Monotonicity of Non deterministic Graph Searching

Frédéric Mazoit[1] Nicolas Nisse[2]

LABRI, University Bordeaux I, France.
LRI, University Paris-Sud, France.

1st Workshop on GRAph Searching, Theory and Applications, October 10, 2006

Introduction

Non deterministic graph searching

[Fomin, Fraigniaud, Nisse, 2005]
Parametrized variant that unifies visible and invisible graph searching.

Monotone non deterministic graph searching : interpretation in terms of graph decomposition

Does recontamination help ?

Our result : Non deterministic graph searching is monotone.

Introduction

Non deterministic graph searching

[Fomin, Fraigniaud, Nisse, 2005]
Parametrized variant that unifies visible and invisible graph searching.

Monotone non deterministic graph searching :
interpretation in terms of graph decomposition

Does recontamination help ?

Our result : Non deterministic graph searching is monotone.

Non deterministic Graph Searching

An omniscient arbitrary fast invisible fugitive runs at the vertices of the graph.
The searchers cannot see the fugitive, however :
An Oracle permanently knows the position of the fugitive.

The searchers can perform a query to the oracle :

Answer of the oracle :

The connected component of the contaminated part of the graph, where the fugitive is currently standing.

Non deterministic Graph Searching

An omniscient arbitrary fast invisible fugitive runs at the vertices of the graph.
The searchers cannot see the fugitive, however :
An Oracle permanently knows the position of the fugitive.

The searchers can perform a query to the oracle :

Answer of the oracle :

The connected component of the contaminated part of the graph, where the fugitive is currently standing.

Non deterministic Search Strategy

three basic operations :

- Place a searcher ;
- Remove a searcher ;
- Perform a query to the oracle.

The searchers aim at catching the fugitive.

The fugitive is caught when it occupies the same vertex as a searcher and it has no way to escape.
An edge is cleared when both its ends are occupied.

q-limited non deterministic search number

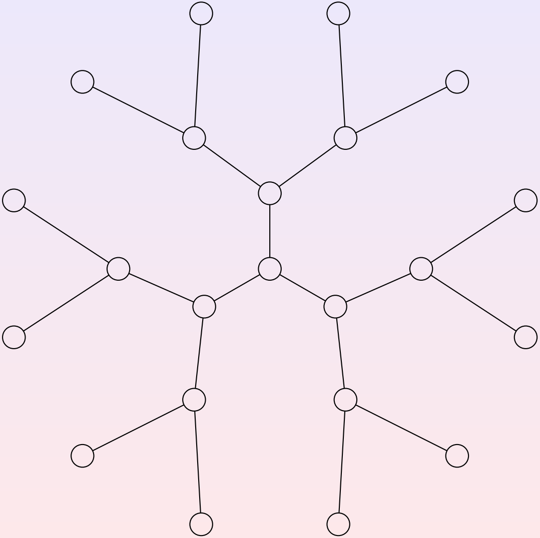
Tradeoff number of searchers / number of query steps

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

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

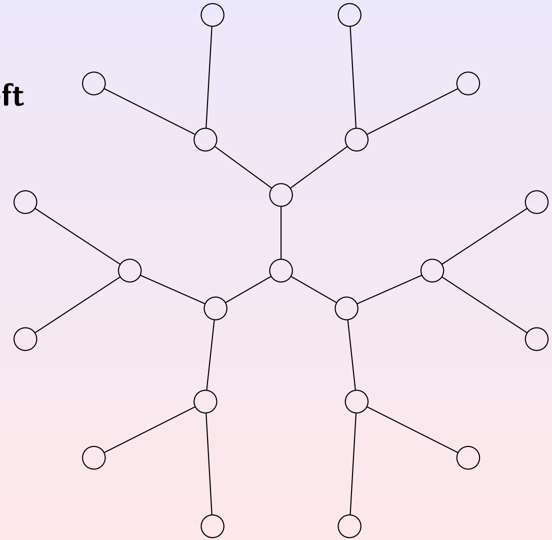
Exemple with q=2 :

$s_0(T)=3$



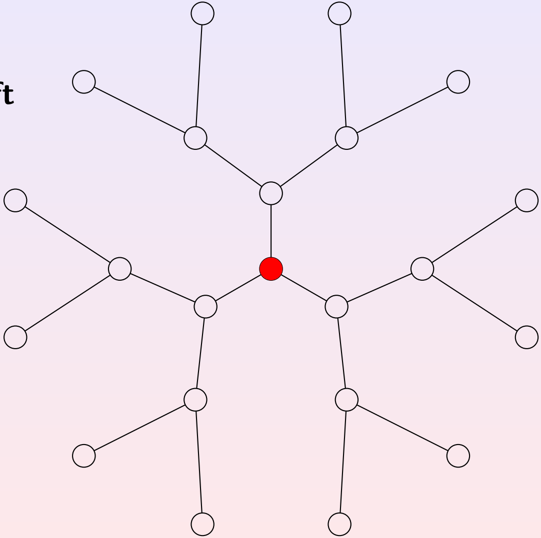
Exemple with q=2 :

2 queries left



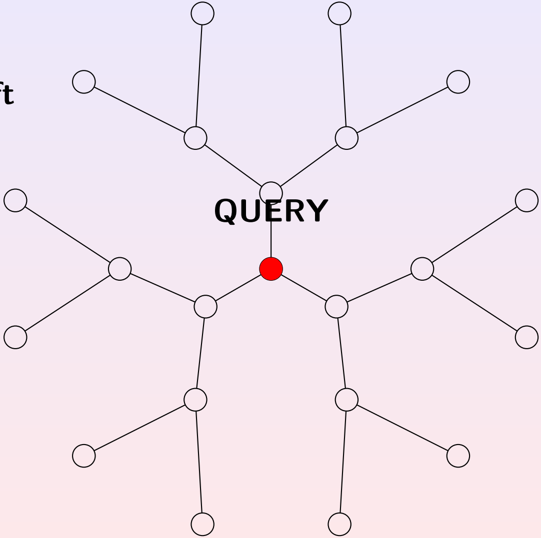
Exemple with q=2 :

2 queries left



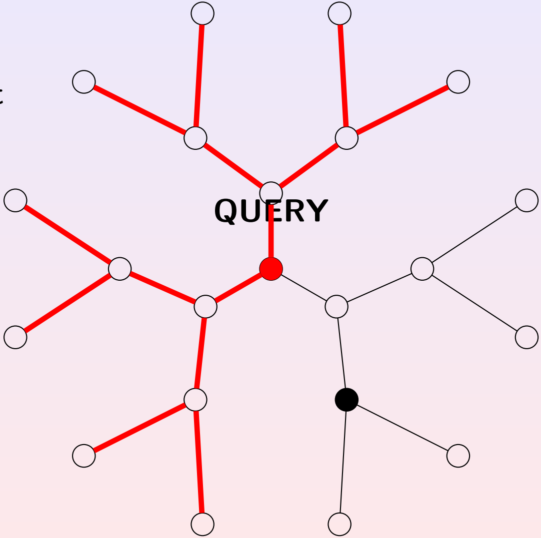
Exemple with q=2 :

2 queries left



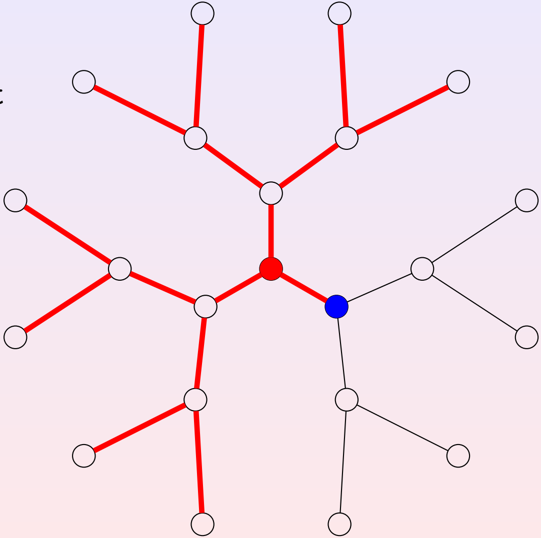
Exemple with q=2 :

1 query left



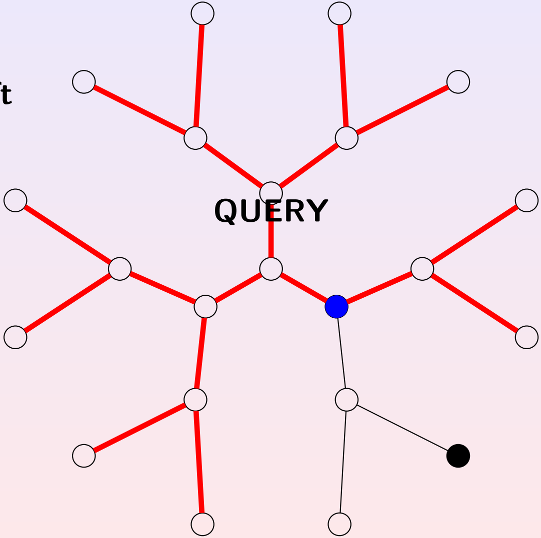
Exemple with q=2 :

1 query left



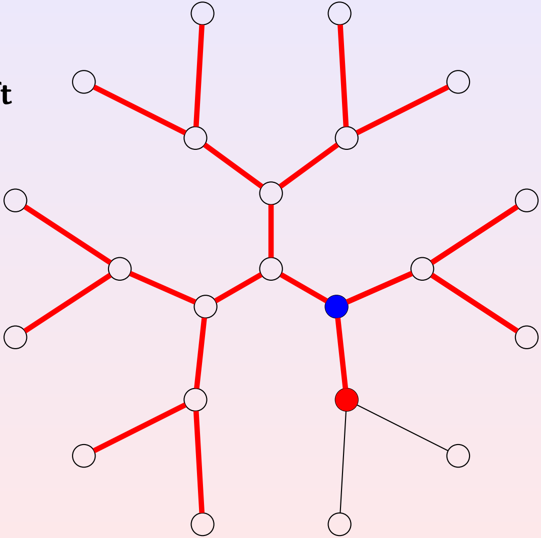
Exemple with q=2 :

no query left



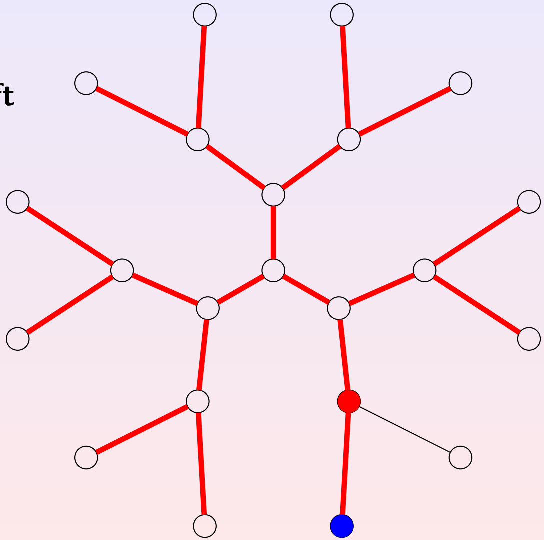
Exemple with q=2 :

no query left



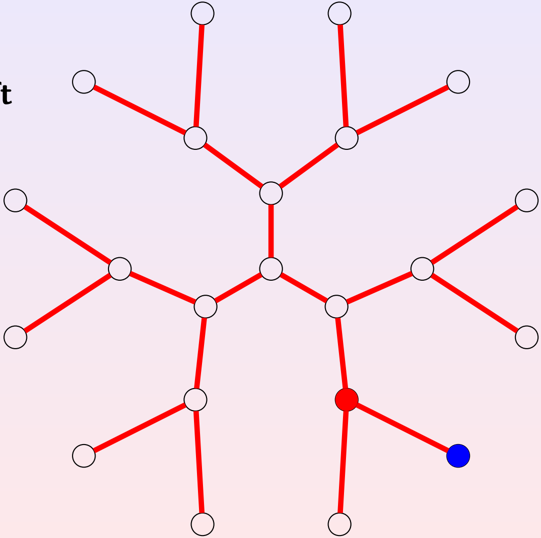
Exemple with q=2 :

no query left



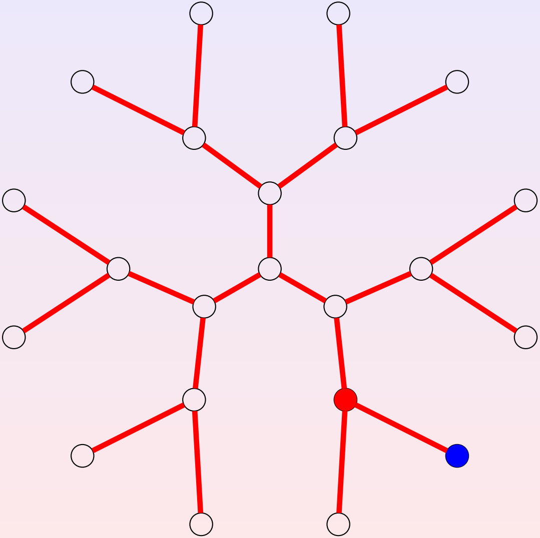
Exemple with q=2 :

no query left



Exemple with q=2 :

$s_2(T)=2$



Some terminology

q-branched tree

- rooted tree ;
- branching node (at least two children) ;
- every path from the root to a leaf contains at most q branching nodes.

Interpretation in terms of Graph Decompositions

(T, X) : q-branched tree decomposition

(T, X) a tree-decomposition with T a q-branched tree.

q-branched treewidth, $tw_q(G)$, minimum width among any q-branched tree-decomposition of G.

- path decomposition $=$ 0-branched tree decomposition
  $\mathbf{pw}(G) = \mathbf{tw}_0(G)$;
- tree decomposition $=$ ∞-branched tree decomposition
  $\mathbf{tw}(G) = \mathbf{tw}_\infty(G)$;

Interpretation in terms of Graph Decompositions

The branched decompositions correspond to monotone search strategies for non deterministic graph searching.
$\mathbf{ms}_q(G)$: q-limited monotone search number

Theorem[Fomin, Fraigniaud, Nisse, 2005] :

1. For any $q \geq 0$, for any graph G, $\mathbf{ms}_q(G) = \mathbf{tw}_q(G) + 1$;
2. Computing $tw_q(G)$ is NP-complete for any q ;
3. Exact exponential algorithm that, for any graph G and any $q \geq 0$, computes $\mathbf{tw}_q(G)$ and an optimal decomposition.

Does recontamination help for any $q \geq 0$?

Interpretation in terms of Graph Decompositions

The branched decompositions correspond to monotone search strategies for non deterministic graph searching.
$\mathbf{ms}_q(G)$: q-limited monotone search number

Theorem[Fomin, Fraigniaud, Nisse, 2005] :

1. For any $q \geq 0$, for any graph G, $\mathbf{ms}_q(G) = \mathbf{tw}_q(G) + 1$;
2. Computing $tw_q(G)$ is NP-complete for any q ;
3. Exact exponential algorithm that, for any graph G and any $q \geq 0$, computes $\mathbf{tw}_q(G)$ and an optimal decomposition.

Does recontamination help for any $q \geq 0$?

Related work

Recontamination does not help to catch an invisible fugitive.

Case of an invisible fugitive : $\mathbf{s}_0(G) = \mathbf{ms}_0(G)$

- **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 proof by Bienstock and Seymour :
Local optimisation that transforms a search strategy into a monotone one without increasing the number of searchers.

Related work

Recontamination does not help to catch an invisible fugitive.

Case of an invisible fugitive : $\mathbf{s}_0(G) = \mathbf{ms}_0(G)$

- **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 proof by Bienstock and Seymour :
Local optimisation that transforms a search strategy into a monotone one without increasing the number of searchers.

Related work

Recontamination does not help to catch a visible fugitive.

Case of a visible fugitive : $\mathbf{s}_\infty(G) = \mathbf{ms}_\infty(G)$

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

scheme of the proof :

there is no search strategy using k searchers.

$\Rightarrow$ there is no monotone search strategy using k searchers

$\Rightarrow$ there exists an escape strategy for the fugitive

$\Rightarrow$ there exists a general escape strategy for the fugitive

Related work

Recontamination does not help to catch a visible fugitive.

Case of a visible fugitive : $\mathbf{s}_\infty(G) = \mathbf{ms}_\infty(G)$

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

scheme of the proof :

there is no search strategy using k searchers.

$\Rightarrow$ there is no monotone search strategy using k searchers

$\Rightarrow$ there exists an escape strategy for the fugitive

$\Rightarrow$ there exists a general escape strategy for the fugitive

Our result

Recontamination never helps in non deterministic graph searching.

For any $q \geq 0$ and any graph G, $\mathbf{s}_q(G) = \mathbf{ms}_q(G)$

Remarks :

- Constructive proof that unifies the existing proofs ;
- Deciding $\mathbf{s}_q(G) \leq k$ is in NP ;
- The algorithm of Fomin *et al.* actually computes $\mathbf{s}_q(G)$.

Sketch of the proof

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

Let G be a connected graph, $q \geq 0$ and $k \geq 1$. The following are equivalent

1. there exists a non deterministic search strategy using $\leq k$ searchers and at most q queries ;
2. there is a q-branched search-tree with width $\leq k$;
3. there is a monotone q-branched search-tree with width $\leq k$;
4. there exists a non deterministic monotone search strategy using $\leq k$ searchers and at most q queries ;

Some terminology

Border of subsets of edges $E_1, \cdots, E_p \subseteq E(G)$:

$\delta(E_1, E_2)$= set of vertices incident to an edge of E_1 and an edge of E_2.

$\delta(E_1) = \delta(E_1, E(G) \setminus E_1)$.

$\delta(E_1, \cdots, E_p) = \bigcup_{i \neq j} \delta(E_i, E_j)$.

Definition of search-tree

(T, α, β, r) a search-tree of a graph G ;

- T : a tree rooted in $r \in V(T)$;
- α : incidence of $T \longrightarrow$ subset of $E(G)$;
  $v \in V(T)$, e incident to $v \longrightarrow \alpha(v, e) \subseteq E(G)$.
- β : $V(T) \longrightarrow$ subset of $E(G)$;
  $v \in V(T) \longrightarrow \beta(v) \subseteq E(G)$.

Definition of search-tree

(T, α, β, r) a search-tree of a graph G ;

- T : a tree rooted in $r \in V(T)$;
- α : incidence of $T \rightarrow$ subset of $E(G)$;
  $v \in V(T)$, e incident to $v \rightarrow \alpha(v, e) \subseteq E(G)$.
- β : $V(T) \rightarrow$ subset of $E(G)$;
  $v \in V(T) \rightarrow \beta(v) \subseteq E(G)$.

Definition of search-tree

(T, α, β, r) a search-tree of a graph G;

- T : a tree rooted in $r \in V(T)$;
- α : incidence of T $\rightarrow$ subset of $E(G)$;
  $v \in V(T)$, e incident to v $\rightarrow$ $\alpha(v, e) \subseteq E(G)$.
- $\beta : V(T) \rightarrow$ subset of $E(G)$;
  $v \in V(T) \rightarrow \beta(v) \subseteq E(G)$.

Definition of search-tree

(T, α, β, r) a search-tree of a graph G;

- T : a tree rooted in $r \in V(T)$;
- α : incidence of $T \longrightarrow$ subset of $E(G)$;
  $v \in V(T)$, e incident to $v \longrightarrow \alpha(v, e) \subseteq E(G)$.
- β : $V(T) \longrightarrow$ subset of $E(G)$;
  $v \in V(T) \longrightarrow \beta(v) \subseteq E(G)$.

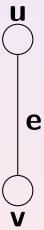
Definition of search-tree

(T, α, β, r) must satisfy two properties :

(P1) for any edge $e = \{u, v\}$ of T, $\alpha(u, e) \cap \alpha(v, e) = \emptyset$;

set of contaminated edges before one step $\alpha(u, e)$

cleared edges that remains clear after one step $\alpha(v, e)$



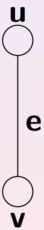
If $\alpha(u, e) = E(G) \setminus \alpha(v, e)$, e is said monotone.

(T, α, β, r) must satisfy two properties :

(P1) for any edge $e = \{u, v\}$ of T, $\alpha(u, e) \cap \alpha(v, e) = \emptyset$;

set of contaminated edges before one step $\alpha(u, e)$

cleared edges that remains clear after one step $\alpha(v, e)$

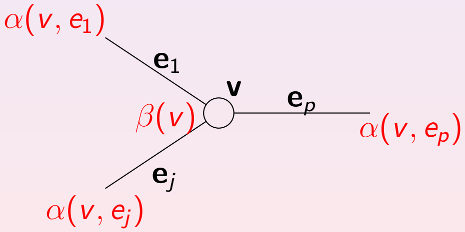


If $\alpha(u, e) = E(G) \setminus \alpha(v, e)$, e is said monotone.

Definition of search-tree

(T, α, β, r) must satisfy two properties :

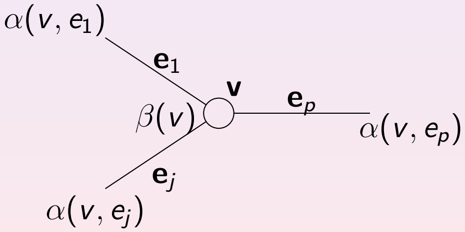
(P2) for any node v of T incident to $e_1, \ldots, e_p$,
$\{\beta(v), \alpha(v, e_1,), \ldots, \alpha(v, e_p)\}$ is a partition of E



Definition of search-tree

(T, α, β, r) is q-branched if T is q-branched.

width(T)$= \max_{v \in V(T)} |\delta(\alpha(v, e_1,), \ldots, \alpha(v, e_p)) \bigcup V[\beta(v)]|$;



From the strategy to the search-tree

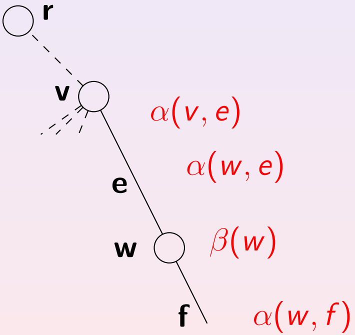
We construct the search-tree (T, α, β, r) recursively ;

Each edge **e** corresponds to a step of the strategy ;

Each branching node corresponds to a query step.

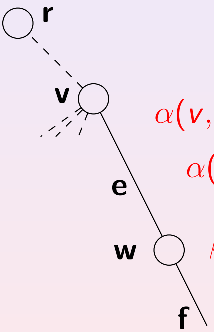
From the strategy to the search-tree

If the considered step i is a *Placing searcher* or *Removing searcher* step :



From the strategy to the search-tree

If the considered step i is a *Placing searcher* or *Removing searcher* step :



$\alpha(v, e)$, **contaminated edges before step i**

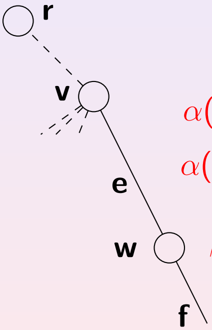
$\alpha(w, e)$

$\beta(w)$

$\alpha(w, f)$

From the strategy to the search-tree

If the considered step *i* is a *Placing searcher* or *Removing searcher* step :



$\alpha(v,e)$, contaminated edges before step i

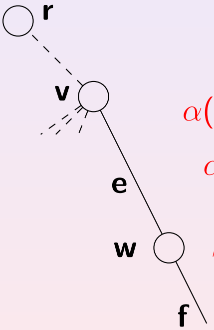
$\alpha(w,e)$, **edges cleared before step i and that remain clear**

$\beta(w)$

$\alpha(w,f)$

From the strategy to the search-tree

If the considered step i is a *Placing searcher* or *Removing searcher* step :



$\alpha(v, e)$, contaminated edges before step i

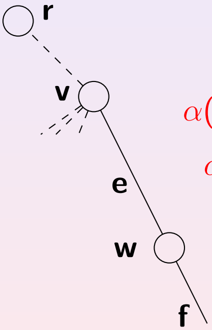
$\alpha(w, e)$, edges cleared before step i
and that remain clear

$\beta(w)$, **edges cleared at step i**
thanks to placement of new searchers

$\alpha(w, f)$

From the strategy to the search-tree

If the considered step i is a *Placing searcher* or *Removing searcher* step :



$\alpha(v, e)$, contaminated edges before step i

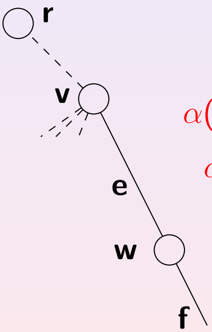
$\alpha(w, e)$, edges cleared before step i
and that remain clear

$\beta(w)$, edges cleared at step i
thanks to placement of new searchers

$\alpha(w, f)$, **contaminated edges after step i**

From the strategy to the search-tree

If the considered step i is a *Placing searcher* or *Removing searcher* step :



$\alpha(v, e)$, contaminated edges before step i

$\alpha(w, e)$, edges cleared before step i
and that remain clear

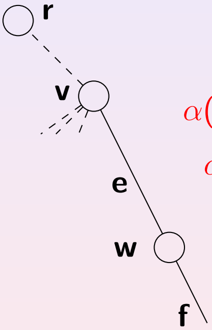
$\beta(w)$, edges cleared at step i
thanks to placement of new searchers

$\alpha(w, f)$, contaminated edges after step i

P1 and **P2** are satisfied.

From the strategy to the search-tree

If the considered step i is a *Placing searcher* or *Removing searcher* step :



$\alpha(v, e)$, contaminated edges before step i

$\alpha(w, e)$, edges cleared before step i
and that remain clear

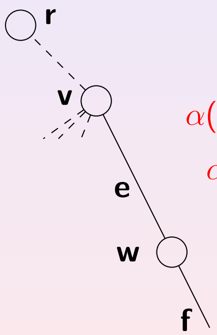
$\beta(w)$, edges cleared at step i
thanks to placement of new searchers

$\alpha(w, f)$, contaminated edges after step i

If initial search strategy allows recontamination at this step, the corresponding edge e is not monotone.

From the strategy to the search-tree

If the considered step i is a *Placing searcher* or *Removing searcher* step :



$\alpha(v, e)$, contaminated edges before step i

$\alpha(w, e)$, edges cleared before step i and that remain clear

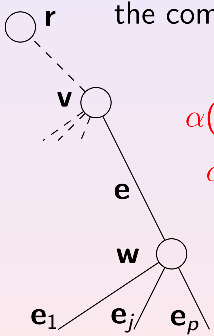
$\beta(w)$, edges cleared at step i thanks to placement of new searchers

$\alpha(w, f)$, contaminated edges after step i

$$\delta(\alpha(w, e), \alpha(w, f)) \bigcup V[\beta(w)] \leq \#(\text{searchers}).$$

From the strategy to the search-tree

If we consider a *Performing a query* step, with $C_1, \cdots, C_p$ the components where the fugitive can stand



$\alpha(v, e)$, contaminated edges before step i

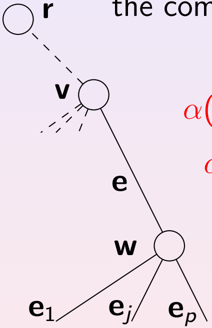
$\alpha(w, e)$, edges cleared before step i and that remain clear

$\beta(w) = \emptyset$, edges cleared at step i thanks to placement of new searchers

$\alpha(w, e_j) = E(C_j)$, **contaminated edges after step** i **according to the answer of the oracle**

From the strategy to the search-tree

If we consider a *Performing a query* step, with $C_1, \cdots, C_p$ the components where the fugitive can stand



$\alpha(v, e)$, contaminated edges before step i

$\alpha(w, e)$, edges cleared before step i and that remain clear

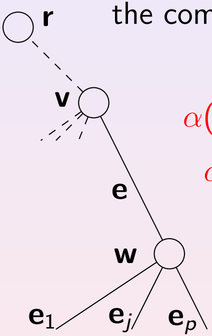
$\beta(w) = \emptyset$, edges cleared at step i thanks to placement of new searchers

$\alpha(w, e_j) = E(C_j)$, **contaminated edges after step *i* according to the answer of the oracle**

P1 and **P2** are satisfied.

From the strategy to the search-tree

If we consider a *Performing a query* step, with $C_1, \cdots, C_p$ the components where the fugitive can stand



$\alpha(v, e)$, contaminated edges before step i

$\alpha(w, e)$, edges cleared before step i and that remain clear

$\beta(w) = \emptyset$, edges cleared at step i thanks to placement of new searchers

$\alpha(w, e_j) = E(C_j)$, **contaminated edges after step *i* according to the answer of the oracle**

$$\delta(\alpha(w, e), \cdots, \alpha(w, e_p)) \bigcup V[\beta(w)] \leq \#(\text{searchers}).$$

From a search-tree to a monotone search-tree

A search-tree is monotone if all its edges are monotone.

By local optimizations, we build a monotone search-tree from a search tree (T, α, β, r).

From a search-tree to a monotone search-tree

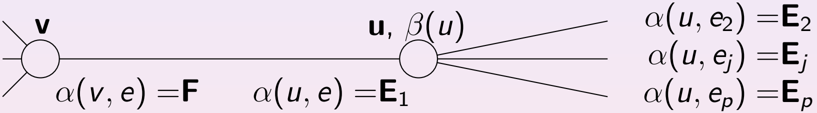
A search-tree is monotone if all its edges are monotone.

By local optimizations, we build a monotone search-tree from a search tree (T, α, β, r).

From a search-tree to a monotone search-tree

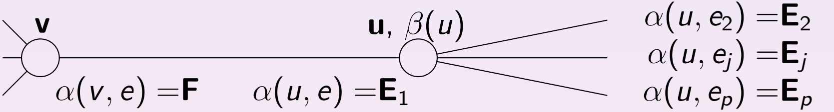
Local Optimisation

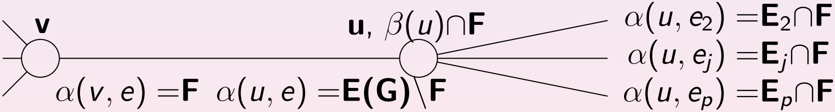
Let $e = \{v, u\}$ a non monotone edge (i.e., $E_1 \cup F \neq E$)



Local Optimisation

Let $e = \{v, u\}$ a non monotone edge (i.e., $E_1 \cup F \neq E$)





We define a weight function that strictly decreases by this optimisation.

From a search-tree to a monotone search-tree

2 weight functions

- w(T)= $\sum_{v \in V(T)} |\delta\big(\alpha(v, e_1)..\alpha(v, e_\ell)\big) \bigcup V[\beta(v)]|$
- bd(T)= $\sum n^{-\mathbf{dist}(r,e)}$ the sum being taken over the non monotone edges

From monotone search-tree to monotone strategy

(T, α, β, r) a q-branched monotone search-tree with width $\leq k$

Then, $(T, (X_v)_{v \in V(T)})$ with

$$X_v = \delta(\alpha(v, e_1)..\alpha(v, e_\ell)) \bigcup V[\beta(v)]$$

is a q-branched tree-decomposition with width $\leq k - 1$

Using the Theorem of Fomin *et al.*, we get the result.

Open problems

About monotony

Does recontamination help for catching a visible fugitive that runs in a directed graph ?

About non deterministic graph searching

Linear algorithm in case of trees ?
Linear algorithm in case of the class of graph with $\mathbf{tw}_q$ bounded ?

Open problems

About monotony

Does recontamination help for catching a visible fugitive that runs in a directed graph ?

About non deterministic graph searching

Linear algorithm in case of trees ?
Linear algorithm in case of the class of graph with $\mathbf{tw}_q$ bounded ?