Monotonicity of Non-deterministic Graph Searching *

Frédéric Mazoit[1] and Nicolas Nisse[2]

[1] LABRI, University of Bordeaux, 33405 Talence, France
[2] LRI, University of Paris Sud, 91405 Orsay, France

Abstract. In graph searching, a team of searchers is aiming at capturing a fugitive moving in a graph. In the initial variant, called *invisible graph searching*, the searchers do not know the position of the fugitive until they catch it. In another variant, the searchers know the position of the fugitive, i.e. the fugitive is visible. This latter variant is called *visible graph searching.* A search strategy that catches any fugitive in such a way that, the part of the graph reachable by the fugitive never grows is called *monotone. A priori*, monotone strategies may require more searchers than general strategies to catch any fugitive. This is however not the case for visible and invisible graph searching. Two important consequences of the monotonicity of visible and invisible graph searching are: (1) the decision problem corresponding to the computation of the smallest number of searchers required to clear a graph is in NP, and (2) computing optimal search strategies is simplified by taking into account that there exist some that never backtrack.
Fomin *et al.* (2005) introduced an important graph searching variant, called *non-deterministic graph searching*, that unifies visible and invisible graph searching. In this variant, the fugitive is invisible, and the searchers can query an oracle that knows the current position of the fugitive. The question of the monotonicity of non-deterministic graph searching is however left open.
In this paper, we prove that non-deterministic graph searching is monotone. In particular, this result is a unified proof of monotonicity for visible and invisible graph searching. As a consequence, the decision problem corresponding to non-determinisitic graph searching belongs to NP. Moreover, the exact algorithms designed by Fomin *et al.* do compute optimal non-deterministic search strategies.

Keywords: Graph searching, Treewidth, Monotonicity.

1 Introduction

Introduced in [14], graph searching is a game in which a team of searchers aims at catching a fugitive moving in a graph. At each step of the game, a searcher

* The second author received additional supports from the project "Alpage" of the ACI Masses de Données, from the project "Fragile" of the ACI Sécurité Informatique, and from the project "Grand Large" of INRIA.

can either be placed at or removed from a vertex of the graph [10]. The fugitive is invisible, arbitrary fast and aware of the positions of the searchers. It can move along paths of the graph as long as it does not cross any vertex occupied by a searcher. The fugitive is caught when a searcher is placed at the vertex it occupies, and it cannot flee because all the neighbors are occupied by searchers. A *search strategy* for a graph G is a sequence of basic operations, i.e., place or remove a searcher, that results in catching any invisible fugitive in G. The *node search number* of a graph G, denoted by $\mathbf{s}(G)$, is the smallest integer k such that it exists a search strategy for G using at most k searchers. Given a graph G, the *graph searching problem* consists in computing an *optimal* search strategy for G, i.e., a strategy that clears G using at most $\mathbf{s}(G)$ searchers.

During a search strategy, the vertices that are accessible by the fugitive are said *contaminated*. A non-contaminated vertex is said *clear*. A strategy is *monotone* if it does not allow *recontamination*, i.e., after having been cleared, a vertex remains clear until the end of the strategy. LaPaugh [11] proved that "recontamination does not help" to catch an invisible fugitive. That is, for any graph G, there exists a monotone search strategy of G using at most $\mathbf{s}(G)$ searchers. We say that invisible graph searching is monotone. LaPaugh's proof was later simplified by Bienstock and Seymour [4] using the concept of *crusades*. Both these proofs are constructive. Indeed, they transform any strategy into a monotone one without increasing the number of searchers.

In [17], Seymour and Thomas introduce the *visible graph searching*. In this variant [5, 17], the searchers are aware of the position of the fugitive. Hence, they can adapt their strategy according to its position. The *visible search number* of a graph G, denoted by $\mathbf{vs}(G)$, is the smallest integer k such that k searchers are sufficient to catch any visible fugitive in G. Seymour and Thomas [17] proved that visible graph searching is monotone. However, Seymour and Thomas' proof is not constructive. They show that, if no monotone strategies using k searchers exist for a graph G, then there exists an escape strategy for the fugitive which actually is a general escape strategy, and thus, no non-monotone strategies using at most k searchers allow to catch any visible fugitive in G.

Monotonicity plays a crucial role in graph searching. First, a monotone strategy conludes in a polynomial number of steps and thus, gives a certificate of polynomial size for the decision problem corresponding to the graph searching problem. Since the decision problems corresponding to the visible and invisible graph searching problems are known to be NP-hard [12, 17], they are NP-complete. Second, it appears algorithmically difficult to design strategies that are not monotone. Last but not least, monotone strategies for catching an invisible (resp., visible) fugitive in a graph G correspond exactly to path-decompositions (resp., tree-decompositions) [15] of G.

Indeed, the importance of visible graph searching and invisible graph searching comes from their close relationship with crucial notions of graph theory: *treewidth* and *pathwidth* [15]. Roughly speaking, the treewidth $\mathbf{tw}(G)$ (resp., the pathwidth $\mathbf{pw}(G)$) of a graph G measures how close this graph is from a tree (resp., a path). The correspondence between search numbers and width parameters provides different interpretations of these parameters, and thus, different ways of handling them. More precisely, $\mathbf{s}(G) = \mathbf{pw}(G) + 1$ (see [6, 10]), and $\mathbf{vs}(G) = \mathbf{tw}(G) + 1$ (see [5, 17]).

In [7], Fomin *et al.* provide a unique approach to the pathwidth and the treewidth of a graph. For any graph G and any $q \geq 0$, they define the notions of *q-branched tree decomposition* and *q-branched treewidth*, denoted by $\mathbf{tw}_q(G)$. Roughly speaking, a q-branched tree decomposition of a graph is a parametrized tree decomposition such that the number of branching nodes of the tree is limited. In particular, path-decompositions are exactly 0-branched tree decompositions, and tree-decompositions are exactly ∞-branched tree decompositions.

Fomin *et al.* also provide an interpretation of q-branched tree decompositions in terms of graph searching. More precisely, they provide a unique approach to both visible and invisible search problems, called *non-deterministic* graph searching. In this variant, the fugitive is invisible. However, the searchers can query an oracle that knows the position of the fugitive. Given the set $S \subseteq V(G)$ of clear vertices, a query returns a connected component C of $G \setminus S$, and all vertices in $G \setminus C$ are cleared. The choice of C is nondeterministic. Intuitively, the oracle gives the position of the fugitive to the searchers. More formally, the searchers can perform one of the following three basic operations called *search steps*: (1) place a searcher at a vertex of the graph, (2) remove a searcher from a vertex of the graph, and (3) perform a query to the oracle.

The number of query steps that the searchers can perform is however limited. For $q \geq 0$, the monotone q-limited search number $\mathbf{ms}_q(G)$ of a graph G is the smallest number of searchers required to catch any fugitive in G in a monotone way performing at most q queries. The main result of Fomin *et al.* [7] is the following generalization of the above two equations:

$$\textit{For any graph } G \textit{ and any } q \geq 0,\ \mathbf{tw}_q(G) + 1 = \mathbf{ms}_q(G). \tag{1}$$

Moreover, Fomin *et al.* [7] prove the NP-completness of the problem of computing $\mathbf{ms}_q(G)$, for any $q \geq 0$. Using the correspondence between monotone q-limited graph searching and q-branched treedecomposition, they also design an exact exponential algorithm that computes $\mathbf{tw}_q(G)$ and the corresponding decomposition, for any graph G and any $q \geq 0$.

However, Fomin *et al.* only consider monotone non-deterministic search strategies. They left open the problem whether recontamination helps for q-limited graph searching, for any $q \geq 0$. This paper answers this question.

Our Results

Let G be a graph and $q \geq 0$. Let $\mathbf{s}_q(G)$ denotes the smallest number of searchers required to catch any fugitive in G performing at most q queries. We prove that, for any graph G and any $q \geq 0$, recontamination does not help to catch a fugitive in G performing at most q queries. In other words, we prove that for any graph G and any $q \geq 0$, there exists a monotone search strategy of G using at most $\mathbf{s}_q(G)$ searchers, i.e. $\mathbf{s}_q(G) = \mathbf{ms}_q(G)$. In particular, this implies that the decision problem related to non-deterministic graph searching is in NP. This also implies that the exponential exact algorithm designed in [7] actually computes $\mathbf{s}_q(G)$ for any graph G and any $q \geq 0$. More interestingly, our result unifies the proof of the monotonicity of invisible graph searching [4] and the proof of the monotonicity of visible graph searching [17]. The original proof of the monotonicity of visible graph searching is not constructive, while our proof is constructive and turns any general strategy into a monotone one.

Related Works

The monotonicity property of several graph searching variants has been studied before. In [2], Barrière *et al.* have defined the connected graph searching. A search strategy is *connected* if, at any step of the strategy, the subgraph induced by the clear vertices is connected. Barrière *et al.* [2] proved that connected graph searching is monotone as long as the input graph is restricted to be a tree. However, this does not remain true in case of arbitrary graphs. Yang *et al.* [18] proved that there exist graphs for which "recontamination does help" to catch an invisible fugitive in a connected way. In [8], Fraigniaud and Nisse proved that recontamination does help as well to catch a visible fugitive in a connected way.

In [9], Johnson *et al.* introduced *directed* graph searching. In this variant of the game, a visible fugitive is moving in a digraph. However, it is only permitted to move to vertices where there exists a directed searcher-free path from its intended destination back to its current position. The authors exhibit a graph for which recontamination does help. Obdrzálek [13] and Berwanger *et al.* [3] independamently defined a new visible graph searching game in a digraph by relaxing the latter constraint. The question of the monotonicity of this latter variant is however left open. In [1], Barát studies the monotonicity property of a search strategy for catching invisible fugitive moving in a digraph. He proves that mixed-graph searching is monotone in directed graph.

2 Formal Definitions

In this paper, $G = (V, E)$ will denote a connected graph with vertex-set V and edge-set E. For $A \subseteq E$, we denote by $V[A]$ the set of vertices incident to at least one edge in A. The border of two disjoint edge sets A and B is the set $\delta(A, B) = V[A] \cap V[B]$ of the vertices incident both to an edge in A and to an edge in B. We extend this definition to any family of pair-wise disjoint edge sets $\{X_1, \dots, X_p\}$ by setting $\delta(X_1, \cdots, X_n) = \bigcup_{1 \leq i < j \leq n} \delta(X_i, X_j)$. The *border* $\delta(X)$ of $X \subseteq E$ denotes the set $\delta(X, E \setminus X)$.

2.1 Non-deterministic Graph Searching

Now, we formally define the notion of non-deterministic search strategy. Intuitively, given a graph G, a non-deterministic search strategy (or simply a non-deterministic strategy) for G is a sequence of pairs, such that each pair consists of a subset of V, the positions of searchers, and a subset of E, the clear part of G. More precisely, a *non-deterministic strategy* is a sequence of ordered pairs $(Z_i, A_i)_{i \in [0,l]}$ such that

- for any $0 \leq i \leq l$, $Z_i \subseteq V$ and $A_i \subseteq E$;
- $Z_0 = \emptyset$ and $A_0 = \emptyset$;
- for any $0 \leq i < l$ one of the following holds
  - (placing searchers) there is $X_{i+1} \subseteq V$, such that $Z_{i+1} = Z_i \cup X_{i+1}$ and $A_{i+1} = A_i \cup B_{i+1}$ with B_{i+1} the set of edges with both ends in Z_{i+1}, or
  - (removing searchers) there is $X_{i+1} \subseteq V$, such that $Z_{i+1} = Z_i \setminus X_{i+1}$ and A_{i+1} is obtained from A_i by recursively removing the edges $\{x, y\} \in A_i$ with $y \notin Z_{i+1}$ and such that there is $z \in V$ with $\{y, z\} \notin A_{i+1}$, or
  - (performing a query) $Z_{i+1} = Z_i$ and A_{i+1} is the set of edges not incident to a vertex of C, one of the connected component of $G \setminus Z_i$ not incident to a vertex of A_i. The choice of C is non-deterministic.

For any $0 \leq i \leq l$, (Z_i, A_i) is the *configuration* reached by the strategy at the i^{th} step. A strategy $(Z_i, A_i)_{i \in [0,l]}$ uses at most $k \geq 1$ searchers if, for any $0 \leq i \leq l$, $|Z_i| \leq k$. A *non-deterministic search program* is a non-deterministic program that takes as input a graph G and an integer $k \geq 1$, and returns a non-deterministic strategy for G using at most k searchers. A non-deterministic search program *wins* if for every possible fugitive moves, at least one of the strategies that the program computes catches the fugitive. That is, for any non-deterministic choice of the component C during the "performing a query" steps, the computed strategy insures that $A_l = E$. A non-deterministic search program is *monotone* if the strategies that it computes are monotone. The number of searchers required by a non-deterministic strategy is the maximum number of searchers required by the strategies that it computes.

A *q-limited non-deterministic search program* (or simply, a *q-program*) is a non-deterministic search program that computes strategies using at most q query steps. The *q-limited search number* (or simply the *q-search number*) of a graph G, denoted by $\mathbf{s}_q(G)$, is the smallest number of searchers required by a q-program to win against any fugitive in G. Similarily, we define the *monotone q-limited search number* of a graph G, denoted by $\mathbf{ms}_q(G)$, as the smallest number of searchers required by a monotone q-program to win against any fugitive in G.

If $q = 0$, no non-deterministic steps are allowed, and the previous definition is similar to the usual definition of an invisible search strategy [10]. Note that, in this case, the deterministic strategy $(Z_i, A_i)_{i \in [0,l]}$ wins, if and only if, there is $0 < i \leq l$ such that, for any $j \geq i$, $A_j = E$.

2.2 Branched Treewidth

Fomin *et al.* [7] defined a parametrized version of the tree-decomposition of a graph. Their main result is the interpretation of this decomposition in terms of graph searching.

A *tree decomposition* [15] of graph G is a pair $(T, \mathcal{X})$ where T is a tree of node set I, and $\mathcal{X} = \{X_i, i \in I\}$ is a collection of subsets of $V(G)$ satisfying the following three conditions:

i. $V(G) = \cup_{i \in I} X_i$;
ii. for any edge e of G, there is a set $X_i \in \mathcal{X}$ containing both end-points of e;
iii. for any $i_1, i_2, i_3 \in I$ with i_2 is on the path from i_1 to i_3 in T, $X_{i_1} \cap X_{i_3} \subseteq X_{i_2}$.

The *width* $\mathbf{w}(T, \mathcal{X})$ of a tree decomposition is $\max_{i \in I}\{|X_i| - 1\}$ and the *treewidth* of a graph is the minimum width over all its tree-decompositions.

A *rooted* tree decomposition of a graph G, denoted by $(T, \mathcal{X}, r)$, is a tree decomposition $(T, \mathcal{X})$ of G such that T is a rooted tree and r is its root. A *branching node* of a rooted tree decomposition is a node with at least two children. For any $q \geq 0$, a *q-branched tree decomposition* [7] (or simply, a *q-tree decomposition*) of a graph G is a rooted tree decomposition $(T, \mathcal{X}, r)$ of G such that every path in T from the root r to a leaf contains at most q branching nodes. Thus a path decomposition rooted at one of its extremities is a 0-branched tree decomposition, and a usual tree decomposition is a ∞-branched tree decomposition. For any graph G, the *q-branched treewidth* (or simply, the *q-treewidth*) of G, denoted by $\mathbf{tw}_q(G)$, is the minimum width of any q-tree decomposition of G.

Theorem 1. *[7] Let G be a graph, $q \geq 0$ and $k \geq 1$. There is a winning monotone q-program using at most k searchers in G if and only if* $\mathbf{tw}_q(G) < k$.

2.3 Search-tree

To prove the monotonicity of non-deterministic graph searching, we define an auxiliary structure called *search-tree* which is inspired by the tree-labelling defined by Robertson and Seymour [16]. A *search-tree* of a graph G is a triple (T, α, β) with T a tree, α a mapping from the incidence (between vertices and edges) of T into the subsets of E and β a mapping from the vertices of T into the subsets of E such that:

- for any edge $e = \{u, v\}$ of T, $\alpha(u, e) \cap \alpha(v, e) = \emptyset$;
- for no leaf v of T incident to an edge e of T is such that $\alpha(v, e) = E$;
- for any node v of T incident to $e_1, \dots, e_p$, $\{\beta(v)\} \cup \mu(v)$ is a (possibly degenerated) partition of E with $\mu(v) = \{\alpha(v, e_1), \dots, \alpha(v, e_p)\}$.

We extend the function β to any sub-tree T' of T by setting $\beta(T') = \cup_{v \in T'} \beta(v)$. The *width* of a search-tree is defined as $\mathbf{w}(T, \alpha, \beta) = \max_{v \in V(T)}\{|\chi_v|\}$ where $\chi_v = V[\beta(v)] \cup \delta(\mu(v))$ and $|\chi_v|$ denotes the *weight* of the node $v \in V(T)$. As for tree decompositions, we consider *rooted* search-trees, denoted by (T, α, β, r), that are search-trees over rooted trees. A *branching node* of a rooted search-tree is a node with at least two children. For any $q \geq 0$, a *q-branched* search-tree is a rooted search-tree (T, α, β, r) of G such that every path in T from the root r to a leaf contains at most q branching nodes. An edge $e = \{u, v\}$ of a search-tree is *monotone* if $\alpha(u, e) = E \setminus \alpha(v, e)$, and a search-tree is *monotone* if all its edges are monotone. Edges that are not monotone are said *dirty*.

3 Monotonicity of non-deterministic graph searching

The remaining part of the paper is devoted to prove the monotonicity of non-deterministic graph searching. For this purpose, we prove that, from any winning q-program using at most k searchers in a graph G, we can build a q-branched search-tree of width at most k for G (Lemma 1). Then, by performing local optimisations, we transform any q-search-tree into a monotone one (Lemma 3) without increasing its width. To conclude, any monotone q-branched search-tree, of width k, of a graph G can be transformed into a q-branched tree-decomposition, of width at most $k-1$, of G (Lemma 5). Then, the proof of the monotonicity property of non-deterministic graph searching follows from Theorem 1. More formally, we prove the following theorem:

Theorem 2. *Let G be a connected graph, $q \geq 0$ and $k \geq 2$. The following are equivalent:*

i. There is a winning q-program for G using at most k searchers;
ii. There is a q-search-tree of width at most k for G;
iii. There is a monotone q-search-tree of width at most k for G;
iv. There is a q-tree decomposition of width at most $k-1$ for G;
v. There is a winning monotone q-program for G using at most k searchers.

Proof. We prove that $(i) \Rightarrow (ii)$ (Lemma 1), $(ii) \Rightarrow (iii)$ (Lemma 3), $(iii) \Rightarrow (iv)$ (Lemma 5). Proposition $(iv) \Rightarrow (v)$ follows from Theorem 1 and $(v) \Rightarrow (i)$ is obvious.

3.1 From strategies to search-trees

Lemma 1. *Let G be a connected graph, $q \geq 0$ and $k \geq 2$, $(i) \Rightarrow (ii)$.*

i. There is a winning q-program using at most k searchers for G;
ii. There is a q-search-tree of width at most k for G.

Proof. In this proof, we consider extended search programs whose the *starting configuration* is not necessarily the $(\emptyset, \emptyset)$ configuration. That is, we consider search programs whose strategies start from a configuration (Z_0, A_0) that satisfies $\delta(A_0) \subseteq Z_0$. The *length* of a search program is the maximum number of steps of the strategies it computes. Let us define the *partial width* of a rooted search tree as the maximum weight of its nodes, the maximum being taken over all the nodes of the search-tree but its root. We prove the following claim by induction on the length of the search program.

Claim. For every winning q-program using at most k searchers with (Z_0, A_0) as starting configuration, there is a rooted q-search-tree (T, α, β, r) of partial width at most k, and such that, r is incident to a unique edge $e \in E(T)$, and $\alpha(r, e) = E \setminus A_0$.

Let $q \geq 0$ and let $\mathcal{S}$ be a q-program on G with k searchers and with (Z_0, A_0) as starting configuration.

- Suppose that $\mathcal{S}$ has length 1.
  The only search step has to be a "placing searchers" step. Thus, $\mathcal{S}$ computes only the following 0-strategy: $(Z_0, A_0), (Z_1, A_1)$ in which $Z_1 = Z_0 \cup X_1$ and $A_1 = A_0 \cup B_1 = E$.
  Define the tree T with only one edge $\{r, v\}$, $\beta(v) = \alpha(r, \{r, v\}) = E \setminus A_0$ and $\beta(r) = \alpha(v, \{r, v\}) = A_0$. Since $V\big[\beta(v)\big] \cup \delta(\mu_v) = V[E \setminus A_0]$ which is a subset of Z_1, (T, α, β, r) is a rooted 0-search tree of partial width at most k.
- Suppose that $\mathcal{S}$ has length $l > 1$. Let us assume that for any winning q-program $\mathcal{S}'$ using at most k searchers with (Z, A) as starting configuration (with $\delta(A) \subseteq Z$) and such that $\mathcal{S}'$ has length $l' < l$, there is a rooted q-search-tree (T, α, β, r) of partial width at most k, and such that, r is incident to a unique edge $e \in E(T)$, and $\alpha(r, e) = E \setminus A$. Consider $\mathcal{S}'$ obtained by removing the first configuration of the sequences of $\mathcal{S}$. Note that, $\mathcal{S}'$ is strictly shorter than $\mathcal{S}$. We consider three cases according to the type of the first step of $\mathcal{S}$.
  a. if the first step of $\mathcal{S}$ is a "removing searchers" step, $\mathcal{S}'$ is a q-program with (Z_1, A_1) as a starting configuration, $Z_1 \subseteq Z_0$ and $A_1 \subseteq A_0$. According to the induction hypothesis, there is a rooted q-search-tree $(T', \alpha', \beta', r')$ of partial width at most k and such that there is an edge e' incident to r' with $\alpha'(r', e') = E \setminus A_1$.
     Define a new q-search-tree (T, α, β, r) from $(T', \alpha', \beta', r')$ as follows:
     - add a new leaf r linked to r' in T', and set r as the new root,
     - put $\alpha\big(r, \{r, r'\}\big) = E \setminus A_0$, $\alpha\big(r', \{r, r'\}\big) = A_1$ and $\alpha = \alpha'$ otherwise;
     - put $\beta(r) = A_0$, $\beta(r') = \emptyset$ and $\beta = \beta'$ otherwise.

     Since $A_1 \subseteq A_0$, $\alpha(r, \{r, r'\}) \cap \alpha(r', \{r, r'\}) = \emptyset$ and (T, α, r) is a q-search-tree. Moreover, $V\big[\beta(r')\big] \cup \delta\big(\mu(r')\big) \subseteq Z_1$ and (T, α, β, r) satisfies the required conditions.
  b. if the first step of $\mathcal{S}$ is a "placing searchers" step, $\mathcal{S}'$ is a q-program with (Z_1, A_1) as a starting configuration, $Z_1 = Z_0 \cup X_1$ and $A_1 = A_0 \cup B_1$. According to the induction hypothesis, there is a rooted q-search-tree $(T', \alpha', \beta', r')$ of partial width at most k and such that there is an edge e' incident to r' with $\alpha'(r', e') = E \setminus A_1$.
     Define a new q-search-tree (T, α, β, r) from $(T', \alpha', \beta', r')$ as follows:
     - add a new leaf r linked to r' in T', and set r as the new root,
     - put $\alpha\big(r, \{r, r'\}\big) = E \setminus A_0$, $\alpha\big(r', \{r, r'\}\big) = A_0$ and $\alpha = \alpha'$ otherwise;
     - set $\beta(r) = A_0$, $\beta(r') = B_1$ and $\beta = \beta'$ otherwise.

     By construction, (T, α, β, r) is a q-search-tree that satisfies the required conditions.
  c. if the first step of $\mathcal{S}$ is a "performing a query" step, there are $p \geq 1$ distinct $(q-1)$-programs $\mathcal{S}_1, \ldots, \mathcal{S}_p$ for G such that: $\{A_0, E \setminus Y_1, \ldots, E \setminus Y_p\}$ is a partition of E, and, for any $1 \leq i \leq p$, $\mathcal{S}_i$ is a winning $(q-1)$-program for G, starting from the configuration (Z_i, Y_i) and using at most k searchers. For any $1 \leq i \leq p$, since the $(q-1)$-programs $\mathcal{S}_i$ are shorter than $\mathcal{S}$, there exists a rooted $(q-1)$-search-tree $(T_i, \alpha_i, \beta_i, r_i)$ of partial width at most k, and such that there is an edge e_i incident to r_i with $\alpha_i(r_i, e_i) = E \setminus Y_i$.
     Define a new q-search-tree (T, α, β, r) from these search-trees as follow:

- identify the roots r_i with a node r', add a new leaf r linked to r' in T', and set r as the new root,.
- put $\alpha(r, \{r, r'\}) = E \setminus A_0$, $\alpha(r', \{r, r'\}) = A_0$ and $\alpha(u, e) = \alpha_i(u, e)$ for every edge e of T_i;
- put $\beta(r) = A_0$, $\beta(r') = \emptyset$, and, for any $1 \leq i \leq p$ and any node u of T_i, $\beta(u) = \beta_i(u)$.

The rooted search-tree (T, α, β, r) has one more branching node than any search-tree $(T_i, \alpha_i, \beta_i, r_i)$ and, since each of them has at most $q-1$ branching nodes, (T, α, β, r) satisfies the required conditions.

Therefore, for any winning q-program $\mathcal{S}$ using at most k searchers with (Z_0, A_0) as starting configuration (with $\delta(A_0) \subseteq Z_0$) and such that $\mathcal{S}$ has length l, there is a rooted q-search-tree (T, α, β, r) of partial width at most k, and such that, r is incident to a unique edge $e \in E(T)$, and $\alpha(r, e) = E \setminus A_0$. This concludes the induction and the proof of the claim.

To conclude the proof, is it sufficient to note that, if $A_0 = \emptyset$, the weight of the root of the search-tree equals 0. Thus, its partial width equals its width.

3.2 From search-trees to monotone search-trees

To prove the second step of the proof, we need the following technical lemma.

Lemma 2. *Let $G = (V, E)$ be a connected graph, $\mu = \{E_1, \ldots, E_p\}$ be a (possibly degenerated) partition of E and $F \subseteq E \setminus E_1$. Set $E'_1 = E \setminus F$, $E'_i = E_i \cap F$ for $2 \leq i \leq p$ and $\mu' = \{E'_1, \ldots, E'_p\}$.*

$$\textit{If } |\delta(F)| \leq |\delta(E_1)| \textit{ then } |\delta(\mu')| \leq \big|\delta(\mu)\big|$$
$$\textit{If } |\delta(F)| < |\delta(E_1)| \textit{ then } |\delta(\mu')| < \big|\delta(\mu)\big|$$

Proof. Since $\delta(E_1) \subseteq \delta(\mu)$ and $\delta(F) \subseteq \delta(\mu')$, we get that $\big|\delta(\mu)\big| = \big|\delta(\mu) \setminus \delta(E_1)\big| + \big|\delta(E_1)\big|$ and $\big|\delta(\mu')\big| = \big|\delta(\mu') \setminus \delta(F)\big| + \big|\delta(F)\big|$. This implies that

$$\begin{aligned} \big|\delta(\mu)\big| - \big|\delta(\mu')\big| &= \Big(\big|\delta(\mu) \setminus \delta(E_1)\big| + \big|\delta(E_1)\big|\Big) - \Big(\big|\delta(\mu') \setminus \delta(F)\big| + \big|\delta(F)\big|\Big) \\ &= \big|\delta(E_1)\big| - \big|\delta(F)\big| + \Big(\big|\delta(\mu) \setminus \delta(E_1)\big| - \big|\delta(\mu') \setminus \delta(F)\big|\Big) \end{aligned}$$

To complete the proof, it is sufficient to show that $\delta(\mu') \setminus \delta(F) \subseteq \delta(\mu) \setminus \delta(E_1)$.

To prove this latter assertion, first note that any vertex $w \in \delta(E_1) \cap \delta(\mu')$ belongs to $\delta(F)$. Indeed, $w \in \delta(\mu')$ implies, by definition of μ', the existence of $e_1 \in F$ incident to w. Beside, $w \in \delta(E_1)$ implies the existence of $e_2 \in E_1$ incident to w. Since $E_1 \subseteq E \setminus F$, we have $e_1 \in F$ and $e_2 \notin F$. Therefore, $w \in \delta(F)$. Hence, we obtain that $\big(\delta(\mu') \setminus \delta(F)\big) \cap \delta(E_1) = \emptyset$. Finally, since $\delta(\mu') \setminus \delta(F) \subseteq \delta(\mu)$, it implies that $\delta(\mu') \setminus \delta(F) \subseteq \delta(\mu) \setminus \delta(E_1)$. That concludes the proof.

Lemma 3. *Let $n > 0$. Let G be a n-node connected graph, $q \geq 0$ and $k \geq 2$, (ii) $\Rightarrow$ (iii).*

ii. There is a q-search-tree of width k on G;
iii. There is a monotone q-search-tree of width k on G.

Proof. Let $\mathcal{T} = (T, \alpha, \beta, r)$ be a rooted q-search-tree of G of width k.

For every edge e of T, denote by $\mathbf{dist}(e)$ the *distance* of e to the root r. The *weight* $\mathbf{wg}(\mathcal{T})$ of $\mathcal{T}$ is $\sum_{v \in V(T)} \big|\delta\big(\mu_\alpha(v)\big)\big|$ and the *badness* $\mathbf{bn}(\mathcal{T})$ of $\mathcal{T}$ is $\sum n^{-\mathbf{dist}(e)}$ the sum being taken over the dirty edges of $\mathcal{T}$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two rooted q-search-trees. $\mathcal{T}_1$ is *tighter* than $\mathcal{T}_2$ if either $\mathbf{wg}(\mathcal{T}_1) < \mathbf{wg}(\mathcal{T}_2)$, or $\mathbf{wg}(\mathcal{T}_1) = \mathbf{wg}(\mathcal{T}_2)$ and $\mathbf{bn}(\mathcal{T}_1) < \mathbf{bn}(\mathcal{T}_2)$.

The remaining part of this lemma is devoted to prove that the tightest search-tree among any q-search-tree of width k of G is monotone. For this purpose, we make local optimisations that are compatible with the above relation. Let $\{u, v\}$ be a dirty edge of $\mathcal{T}$.

a. Let us assume that $\big|\delta\big(\alpha(u, \{u, v\})\big)\big| < \big|\delta\big(\alpha(v, \{u, v\})\big)\big|$.
   - Let us assume that v is a leaf. If $\alpha(u, \{u, v\}) = \emptyset$, just remove the leaf (by setting u as the new root, if $r = v$). Otherwise, set $\alpha(v, \{u, v\}) = E \setminus \alpha(u, \{u, v\})$ and $\beta(v) = \alpha(u, \{u, v\})$. The resulting search-tree is tighter than $\mathcal{T}$.
   - Now, let us assume that v is an internal node of T. Set $u = u_1$, let $u_2, \ldots, u_p$ be the other neighbours of v, set $\alpha(v, \{v, u_i\}) = E_i$, $\mu_v = \{E_1, \ldots, E_p\}$ and $F = \alpha(u, \{u, v\})$ so that the condition on $\{u, v\}$ can be rephrased as $\big|\delta(E_1)\big| > \big|\delta(F)\big|$. Let us modify $\mathcal{T}$ by setting $\beta(v) = \beta(v) \cap F$, $\alpha(v, vu_i) = E'_i$ for $1 \le i \le p$. Since, $E'_i \subseteq E_i$ for $2 \le i \le p$ and $E'_1 = E \setminus F$, we obtain a new q-search-tree $\mathcal{T}'$. It remains to prove that $\mathcal{T}'$ is tighteer than $\mathcal{T}$. Let η_v be the partition $\{E_1, \ldots, E_p, \beta(v)\}$ of E. Consider $\eta'_v = \{E'_1, \ldots, E'_p, \beta(v) \cap F\}$ with $E'_1 = E \setminus F$, and $E'_i = E_i \cap F$ for $2 \le i \le p$. By lemma 2, $\big|\delta(\eta'_v)\big| < \big|\delta(\eta_v)\big|$.
     Beside, $|\chi_{\mathcal{T}}(v)| = |\delta(\mu_v) \cup V[\beta(v)]| = |\delta(\eta_v) \cup (V[\beta(v)] \setminus \delta(\beta(v)))| = |\delta(\eta_v)| + |V[\beta(v)] \setminus \delta(\beta(v))| > |\delta(\eta'_v)| + |V[\beta(v) \cap F] \setminus (\delta(\beta(v) \cap F))| = |\delta(E'_1, \ldots, E'_p) \cup V[\beta(v) \cap F]| = |\chi'_{\mathcal{T}}(v)|$. Thus, $\mathcal{T}'$ has strictly smaller weight than $\mathcal{T}$. Therefore, $\mathcal{T}'$ is tighter than $\mathcal{T}$.

b. Let us assume that $\big|\delta\big(\alpha(u, \{u, v\})\big)\big| = \big|\delta\big(\alpha(v, \{u, v\})\big)\big|$. We can suppose without loss of generality that u is closer to the root r than v.
   - Let us assume that v is a leaf. We set $\alpha(v, \{u, v\}) = E \setminus \alpha(u, \{u, v\})$ and $\beta(v) = \alpha(u, \{u, v\})$. The resulting search-tree $\mathcal{T}'$ is such that $\mathbf{wg}(\mathcal{T}') \le \mathbf{wg}(\mathcal{T})$, and has smaller badness. Thus, $\mathcal{T}'$ is tighter than $\mathcal{T}$.
   - Now, let us assume that v is an internal node of T. We consider exactly the same new q-search-tree $\mathcal{T}'$ as in the second item of case a. The only difference is that using lemma 2, we only get $\big|\delta(\eta'_v)\big| \le \big|\delta(\eta_v)\big|$ and thus $\mathbf{wg}(\mathcal{T}') \le \mathbf{wg}(\mathcal{T})$. However, in $\mathcal{T}'$, the edge $\{u, v\}$ is monotone. Moreover, the only edges that were monotone in $\mathcal{T}$, and that could have become dirty are the edges $\{v, u_i\}$ for $2 \le i \le p$. Since $p \le n + 1$ and $\mathbf{dist}(\{v, u_i\}) = \mathbf{dist}(\{v, u\}) + 1$ for $2 \le i \le p$, we have
     $$\begin{aligned} \mathbf{bn}(\mathcal{T}) - \mathbf{bn}(\mathcal{T}') &\ge n^{-\mathbf{dist}(\{u,v\})} - \sum_{i=2}^{p} n^{-\mathbf{dist}(\{v,u_i\})} \\ &\ge n^{-\mathbf{dist}(\{u,v\})} - (n-1) n^{-\mathbf{dist}(\{v,u\})-1} > 0 \end{aligned}$$
     The q-search-tree $\mathcal{T}'$ is tighter than $\mathcal{T}$.

If a q-search-tree of width k has a dirty edge, we can algorithmically turn it into a new q-search-tree of width at most k which is tighter. Since there are no infinitely decreasing sequences for this relation, there exists a monotone q-search-tree of width at most k.

3.3 From monotone search-trees to tree decompositions

The two following lemmas conclude the third step of the proof of Theorem 2.

Lemma 4. *Let G be a connected graph and $\mathcal{T} = (T, \alpha, \beta, r)$ be a monotone search-tree on G. For any edge $\{u, v\}$ of T, $\alpha(u, \{u, v\}) = \beta(T_v)$ with T_v the connected component of $T \setminus \{u, v\}$ that contains v.*

Proof. We prove this by induction of $|V(T_v)|$.

- if $|V(T_v)| = 1$, then $\beta(v) = E \setminus \alpha(v, \{u, v\})$ and since $\alpha(u, \{u, v\}) = E \setminus \alpha(v, \{u, v\})$ ($\mathcal{T}$ is monotone), we have $\alpha(u, \{u, v\}) = \beta(v) = \beta(T_v)$.
- otherwise, let $w_1, \ldots, w_p$ be the neighbours of v in T_v and for $1 \leq i \leq p$, let T_{w_i} be the connected components of $T_v \setminus \{v, w_i\}$ that contains w_i. By induction hypothesis, $\alpha(v, \{v, w_i\}) = \beta(T_{w_i})$. Since T is a search-tree, the sets $\beta(v)$, $\alpha(v, \{u, v\})$ and $\beta(T_{w_1}), \ldots, \beta(T_{w_p})$ induce a partition of E, thus $\alpha(v, \{u, v\}) = E \setminus \beta(T_v)$. Since $\mathcal{T}$ is monotone, $\alpha(u, \{u, v\}) = E \setminus \alpha(v, \{u, v\}) = \beta(T_v)$ which finishes the proof.

Lemma 5. *Let G be a connected graph, $q \geq 0$ and $k \geq 2$, (iii) $\Rightarrow$ (iv).*

iii. There is a monotone q-search-tree of width k on G;
iv. There is a q-tree decomposition of width at most $k-1$ on G.

Proof. Let $\mathcal{T} = (T, \alpha, \beta, r)$ be a q-search-tree of width k.

We claim that $\Theta = \big(T, \mathcal{X}, r\big)$ with $\mathcal{X} = \{\chi(v) \mid v \text{ node of } T\}$ is a tree decomposition of width at most $k-1$. Since G is connected and $|E| > 0$, condition *ii.* of a tree decomposition implies condition *i*.

Let $\{x, y\} \in E$ be an edge of G. Since $\mathcal{T}$ is monotone, for every edge $\{u, v\}$ of T, $\{x, y\}$ belongs to either $\alpha(u, \{u, v\})$ or $\alpha(v, \{u, v\})$. Suppose $\{x, y\} \in \alpha(u, \{u, v\})$, by lemma 4, $\{x, y\} \in \beta(T_v)$ with T_v the connected component of $T \setminus \{u, v\}$ that contains v. The edge $\{x, y\}$ thus belongs to at least one $\beta(w)$ for some node w of T_v. By definition of $\chi(w)$, $\{x, y\} \subseteq \chi(w)$.

Let u, v, w be three nodes of T with v on the path $\{u, u', \ldots, v, \ldots, w', w\}$ from u to w. Let T_u (resp., T_w) be the connected component of $T \setminus \{u, u'\}$ (resp., $T \setminus \{w, w'\}$) that contains u (resp., w). Let T_u^v (resp., T_w^v) be the connected component of $T \setminus v$ that contains u (resp., w).

Let $u_1 = u', \ldots, u_p$ be the neighbours of u in T and $x \in \chi(u)$. Either there is an edge of G incident to x in $\beta(u)$, or there exist $1 < i \leq p$ such that there is an edge incident to x in $\alpha(u, \{u, u_i\})$. By lemma 4, there is an edge incident to x in $\beta(T_{u_i}) \subseteq \beta(T_u)$.

Suppose that $x \in \chi(u) \cap \chi(w)$. There exists an edge incident to x in $\beta(T_u^v) \supseteq \beta(T_u)$ and an edge incident to x in $\beta(T_w^v) \supseteq \beta(T_w)$. By lemma 4, we get that

$x \in \delta(\mu_v)$. Thus, $x \in \chi(v)$. This proves that Θ is a tree-decomposition. Moreover, by construction, $\mathbf{w}(\Theta) = \mathbf{w}(T, \alpha, \beta) - 1$. Since both $\mathcal{T}$ and Θ use the same underlying three, Θ is a q-tree decomposition of width at most $k - 1$.

References

1. J. Barát. Directed Path-width and Monotonicity in Digraph Searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.
2. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Parallel Algorithms and Architectures (SPAA'02)*, pages 200–209, 2002.
3. D. Berwanger, A. Dawar, P. W. Hunter, and S. Kreutzer. Dag-width and Parity Games. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS'06)*, LNCS 3884, Springer, pages 524–536, 2006.
4. D. Bienstock and P. D. Seymour. Monotonicity in graph searching. *Journal Algorithms*, 12(2):239–245, 1991.
5. N. D. Dendris, L. M. Kirousis, and D. M. Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172(1–2):233–254, 1997.
6. J. A. Ellis, I. H. Sudborough, and J. S. Turner. The Vertex Separation and Search Number of a Graph. *Information and Computation*, 113(1):50–79, 1994.
7. F. V. Fomin, P. Fraigniaud, and N. Nisse. Nondeterministic Graph Searching: From Pathwidth to Treewidth. In *Proceedings of the 30th international Symposium on Mathematical Foundations of Computer Science (MFCS'05)*, LNCS 3618, Springer, pages 364–375, 2005.
8. P. Fraigniaud and N. Nisse. Monotony Properties of Connected Visible Graph Searching. In *Proceedings of the 32th international Workshop on Graphs (WG'06)*, LNCS vol. 4271, Springer, pages 229–240, 2006.
9. T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed Tree-Width. *Journal of Combinatorial Theory Series B*, 82(1):138–155, 2001.
10. L. M. Kirousis and C. H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, 1986.
11. A. S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993.
12. N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1):18–44, 1988.
13. J. Obdrzálek. DAG-width: connectivity measure for directed graphs. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 814–821, 2006.
14. T. D. Parsons. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, pages 426–441, 1976.
15. N. Robertson and P. D. Seymour. Graph Minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.
16. N. Robertson and P. D. Seymour. Graph Minors. X. Obstructions to Tree-Decomposition. *Journal of Combinatorial Theory Series B*, 52(2):153–190, 1991.
17. P. D. Seymour and R. Thomas. Graph Searching and a Min-Max Theorem for Tree-Width. *Journal of Combinatorial Theory Series B*, 58(1):22–33, 1993.
18. B. Yang, D. Dyer, and B. Alspach. Sweeping Graphs with Large Clique Number. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC'04)*, pages 908–920, 2004.