to find a strategy that catch the fugitive
using the fewest searchers as possible.

Motivations

- network security, speleological rescue...
- game related to well known graphs'parameters : treewidth and pathwidth ;

Graph Searching

Goal

In a contaminated network,

- an invisible omniscient arbitrary fast fugitive ;
- a team of searchers ;

We want to find a strategy that catch the fugitive
using the fewest searchers as possible.

Motivations

- network security, speleological rescue...
- game related to well known graphs'parameters : treewidth and pathwidth ;

Search Strategy, Parson. [GTC,1978]

Sequence of three basic operations,. . .

1. Place a searcher at a vertex of the graph ;
2. Move a searcher along an edge of the graph ;
3. Remove a searcher from a vertex of the graph.

. . . that must result in catching the fugitive

The fugitive is caugth when it meets a searcher at a vertex or in an edge of the graph.

We want to minimize the number of searchers.

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

Search Strategy, Parson. [GTC,1978]

Sequence of three basic operations,. . .

1. Place a searcher at a vertex of the graph ;
2. Move a searcher along an edge of the graph ;
3. Remove a searcher from a vertex of the graph.

. . . that must result in catching the fugitive

The fugitive is caugth when it meets a searcher at a vertex or in an edge of the graph.

We want to minimize the number of searchers.

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

Search Strategy, Parson. [GTC,1978]

Sequence of three basic operations,. . .

1. Place a searcher at a vertex of the graph ;
2. Move a searcher along an edge of the graph ;
3. Remove a searcher from a vertex of the graph.

. . . that must result in catching the fugitive

The fugitive is caugth when it meets a searcher at a vertex or in an edge of the graph.

We want to minimize the number of searchers.

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

Search Strategy, Parson. [GTC,1978]

Sequence of three basic operations,...

1. Place a searcher at a vertex of the graph ;
2. Move a searcher along an edge of the graph ;
3. Remove a searcher from a vertex of the graph.

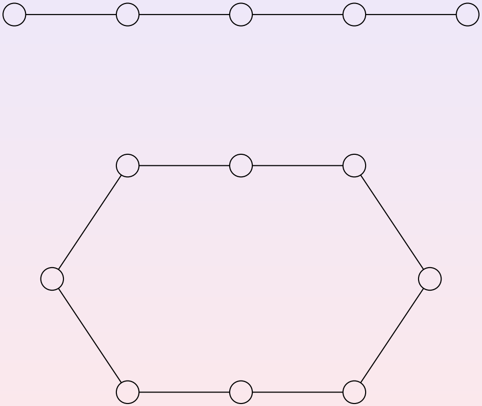
...that must result in catching the fugitive

The fugitive is caugth when it meets a searcher at a vertex or in an edge of the graph.

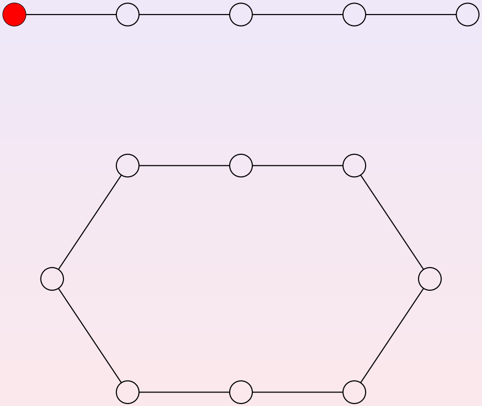
We want to minimize the number of searchers.

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

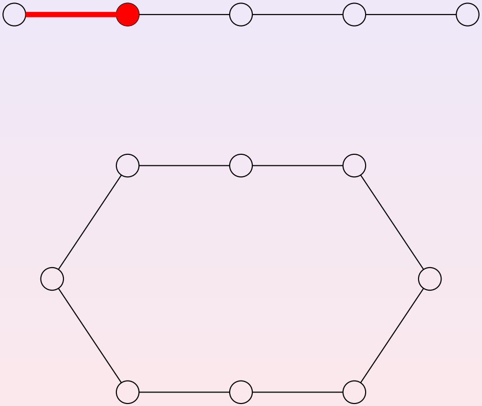
Simple Examples : Path and Ring



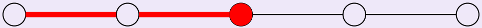
Simple Examples : Path and Ring

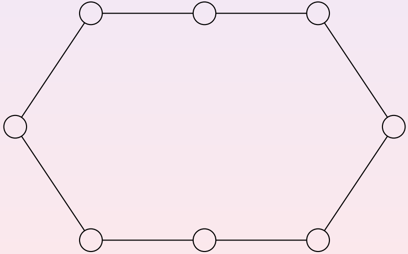


Simple Examples : Path and Ring

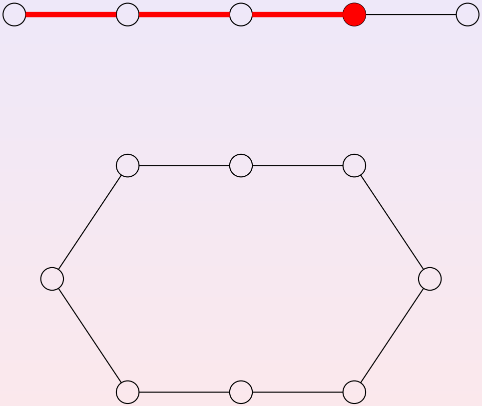


Simple Examples : Path and Ring



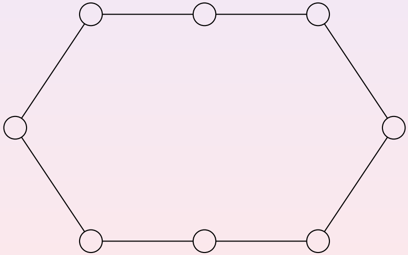


Simple Examples : Path and Ring



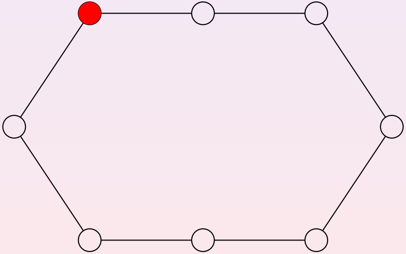
Simple Examples : Path and Ring

s(Path)=1



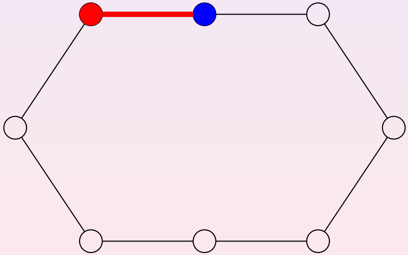
Simple Examples : Path and Ring

s(Path)=1



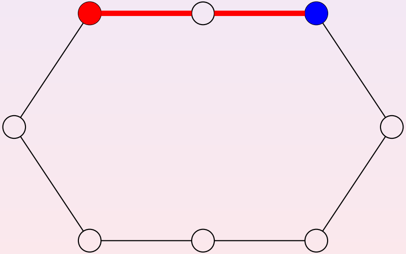
Simple Examples : Path and Ring

s(Path)=1



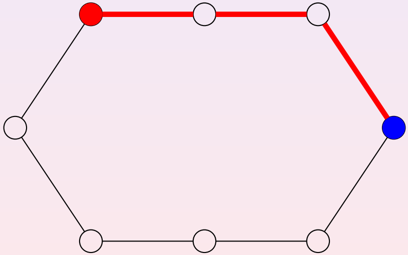
Simple Examples : Path and Ring

s(Path)=1



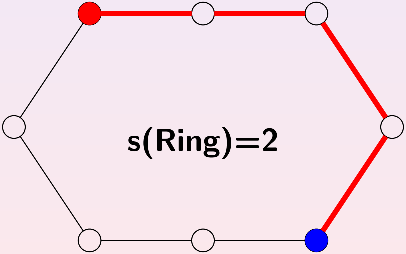
Simple Examples : Path and Ring

s(Path)=1

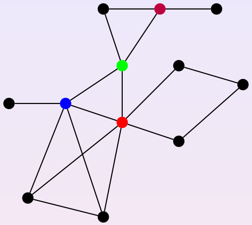


Simple Examples : Path and Ring

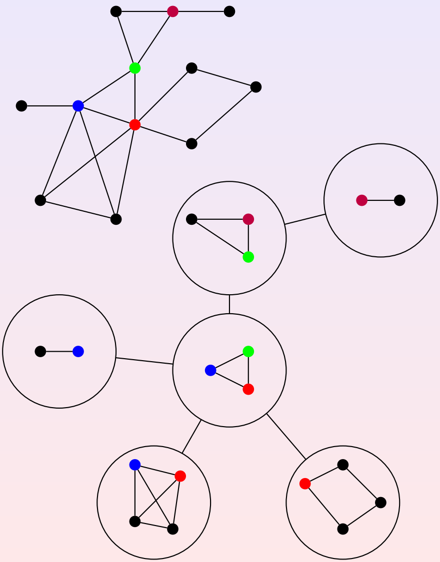
s(Path)=1



Tree and Path Decompositions

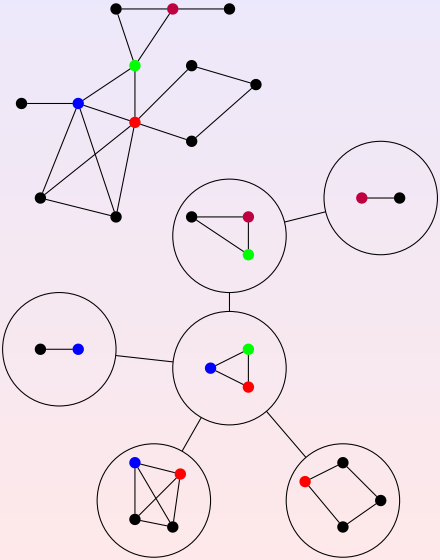


Tree and Path Decompositions



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

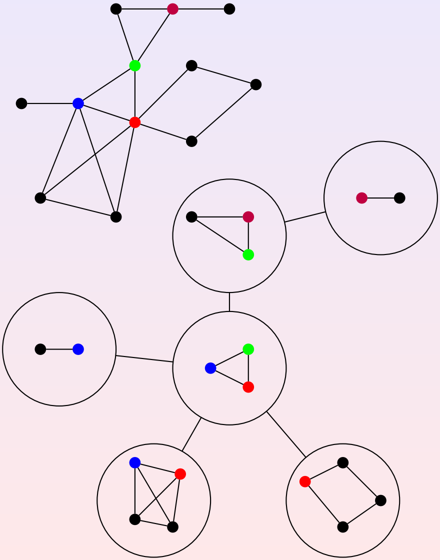
Tree and Path Decompositions



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

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

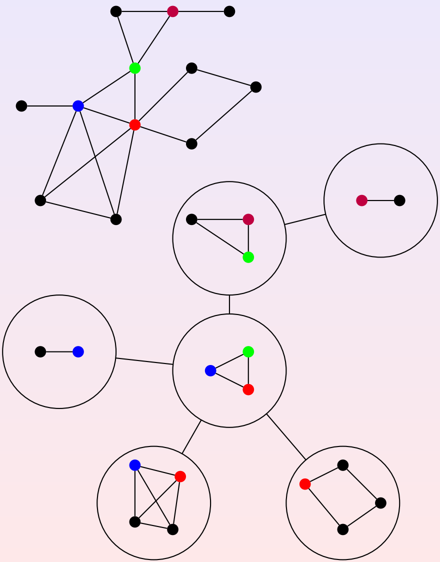
Tree and Path Decompositions



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 ;

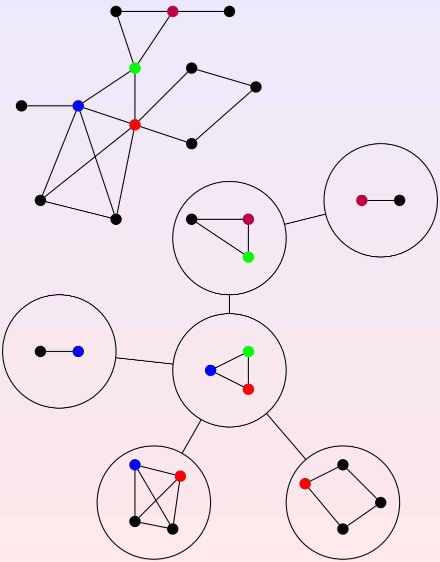
Tree and Path Decompositions



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.

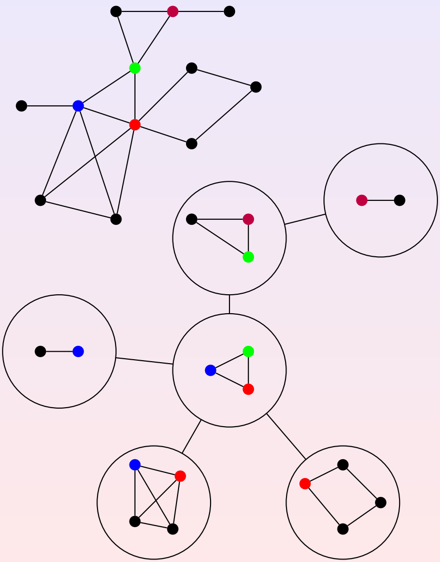
Tree and Path Decompositions



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



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

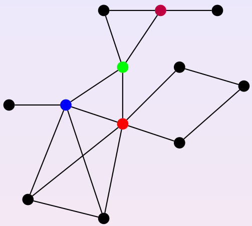
- every **vertex** of G is at least in one bag ;
- both ends of an **edge** of G are at least in one bag ;
- For any vertex of G, all bags that contain it, form a **subtree**.

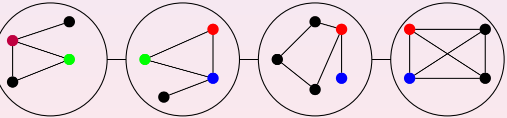
Width = Size of largest Bag -1

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

Relationship between search number and pathwidth

Ellis, Sudborough and Turner. [Inf. Comput.,1994]

For any graph G, $vs(G) \leq s(G) \leq vs(G) + 2$

Kinnersley. [IPL.,1992]

For any graph G, $vs(G) = pw(G)$

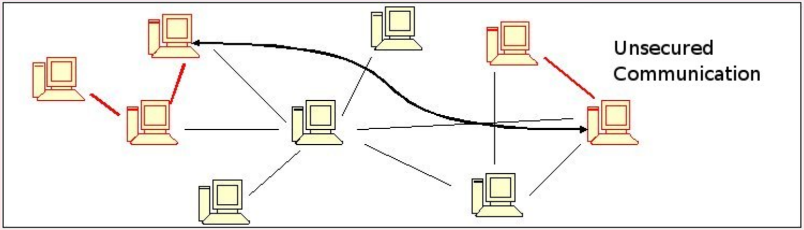
For any n-node graph G :

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

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.



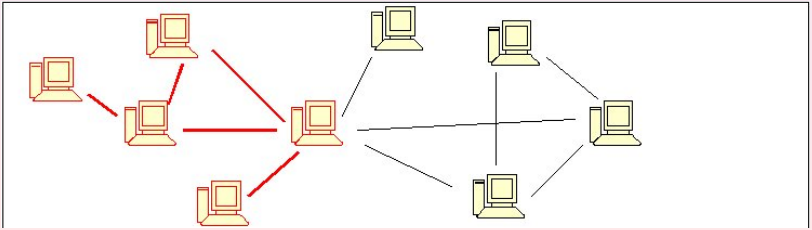
Connected Graph Searching

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

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.



Cost of connectedness : case of trees

Barrière, Flocchini, Fraigniaud and Santoro. [SPAA, 2002]

Linear Algorithm

Barrière, Fraigniaud, Santoro and Thilikos. [WG, 2003]

For any tree T, $s(T) \leq cs(T) \leq 2\ s(T) - 2$.
Moreover, these bounds are tight.

Cost of connectedness : case of arbitrary graphs

Seymour and Thomas. [Combinatorica, 1994]

Bond Carving

Fomin, Fraigniaud and Thilikos. [Technical repport, 2004]

- Using a branch-decomposition, polynomial constructive algorithm that computes a connected search strategy.
- For any connected graph G,
  $cs(G) \leq s(G)\ (2 + \log |E(G)|)$.

Connected Treewidth

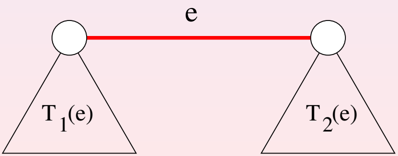
Connected e-cut of a tree-decomposition (T, X)

The edge e is said connected if both $G[T_1(e)]$ and $G[T_2(e)]$ induced connected subgraphs of G.

Connected tree-decomposition (T, X)

For any $e \in E(T)$, e is connected.

Connected treewidth, **ctw**(G)



Connected Treewidth

For any connected graph G, $ctw(G) = tw(G)$

Golumbic. Algorithmic graph theory and perfect graphs

A "clique tree" of a minimal triangulation H of a connected graph G is an optimal tree-decomposition of G.

Parra and Scheffler. [DAM 1997]

A "clique tree" of a minimal triangulation H of a connected graph G is a connected tree-decomposition of G.

Results (1)

Theorem 1 : new proof

For any connected graph G, $ctw(G) = tw(G)$

Constructive proof

Given a tree-decomposition of width $\leq k$ of a connected graph G with n vertices, our algorithm computes a connected tree-decomposition of width $\leq k$ of G, in time $O(n.k^3)$.

Results (2)

Theorem 2

For any connected graph G, $\mathbf{cs}(G) \leq \mathbf{s}(G)\ (1 + \log_2 |V(G)|)$.

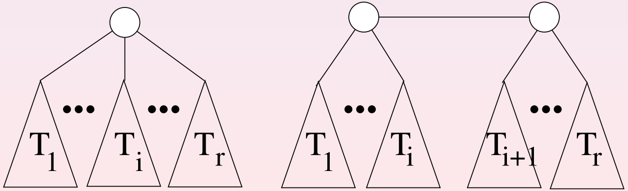
Constructive proof

Given a tree-decomposition of a graph G, our algorithm computes a connected search strategy for G, using at most $\mathbf{tw}(G) \log |V(G)|$ searchers, in polynomial time.

Squetch of the proof of Theorem 2

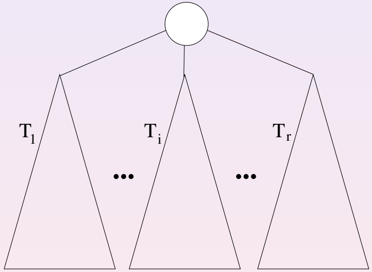
For any connected n-node graph G, $\mathbf{cs}(G) \leq \mathbf{s}(G)\,(1 + \log_2 n)$

- proof by induction on n
- Robertson and Seymour. Graph Minors II. Algorithmic Aspects of Tree-Width. J. of Alg 7, 1986.
  - For any tree-decomposition (T, X) of a n-node graph G, there are one (or two adjacent vertices) of T such that :
  - for any $1 \leq j \leq r$, $|G[T_j]| \leq n/2$



Squetch of the proof of Theorem 2

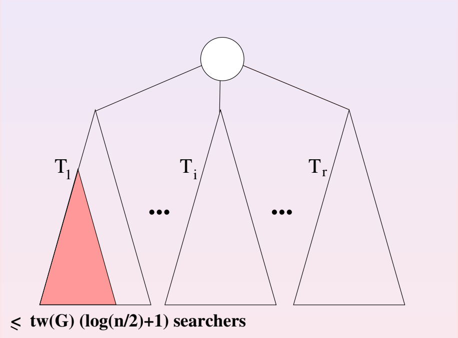
Starting from a connected tree-decomposition of G



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

Squetch of the proof of Theorem 2

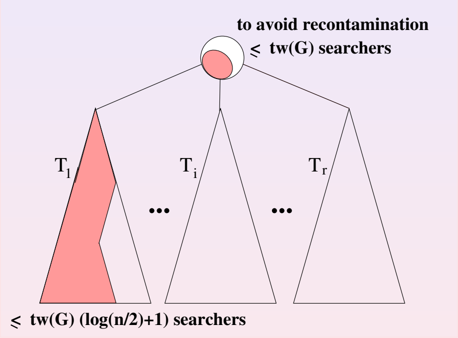
Starting from a connected tree-decomposition of G



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

Squetch of the proof of Theorem 2

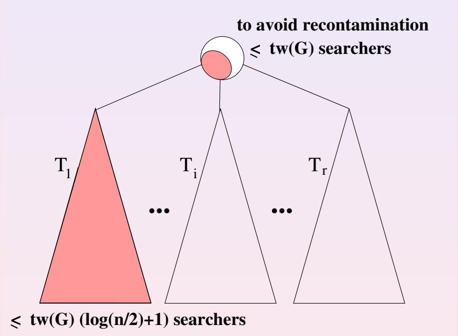
Starting from a connected tree-decomposition of G



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

Squetch of the proof of Theorem 2

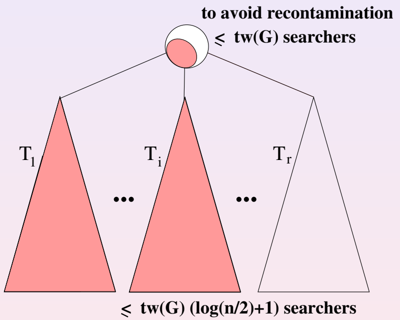
Starting from a connected tree-decomposition of G



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

Squetch of the proof of Theorem 2

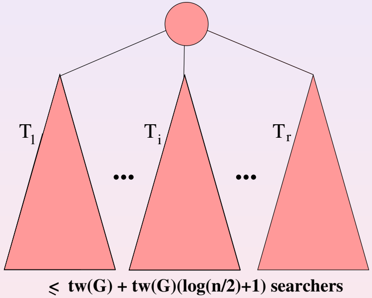
Starting from a connected tree-decomposition of G



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

Squetch of the proof of Theorem 2

Starting from a connected tree-decomposition of G



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

Conclusion and Further Work

Cost of connectedness

- new upper bound of the ratio $\mathbf{cs}(G)/\mathbf{s}(G)$
- constructive algorithm

Open problems

- What is the optimal bound ?
  In trees : $\mathbf{cs}(T)/\mathbf{s}(T) \leq 2$ and this bound is tight [Barrière et al.].
  If the fugitive is visible : $\mathbf{cs}(G)/\mathbf{s}(G) \leq \log n$ and this bound is tight.
- Is the problem of computing $\mathbf{cs}(G)$ NP-complete ?
  It is known to be NP-hard.