Some Results on Non-deterministic Graph Searching in Trees

Omid Amini, David Coudert, Nicolas Nisse

Some Results on Non-deterministic Graph Searching in Trees*

Omid Amini[1], David Coudert[1], and Nicolas Nisse[2]

[1] Mascotte, I3S(CNRS/UNSA)-INRIA, Sophia Antipolis, France.
{omid.amini,david.coudert}@sophia.inria.fr

[2] Dep. de Ingenieria Matematica, Universidad de Chile, Santiago, Chile.
nisse@lri.fr

Abstract

Pathwidth and treewidth of graphs have been extensively studied for their important structural and algorithmic aspects. Determining these parameters is NP-complete in general, however it becomes linear time solvable when restricted to some special classes of graphs. In particular, many algorithms have been proposed to compute efficiently the pathwidth of trees. Skodinis (2000) proposes a linear time algorithm for this task.

Pathwidth and treewidth have also been studied for their nice game-theoretical interpretation, namely graph searching games. Roughly speaking, graph searching problems look for the smallest number of searchers that are sufficient to capture a fugitive in a graph. Fomin *et al.* (2005) define the non-deterministic graph searching that provides an unified approach for the pathwidth and the treewidth of a graph. Given $q \geq 0$, the q-limited search number, denotes by $s_q(G)$, of a graph G is the smallest number of searchers required to capture an invisible fugitive in G, such that the searchers are allowed to know the position of the fugitive at most q times. Roughly, $s_0(G)$ corresponds to the pathwidth of a graph G, and $s_\infty(G)$ corresponds to its treewidth. Fomin *et al.* proved that computing $s_q(G)$ is NP-complete in general, and left open the complexity of the problem restricted to the class of trees.

This paper studies this latter problem. On one hand, we give tight upper bounds on the number of queries required to search a tree when the number of searchers is fixed. We also prove that this number can be computed in linear time when two searchers are used. On the other hand, our main result consists in the design of a simple polynomial time algorithm that computes a 2-approximation of $s_q(T)$, for any tree T and any $q \geq 0$. This algorithm becomes exact if $q \in \{0, 1\}$, which proves that the decision problem associated to s_1 is polynomial in the class of trees.

1 Introduction

Graph searching problems [Par78, Bie91] have been extensively studied for practical aspects such as pursuit-evasion problems [Par78], but also for their close relationship with fundamental structural parameters of graphs, namely *pathwidth* and *treewidth*, that serve as important tools in Robertson and Seymour's Graph Minors Project [RS86]. In particular, many intractable problems can be solved in linear time when the input is restricted to graphs of bounded treewidth [BK96]. In the following, $tw(G)$ (resp., $pw(G)$) denotes the treewidth (resp., the pathwidth) of a graph G [RS86].

*This project has been supported by ACI-SI Fragile, ANR JC OSERA, ASR ResCom, European projects IST FET AEOLUS and COST 293

Graph searching is a game in which a team of searchers is aiming at capturing a fugitive hidden in a graph. The searchers can be placed at or removed from the vertices of the graph. The fugitive stands at some vertex of the graph and can move arbitrary fast from its current vertex to another by following the paths in the graph as long as it does not cross any vertex occupied by a searcher. The fugitive has perfect knowledge about the position and future moves of searchers. The fugitive is caught when it occupies the same vertex than a searcher and has no way to escape. A vertex is *contaminated* if it may harbor the fugitive, and is *cleared* by placing a searcher at it. Once cleared, a vertex remains clear as long as every path from it to a contaminated vertex is guarded by at least one searcher. Otherwise, the vertex is *recontaminated*. The graph is clear as soon as all the vertices are simultaneously clear. Therefore, the fugitive is caught. A *node search strategy* is a sequence of searchers moves (place or remove), or *steps*, that guarantees the fugitive's capture. A strategy is *monotone* if no vertices are visited twice by a searcher, i.e. *recontamination* never occurs.

Two main variants of graph searching have been particularly studied: either the fugitive is *invisible*, meaning that the searchers do not know its position unless it is caught, or it is *visible*, i.e., at any step of the strategy, the searchers know the current position of the fugitive and they can thus adapt their strategy according to this knowledge. The *node search number* $s(G)$ of a graph G is the minimum number of searchers for which a search strategy capturing an invisible fugitive exists for G [KP86]. Similarly, the *visible search number* $vs(G)$ of a graph G is defined as the minimum number of searchers required to capture a visible fugitive in G [ST93].

In [FFN05], Fomin *et al.* introduced a parametric variant of search problem, called *non-deterministic graph searching*. This new game establishes a link between visible and invisible graph searching games. Fomin *et al.* proved that the corresponding parameter, that will be precisely defined below, establishes a link between invisible and visible search numbers, namely between pathwidth and treewidth. They proved that deciding this parameter is NP-hard in general and asked whether it can be decided in polynomial time when the input is restricted to be a tree. In this paper, we study this latter problem.

In non-deterministic graph searching, the fugitive is invisible, but the searchers can query an oracle that knows the current position of the fugitive. That is, given the set W of clear vertices when the query is performed, a *query* returns a connected component C of $G \setminus W$. The vertices of C remain contaminated and those of $G \setminus C$ become clear. Obviously, the number of searchers required to catch the fugitive cannot increase when the number of allowed queries increases. More formally, a non-deterministic search strategy is a sequence of the three following basic operations:

- Place a searcher at a vertex,
- Remove a searcher from a vertex, and
- Perform a query.

This sequence must result in catching the fugitive whatever it does. The number of query-steps is however limited. In the following, it will be denoted by $q \geq 0$. The *q-limited search number* of a graph G, denoted by $s_q(G)$, is the smallest number of searchers required to catch a fugitive performing at most q query-steps. Mazoit and Nisse [MN07] generalized the monotonicity results of Bienstock and Seymour [BS91], and Seymour and Thomas [ST93]. They proved that *recontamination does not help* neither in non-deterministic case. That is, for any $q \geq 0$ and any graph G, there is a monotone search strategy performing at most q queries that uses at most $s_q(G)$ searchers [MN07]. This monotonicity result is important because monotone non-deterministic graph

searching realizes a link between treewidth and pathwidth through the notion of *q-branched tree decompositions* [FFN05] . In the following we are only interested in monotone search strategies.

Determining the q-limited search number of a graph is NP-complete in general [FFN05]. For fixed $q \geq 0$, Fomin *et al.* [FFN05] proposed an exact exponential time algorithm, in time $O(2^n n \log n)$, that computes $s_q(G)$ and the corresponding non-deterministic search strategy in any n-node graph G. Remark that for fixed $k \geq 1$, the decision version of the algorithm answers in time $O(n^{k+1})$ whether the graph can be searched with k searchers and q queries. Fomin *et al.* [FFN05] left open the design of a polynomial time algorithm computing the q-limited search number of any tree.

1.1 Our Results

In this paper, we study the problem of non-deterministic graph searching restricted to the class of trees. On one hand, we prove upper bounds on the number of queries required to clear a tree. We prove that this number is at most $\lceil |V(T)|/8 \rceil$, and that this bound is tight. Moreover, we prove that, if more than two searchers are allowed, the number of queries becomes upper bounded by $\lceil \log_2 |V(T)|/27 \rceil$. Then, we propose a linear time algorithm `TwoSearchers` that determines the minimum number of queries needed to clear a tree using two searchers (Section 3).

On the other hand, we use the algorithm of Skodinis [Sko03] to design Protocol `InitPos` that computes in polynomial time the search number of any tree when the initial positions of the searchers are imposed (Section 2.3). `InitPos` enables us to design a polynomial time algorithm, called `OneQuery`, that computes $s_1(T)$ for any tree T (Section 4). This proves that the decision problem corresponding to this parameter belongs to P. Generalizing `OneQuery`, we propose Protocol `Approx` that computes in time polynomial in n (independent in $q \geq 0$) the *restricted q-limited search number* of any n-node tree T, a new search parameter that we prove to be a 2-approximation of $s_q(T)$ (Section 5).

1.2 Related Work

Many versions of graph searching problems have been considered. They differ on various parameters. For instance, the fugitive may be arbitrary fast, or its speed may be limited (e.g. cops and robber games). In each version, either the fugitive is invisible, or it may be visible. Many other parameters have also been considered (e.g. the connectivity of the clear part, see [BFFS02],...). Indeed, in many cases, these parameters influence the relations between the considered graph searching problems and the structural properties of graphs. The problem of determining the pathwidth of a graph is NP-complete [MHG$^+$88], even for the class of star-like graphs [Gus93]. However, it can be computed in linear time for bounded treewidth graphs [BK96]. Trees have been particularly studied [Par78, MHG$^+$88, EST94]. Skodinis [Sko03] obtained a linear time algorithm with small constant factor, that computes an optimal path decomposition of any tree. More precisely,

Lemma 1 *[$q = 0$ [Sko03, MHG$^+$88]] For any n-node tree T, $s(T) \leq 1 + \log_3(n-1)$ and can be computed in time linear in n.*

Barrière *et al.* [BFFS02] gave a distributed algorithm for computing the connected search number of trees in linear time. Ellis and Markov [EM04] gave a linear time algorithm for the class of unicyclic graphs. Bodlaender and Fomin [BF02] and Coudert *et al.* [CHS07] use the weak dual of an outerplanar graph, that is a tree, to approximate its pathwidth.

2 Preliminaries

In this section, we present several notations that will be used in the following. We also propose a polynomial time algorithm that computes the smallest number of searchers required to monotonously capture an invisible fugitive in any graph, with the extra constraint that initial positions of the searchers are imposed. This algorithm will be used as cornerstone of the forthcoming algorithms.

2.1 q-branched tree decomposition.

First, let us formally define the q-limited treewidth of a graph [FFN05], and its relationship with its q-limited search number.

Given a rooted tree T, a *branching node* of T is a node with at least two children. Let $q \geq 0$. A *q-branched tree* T is a rooted tree such that every path in T from the root to a leaf contains at most q branching nodes.

Let $G = (V, E)$ be a connected graph and let $q \geq 0$. A *q-branched tree decomposition* [FFN05] of a graph G is a pair $(T, \mathcal{X})$ where T is a q-branched tree of node set I, and $\mathcal{X} = \{X_i : i \in I\}$ is a collection of subsets of V, satisfying the following three conditions:

1. $V = \cup_{i \in I} X_i$;
2. For any edge e of G, there is a set $X_i \in \mathcal{X}$ containing both end-points of e;
3. and (3) For any triple i_1, i_2, i_3 of nodes of T, if i_2 is on the path from i_1 to i_3 in T, then $X_{i_1} \cap X_{i_3} \subseteq X_{i_2}$.

The *width* of a q-branched tree decomposition is defined as $w(T, \mathcal{X}) = \max_{i \in I} |X_i| - 1$. The *$q$-branched treewidth*, $tw_q(G)$, of a graph G, is the minimum width of any q-branched tree decomposition of G. Note that $tw_{q'}(G) \leq tw_q(G)$ for any $q \leq q'$. If q is unbounded, i.e., $q = \infty$, $tw_\infty(G) = tw(G)$ where $tw(G)$ denotes the treewidth of G. Moreover, $tw_0(G) = pw(G)$, where $pw(G)$ denotes the pathwidth of G. The main theorem of [FFN05] and the monotonicity result of Mazoit and Nisse [MN07] establish the link between q-limited search number and q-branched treewidth:

Theorem 2 ([FFN05]) *For any $q \geq 0$, and for any graph G, $tw_q(G) = s_q(G) - 1$.*

Moreover, any q-branched tree-decomposition with width k of G corresponds to a monotone non-deterministic search strategy for G, using at most k searchers performing at most q queries.

The following lemma easily follows from Theorem 2 and the fact that, there always is a q-branched tree-decomposition of width $tw_q(G)$ with its root of degree strictly more than 1.

Lemma 3 *Let G be a graph. For any $q \geq 1$, there exists a non-deterministic search strategy using at most q queries and $s_q(G)$ searchers, such that the first query step holds before any removing step.*

Proof. This follows from Theorem 2 and the fact that, there always is a q-branched tree-decomposition of width $tw_q(G)$ with its root of degree strictly more than 1. ■

2.2 Basic Lemmas

Lemma 4 *Let G be a graph. Let S be a monotone non-deterministic search strategy using at most k searchers, that clears G. Let C be the connected set of contaminated edges after a query step. Then, at most $k-1$ vertices adjacent to C are occupied by a searcher.*

Proof. For purpose of contradiction, let us assume that there is a step of a strategy such that a connected component of the contaminated part of G is bordered by the k searchers. During the next removing step, recontamination will occur. This contradicts the monotonicity of S. ■

Let us notice that in the case of a strategy using two searchers, Lemma 4 has the following corollary.

Corollary 5 *Let S be a non-deterministic monotone search strategy using two searchers. At any placing step, except the first one, the vertex at which the searcher is placed is adjacent to the vertex occupied by the other searcher.*

2.3 Graph searching with imposed initial searchers' positions

Let G be a graph, and $X \subseteq V(G)$ (possibly empty). A search strategy *starting* from X is a monotone (non-deterministic) search strategy the first steps of which consists in placing searchers at every vertex in X. Let $q \geq 0$. In the following, $s_q\{X\}(G)$ denotes the smallest number of searchers required to catch a fugitive, starting from X, and performing at most q queries. ¿From the decomposition point of view, $s_q\{X\}(G)$ is the smallest integer k such that there exists a q-branched tree-decomposition $(T, \mathcal{X})$ of G with width $k+1$, such that, $X_r = X$ with r the root of T.

At each step of a search strategy, a searcher that is not occupying a vertex, or occupying a vertex all neighbours of which are clear is called a *free* searcher. If several searchers are occupying a same vertex, all but one are said free.

The remaining part of this section is devoted to a polynomial time algorithm, called `InitPos` and described on Algorithm 1, that computes $s_0\{X\}(T)$, for any tree T, $X \subseteq V(T)$. This protocol will be very useful in the next sections. More precisely, we prove Theorem 6.

Theorem 6 *Let T be a tree, and $X \subseteq V(T)$. Protocol `InitPos` computes $s_0\{X\}(T)$ and a corresponding strategy in polynomial time.*

Proof. Let us assume that Protocol `InitPos` returns $k \geq 1$. Obviously, Protocol `InitPos`(T,X) computes a sequence of placement and removal of at most k searchers, starting from the vertices of X, and such that no recontamination occurs. Thus, $s_0\{X\}(T) \leq k$.

Let us assume that $s_0\{X\}(T) \leq k$, and let us consider the corresponding execution of the *for-loop*. We prove that Protocol `InitPos` computes a corresponding strategy. Indeed, let $\mathcal{S}$ be a monotone search strategy clearing T starting from X and finishing at f. We prove that we can modify $\mathcal{S}$ to order the moves of the searchers in the same ordering as our protocol.

Indeed, let consider any step s of $\mathcal{S}$ such that a searcher A stands at a vertex v all neighbors of which are clear. Let $s' > s$ be the step of $\mathcal{S}$ when A is removed from v. The removal of A can obviously be performed at any step of $\mathcal{S}$ between steps s and s'. Similarly, if s is a step of $\mathcal{S}$ such that one searcher A is standing at a vertex v a single neighbor u of which is contaminated and u,

Algorithm 1 Protocol `InitPos`(T, X) that returns $s_0\{X\}(T)$

Require: A tree T, and a subset $X \subseteq V(T)$ with $|X| \leq k$

1: **for** k from 1 to $|V(T)|$ **do**
2: Place a searcher at each vertex of X;
3: **while** it is possible **do**
4: **while** it is possible **do**
5: **if** a searcher A stands at a vertex v all neighbors of which are clear **then**
6: Remove A
7: **else if** $\exists$ one *free* searcher B, and one searcher A standing at a vertex v, a single neighbor u of which is contaminated **then**
8: Place B at u and, Remove A
9: Let X' be the set of vertices occupied by a searcher
10: **if** $\exists$ a contaminated connected component S of $T \setminus X'$ and $s(S) \leq k - |X'|$ **then**
11: Clear S using the $k - |X'|$ free searchers
12: **else if** T is clear **then**
13: Return k

let s' be the step of $\mathcal{S}$ that consists in placing a searcher at u. For any step s'' of S between s and s' such that all searchers are not occupying some vertices of $V(T)$, i.e., there exists a free searcher B, the placement of B at u, followed by the removal of A from v, can be performed at step s''.

Let us assume that at step s of $\mathcal{S}$, all vertices occupied by a searcher have at least two contaminated neighbors. Let $X' \subset V(T)$ be the set of vertices that are occupied by a searcher at this step. We prove that there exists a connected component C of the contaminated part, i.e., a connected component of $T \setminus X$ that is still contaminated, such that $s(C) \leq k - |X'|$. Let C be first connected component of the contaminated part of $T \setminus X'$ to be cleared by $\mathcal{S}$ and let s' be the step of $\mathcal{S}$ when this occurs, i.e. all vertices of C are clear at step s'. We prove that, at any step of $\mathcal{S}$ between s and s', at least $|X'|$ searchers must occupy the vertices of $T \setminus C$.

For any $v \in X'$, let $e_v = \{v, u\}$ be the edge incident to v such that u is the neighbor of v that is closest to C (possibly, $u \in V(C)$). Let C_v be the connected component of $T \setminus e_v$ that contains v. We prove by induction on $|X' \cap V(C_v)|$ that, at any step of $\mathcal{S}$ between s and s', at least $|X' \cap V(C_v)|$ searchers are occupying the vertices of C_v. If $|X' \cap V(C_v)| = 1$, since v has at least two contaminated neighbors, at least one connected components $C' \neq C$ of the contaminated part of $T \setminus X'$ has an edge incident to v. Note that we can take $C' \subseteq C_v$. Since C is cleared before C' and the strategy is monotone, at any step between s and s', at least one searcher occupies a vertex of $C' \cup \{v\} \subseteq C_v$. Let us assume $|X' \cap V(C_v)| > 1$. Let $Y \subseteq (X' \cap V(C_v)) \setminus \{v\}$ be the set of vertices $y \neq v$ that are in $X' \cap V(C_v)$, such that no vertex of X' is on the path between y and v. According to the induction hypothesis, for any $y \in Y$, and for any step between s and s', $|X' \cap V(C_y)|$ are occupying C_y. Moreover, at least one connected components $C' \neq C$ of the contaminated part of $T \setminus X'$ has an edge incident to v. Note that, by definition of Y, for any $y \in Y$, $C' \cap C_y = \emptyset$. Since C is cleared before C' and the strategy is monotone, at any step between s and s', at least one searcher occupies a vertex of $C' \cup \{v\} \subseteq C_v$. Thus, for any step between s and s', $|X' \cap V(C_v)| = 1 + \sum_{y \in Y} |X' \cap V(C_y)|$ searchers must occupy vertices of C_v. To conclude, let $W \subseteq X'$ be the occupied vertices (at step s) incident to an edge of C. $|X'| = \sum_{v \in W} |X' \cap V(C_v)|$ must occupy the vertices of $T \setminus C$ between step s and s' of $\mathcal{S}$. Thus, $s(C) \leq k - |X'|$.

This concludes the proof of the fact that, if $s_0\{X\}(T) \leq k$, Protocol `InitPos`(T,X) returns k.

At each execution of the *while-loop* (line 3) of the protocol, either at least one vertex is cleared, or one searcher becomes free. Therefore, there are at most n^2 execution of this loop. Moreover, each

execution of this loop tests at most n vertices in constant time, and $O(n)$ contaminated components S by computing $s(S)$. Since, by lemmas 1, this latter computation is performed in polynomial time, each execution of the *for-loop* of Protocol `InitPos` is performed in polynomial time (independent of k). ■

3 On the number of non-deterministic steps.

This section is devoted to prove tight upper bounds on the number of queries required to clear any tree. Then, we design Protocol `TwoSearchers` that computes the smallest number of queries required to clear a tree using two searchers.

3.1 Upper and lower bounds on the number of queries.

First, we prove that there is a gap on the number of queries required to clear a tree, when the number of searchers allowed grows from two to three. More precisely, we prove that $\Omega(n)$ queries may be necessary to clear a tree using two searchers, whereas $O(\log n)$ queries are sufficient to clear a tree using more than two searchers. For any tree T and $k \geq 2$, let $q_k(T)$ be the smallest number of queries required to clear T using at most k searchers.

Theorem 7 *For any n-node tree T, $q_2(T) \leq \lceil n/8 \rceil$. Moreover, this bound is tight.*

Proof. Let T be a n-node tree. We present a strategy such that (1): after the first query, more than $\lfloor n/2 \rfloor$ vertices have been cleared, and (2) each of the other queries clears at least 4 vertices.

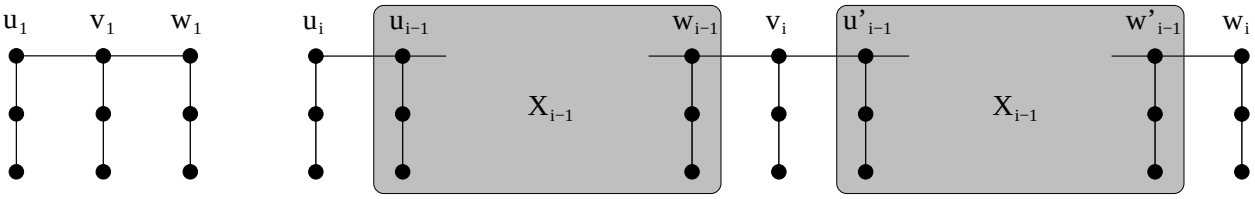
First place a searcher at the centroid of the tree, then perform a query. Now, after each query, let v be the clear vertex incident to the contaminated part, and let u be its contaminated child. Then, place a searcher at u. There are three cases. First, if there is at most one child u' of u such that $|V(T_{u'})| \geq 2$, we use the free searcher to clear the leaves adjacent to u, then we place the free searcher at u' that, now, plays the role of u. If there are at least three children u' of u such that $|V(T_{u'})| \geq 2$, or at least two children u' of u such that $|V(T_{u'})| \geq 4$, then perform a query. Else, there is exactly one child u' of u $|V(T_{u'})| \geq 4$ and one child u'' of u such that $2 \leq |V(T_{u''})| \leq 3$. In this case, we place a searcher at u' and perform the query. It remains the case when u has two children u' such that $|V(T_{u'})| = 2$ and all other children of u are leaves. In this case, we perform a query, and this is the last one. The result follows easily.

Let $q \geq 1$. Let P_q be the path $\{v_1, v_2, , \cdots, v_r\}$, with $r = 5$ if $q = 1$, and $r = 4q - 1$ else. Let T_q be the tree got from P_q, by adding a 2-node path incident to v_{2i+1}, $1 \leq i < 2q - 1$. Note that, $|V(T_q)| = 8q - 5$. We prove that $q_2(T_q) = q = \lceil |V(T_q)|/8 \rceil$. If $q = 1$, the result is obvious. Let us assume that $q > 1$. Let S be monotone strategy that clears T_q using two searchers. According to Lemma 3 and Corollary 5, we may assume that S starts by placing searchers at v_j and v_{j+1}, then performs a query. By symmetry, let us assume that $2q \leq j$. Let us assume that the fugitive remains in the component containing v_1. Thus, S must consist in the following. For any $1 \leq k < \lceil j/2 \rceil$, S places searchers at v_{j-2k} and v_{j+1-2k}, and then S performs a query. Thus, $q_2(T_q) \geq q$. Moreover, if S starts by placing searchers at v_{2q}, then S performs at most q queries. Hence, $q_2(T_q) = q$. ■

Theorem 8 *For any n-node tree T, and any $k > 2$, $q_k(T) \leq O(\log_2 n)$.*

Proof. We prove that $q_3(T) \leq O(\log_2 n)$. The strategy simply consists in iteratively: placing a free searcher at the centroid of the contaminated part, and then performing the query. It is easy to check that there always be at least one free searcher. Moreover, after every query, the number of contaminated vertices has been divided by at least two. ■

We will now prove the tighter that $q_k(T) \leq q_3(T) \leq \lceil \log_2 |V(T)|/27 \rceil$. For that, let T_s be a subtree of a tree T bounded by f nodes. For example in Figure 1(b), X_{i-1} is a subtree of X_i which is bounded by nodes u_{i-1} and w_{i-1}. Now let $q_k^f(T_s, \{u_1, u_2, \ldots, u_f\})$ denote the smallest number of queries required to search the subtree T_s using k searchers, f of them being initialy placed on nodes $u_1, u_2, \ldots, u_k$, and taking into account that the fugitive should not be able to escape T_s. Clearly, when $f = 0$ we have $q_k^0(T) = q_k(T)$.



(a) X_1 (b) X_i, $i \geq 2$, obtained using two copies of X_{i-1}

Figure 1: Familly of trees X_i, $i \geq 1$, with $9(2^i - 1)$ nodes and such that $q_3^2(X_i, \{u_i, w_i\}) = i$.

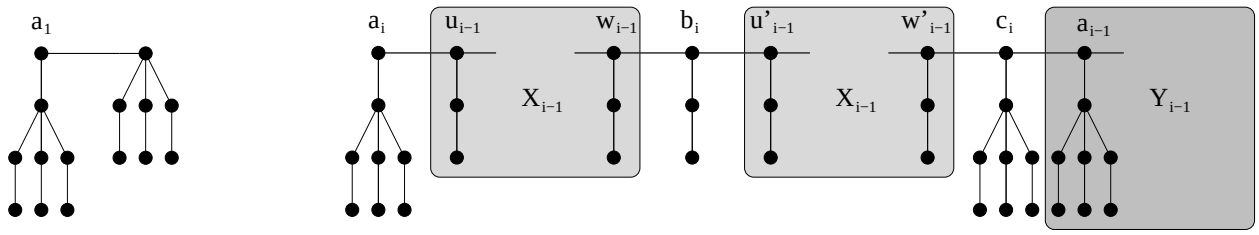
Lemma 9 *Let X_i be the subtree described in Figure 1(b). We have $q_3^1(X_i, \{u_i\}) = 0$ and $q_3^2(X_i, \{u_i, w_i\}) = i$.*

Proof. One can easily check that $q_3^1(X_i, \{u_i\}) = 0$ and that $q_3^2(X_1, \{u_1, w_1\}) = 1$. Now suppose that $q_3^2(X_{i-1}, \{u_{i-1}, w_{i-1}\}) = i - 1$ and let us show that $q_3^2(X_i, \{u_i, w_i\}) = i$.

First, we can place the third searcher on node v_i before the first query. If the fugitive belongs to one of the paths of length 2 attached respectively to u_i, v_i or w_i, then we can find him directly. Otherwise and w.l.o.g, the fugitive is in the subtree composed of X_{i-1} plus nodes u_i and v_i. Since the searcher placed on w_i can be removed, we can place it on u_{i-1}. Therefore, we can remove the searcher from u_i and put it on w_{i-1}. So it remains to search X_{i-1} which can be done with $q_3^2(X_{i-1}, \{u_{i-1}, w_{i-1}\}) = i - 1$ queries, and we have $q_3^2(X_i, \{u_i, w_i\}) \leq 1 + i - 1 = i$.

Since we need two searcher to search a path of length 2, the third searcher can only be used to reduce the search space with a query. W.l.o.g. we can place it on a node $x \in \{v_i, \ldots, w'_{i-1}\}$ and a query will divide the search space into 3 paths of length 2 attached respectively to u_i, x and w_i, plus a subtree X_y bounded by u_{i-1} and y (y being the predecessor of x along the path $\{u_i, \ldots, x\}$), and plus a subtree X_z bounded by z (the successor of x) and w_{i-1}. Since $x \in \{v_i, \ldots, w'_{i-1}\}$, we have $X_{i-1} \subseteq X_y$ and $q_3^2(X_y, \{u_{i-1}, y\}) \geq i - 1$. So $q_3^2(X_i, \{u_i, w_i\}) \geq 1 + q_3^2(X_{i-1}, \{u_{i-1}, w_{i-1}\}) \geq i$. ■

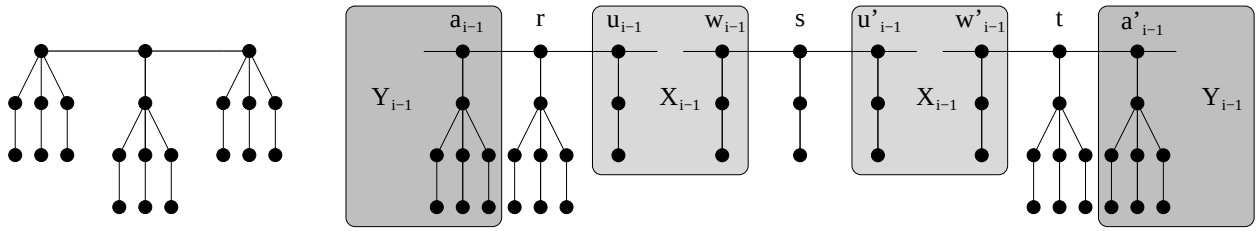
Lemma 10 *Let Y_i be the subtree described in Figure 2(b). We have $q_3^1(Y_i, \{a_i\}) = i$.*



(a) Y_1 (b) Y_i, $i \geq 2$, obtained using two copies of X_{i-1} and one copy of Y_{i-1}

Figure 2: Familly of trees Y_i, $i \geq 1$, with $18.2^i + i - 22$ nodes and such that $q_3^1(Y_i, \{a_i\}) = i$.

Proof. The proof is similar to the proof of Lemma 9. The main argument is that since $q_3^1(Y_{i-1}, \{a_{i-1}\}) = i - 1$, then we need to put a searcher on node c_i before the query. Otherwise i queries will not be sufficient. We also need to put a searcher on node b_i before the query since the subtree bounded by a_i and c_i is equivalent to X_i. ■



(a) Z_1 with 19 nodes (b) Z_i, $i \geq 2$, obtained using two copies of X_{i-1} and two copies of Y_{i-1}

Figure 3: Familly of trees Z_i, $i \geq 1$, with $27.2^i + 2i - 40$ nodes when $i \geq 2$ and 19 nodes for Z_1, and such that $q_3(Z_i) = i$.

Lemma 11 *Let Z_i be the tree described in Figure 3(b). We have $q_3(Z_i) = i$.*

Proof. Clearly, $q_3(Z_1) = 1$. Now, remark that the right subtree bounded by node r is isomorphic to Y_i. Thus we need at least i queries if we put a searcher on r before the the first query. Furthermore, we since the left subtree bounded by a_{i-1} require $i - 1$ queries, we need to put a searcher on node r before the first query and if we put a searcher on r, s and t, we can search Z_i with i queries. So $q_3(Z_i) = i$. ■

Algorithm 2 Protocol `TwoSearchers`(T) that returns $q_2(T)$

Require: A tree T
1: $q \leftarrow |V(T)|$
2: **for all** $r \in V(T)$ **do** {Suppose T rooted in r}
3: **while** it remains an unlabelled vertex $v \in V(T)$ **do**
4: Let v be an unlabelled vertex every children of which have a label
5: **if** v is a leaf **then**
6: label v with (0,0)
7: **else**
8: let (ℓ, c) be the greatest label of the children of v
9: **if** v has only one non-leaf child and $c \in \{1, 2\}$ **then**
10: label v with $(\ell, 2)$
11: **else if** $c = 0$ and exactly one child of v is labelled (ℓ, c) **then**
12: label v with $(\ell, 1)$
13: **else**
14: label v with $(\ell + 1, 0)$.
15: let (q_r, c_r) be the label of the root
16: **if** $q_r < q$ **then**
17: $q \leftarrow q_r$
18: **return** q

3.2 To search a tree using two searchers.

In this section, we design a algorithm, called `TwoSearchers`, that computes the smallest number of queries required to clear any tree using its visible search number. Protocol `TwoSearchers` is formally described on Algorithm 2.

Before describing Protocol `TwoSearchers`, let us do some easy observations that informally explain Protocol `TwoSearchers`. Let T be a rooted tree, and let $v \in V(T)$. For any vertex $u \in V(T)$, let $q(u)$ denote the smallest number of queries required to search T_u using two searchers and such that we first place a searcher at u and then perform the first query. Let us consider a step when a searcher is occupying v and the fugitive stands at some vertex in T_v. Then, three cases may occur.

- Either there is $u \in V(T_v)$ such that $C_{\{u\}}(v)$ is a caterpillar with ends u and v (line 9 of Protocol `TwoSearchers`). In this case, two searchers can easily clear $C_{\{u\}}(v)$ starting from v and finishing by u, without performing any query. Then, $q(v) = q(u)$ (line 10).

- Or v has a least two non-leaf children $v_1, \cdots, v_d$ such that $q(v_1) \geq \cdots \geq q(v_d)$ and $q(v_1) = q(v_2)$. Then $q(v) = q(v_1) + 1$ (line 14). Indeed, a query has to be performed at this step and, w.l.o.g., the second searcher does not stand in T_{v_1} while the fugitive does. By corollary 5, the next step must consist in placing the second searcher at v_1.

- Or, v has a least two non-leaf children $v_1, \cdots, v_d$ such that $q(v_1) \geq \cdots \geq q(v_d)$ and $q(v_1) > q(v_2)$ (line 11). Then, at most $q(v_1)$ queries are required to search T_v using two searchers and starting from v (line 12). Indeed, we can place the second searcher at v_1 and perform the query. The next step consists in placing the free searcher (which is not incident to the contaminated part anymore) at the neighbor of v or v_1 that belongs to the contaminated component.

More precisely, we prove the following theorem.

Theorem 12 *For any tree T with n vertices and at least an edge, Protocol* `TwoSearchers`*(T) computes the smallest $q \geq 0$ such that $s_q(T) = 2$ in time $O(n^2)$.*

Proof. Let T be a rooted tree whose any vertex has been labelled by Protocol `TwoSearchers` during an execution of the *for-loop* (line 2). Let (q, c) be the label of the root. It is easy to see that (q, c) is the greatest label (for the usual lexicographical order) of any vertex of T. First place a searcher at the vertex v which initially is the root. Then, do the following recursively. Use the second searcher to clear all leaves incident to v. Then, if $c = 2$, place it at the single non-leaf child of v. Else, if $c = 1$, place the second searcher at the unique child of v labelled $(\ell, 0)$. Then, if $c = 1$ or $c = 0$, perform a query, and after that, place the free searcher at the contaminated vertex incident to the guarded vertex. Note that this latter vertex is labelled (ℓ', c') with $\ell' \leq \ell - 1$. This strategy obviously clears T using two searchers and at most q queries. Therefore, $s_q(T) = 2$.

Let S be a monotone strategy that clears T using two searchers and at most $q \geq 0$ queries. Let us assume that the first step of S consists in placing a searcher at $r \in V(T)$. We prove by induction on $q \geq 0$ that there is $c \in \{0, 1, 2\}$ and $q' \leq q$ such that, if T is rooted in r, r will eventually be labelled (q', c) by the corresponding execution of the *for-loop* of Protocol `TwoSearchers`. For $q = 0$, this is obvious because T is a caterpillar whose one end is r. Let us assume that $q \geq 1$. By Lemma 3 and Corollary 5, we may assume that, after having placed the first searcher at r, either the first query is performed or the second searcher is placed at a child v of r and then, the first query is performed. Note that v and r may be taken as non-leaves. If not, we can modify a little bit S without increasing the number of queries. For any child $w \neq v$ of v or r, there is a strategy S_w that clears T_w using two searchers and at most $q - 1$ queries and starting by placing a searcher at w. Thanks to the induction hypothesis, w is eventually labelled (q'', c') by Protocol `TwoSearchers`, with $q'' \leq q - 1$. Thus, r will eventually be labelled (q', c) by Protocol `TwoSearchers` with $q' \leq q$. Therefore, if $s_q(T) = 2$, Protocol `TwoSearchers`(T) computes at most q. ■

Protocol `TwoSearchers` performs in quadratic time but by slightly modifying it, it is easy to compute $q_2(T)$ is linear time. Indeed, instead of trying all vertices as the root, it is sufficient to label the vertices in such a way that a vertex is labelled only if no other vertex could have been labelled with a smaller label (in the lexicographical order).

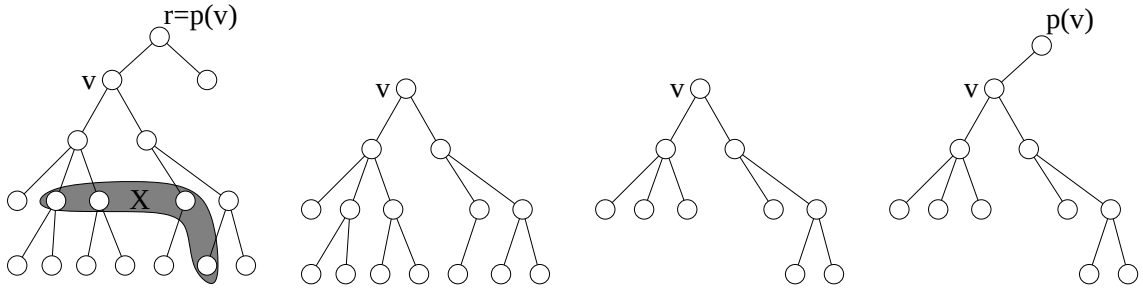
4 To clear a tree performing one query

This section is devoted to prove that Protocol `OneQuery`, formally described on Algorithm 3, computes the 1-limited search number of any tree and the corresponding strategy, in polynomial time.

Let us first set some notations and prove a basic lemma. Let T be a tree rooted in $r \in V(T)$. Let $v \in V(T)$. Let $p(v)$ denote the parent of v (we set $p(r) = \emptyset$) and let T_v denote the subtree of T rooted in v, and $\hat{T}_v$ denotes the subtree induced by T_v and $p(v)$. Let $X \subseteq V(T_v) \setminus \{v\}$ such that no vertices of X are on the path between v and another vertex of X. Let $S_X(v)$ be the component of $T_v \setminus X$ that contains v. Let $C_X(v)$ (resp., $\hat{C}_X(v)$) be the subtree of T_v induced by $V(S_X(v)) \cup X$ (resp., $V(S_X(v)) \cup X \cup p(v)$) (see Figure 4).

Let T be a rooted tree, $I \subseteq V(T)$ and $v \in V(T) \setminus I$. $X(v, I) \subseteq I$ denotes the set of descendants u of v that are in I such that no internal vertices of the path between u and v belong to I.

The remaining part of this section is devoted to prove the following theorem.



(a) T (b) T_v (c) $C_X(v)$ (d) $\hat{C}_X(v)$

Figure 4: Notations.

Algorithm 3 Protocol `OneQuery`(T) that returns $s_1(T)$

Require: A tree T

for k from 1 to $|V(T)|$ **do**
  for all $r \in V(T)$ **do** {Suppose T rooted in r}
    while it remains a unlabelled vertex $v \in V(T)$ **do**
      Let v be an unlabelled vertex every children of which have a label
      if v is a leaf **then**
        return 0
      else
        let $B(v)$ be the set of descendant of v that are labelled 1 and $X = X(v, B(v))$
        if $s_0\{X \cup p(v)\}(\hat{C}_X(v)) \leq k$ **then**
          label v with 0
        else
          label v with 1
    if At most k vertices are labelled with 1 **then**
      return k

Theorem 13 *For any tree T, Protocol `OneQuery`(T) computes $s_1(T)$ and a corresponding 1-limited search strategy in polynomial time.*

Lemma 3 has an important corollary when $q = 1$. Indeed, it means that for any graph G, there exists a 1-limited search strategy that consists of: (1) placing searchers at every vertices of some subset $X \subseteq V(G)$, with $|X| \leq s_1(G)$, (2) performing the query, and (3) clearing the contaminated component C of the graph using $s_1(G)$ searchers, and starting from the vertices of X that are adjacent to C. In other words, for any tree T and any $k \geq 1$, $s_1(T) \leq k$ if and only if there exists $X \subseteq V(T)$ such that, $|X| \leq k$, and, for any connected component C of $T \setminus X$, $s_0\{Y\}(C') \leq k$, with Y the set of vertices in X that are adjacent to a vertex in C, and C' the connected component of T induced by C and Y. Roughly speaking, given $k \geq 1$, Protocol `OneQuery` aims at finding a subset $X \subseteq V(T)$ such that the connected components of $T \setminus X$ can be cleared in the way described above, and such that these components are largest as possible. Performing in such a way allows to minimize the size of X. Finally, if $|X| \leq k$, then $s_1(G) \leq k$.

Roughly speaking, Protocol `OneQuery` proceeds as follows. $k \geq 1$ being fixed, and T being rooted in $r \in V(T)$, Protocol `OneQuery` labels the vertices of T from the leaf toward the root with 0 and 1. Intuitively, at the end of this labelling, the set $X \subseteq V(T)$ that we are looking for will be the set of the vertices labelled with 1. That is, if at most k vertices are labelled with 1, a

1-limited search strategy starts by placing the searchers at these vertices and then, performing the query. Indeed, given $C \subseteq V(T)$ a maximal-inclusion connected subset of vertices labelled 0, and $X \subseteq V(T)$ the set of vertices labelled 1 that are adjacent to a vertex in C, it is easy to verify that $s_0\{X\}(C \cup X) \leq k$. Therefore, if Protocol `OneQuery` returns k, then $s_1(T) \leq k$.

Let us detail the labelling process. A vertex $v \in V(T)$ is labelled when all its children have already been labelled. $X(v)$ is defined as the set of descendants u of v that are labelled with 1, and such that the internal vertices of the path between u and v are labelled 0. Assuming that searchers will be placed at any vertices of $X(v)$ before the first query, Protocol `OneQuery` tests whether a searcher also must be placed at v. This is the case if, by not placing a searcher at v before the query, it creates a connected component that may remain contaminated after the query, and that cannot be cleared using at most k searchers starting from the current positions of the searchers just after the query. In this case, v is labelled with 1, and with 0 otherwise.

To prove that, if $s_1(T) \leq k$, Protocol `OneQuery` returns k, let us consider any 1-limited search strategy S for T using k searchers. Let $X \subseteq V(T)$ be the set of vertices occupied by a searcher before the query, and let $r \in X$. We consider the execution of `OneQuery` when T is rooted in r. Let $v \in V(T_r)$ and let $j = |X \cap V(T_v)|$. We prove by induction on j that at most j vertices in T_v are labelled with 1. Since $|X \cap V(T_r)| = |X| \leq k$, this implies that the number of vertices with level 1 is at most k. The fact that `OneQuery` performs in polynomial time directly follows from Theorem 6.

We can now give a formal proof of the correctness of Protocol `OneQuery`.

Proof. (Theorem 13) Let $k \geq 1$ be the integer computed by `OneQuery`(T). The strategy consists in placing the searchers at the vertices labelled 1, and performing the query. Then, the connected component that remains contaminated can be cleared by k searchers starting from the set of vertices labelled 1. Thus, $s_1(T) \leq k$.

To prove the reverse inequality, let S be a monotone 1-limited search strategy for T using at most k searchers. By Lemma 3, we may assume that S first places the at most k searchers at the vertices of $I \subseteq V(T)$ and then performs the query. Let $r \in I$. Assume that T is rooted in r, and let us consider the labelling of the vertices of T performing by `OneQuery`(T). Let $v \in V(T)$ and let $j = |I \cap V(T_v)|$. We prove by induction on j that there are at most j vertices labelled with 1 in T_v. Since $|I \cap V(T_r)| = |I| \leq k$, this implies that the number of vertices with level 1 is at most k.

Let us prove the induction. If $j = 0$, then for any $w \in V(T_v)$, $s_0\{p(w)\}(\hat{T}_w) \leq k$. Otherwise, any monotone 1-limited search strategy for T using at most k searchers must let a searcher occupy a vertex of T_v before the query. Thus, no vertices of T_v are labelled 1 by `OneQuery`. Let us assume $j > 0$. If $v \in I$, the result is straightforward by applying the induction hypothesis on its children. Let us assume $v \notin I$. Let $X = \{v_1, \cdots, v_\ell\}$ be the set of descendants u of v that are in I and such that there are no vertices of I on the path between v and u. Let us remark that any vertices labelled with 1 in $\hat{C}_X(v)$ must belong to a path between v_i and v for some i, $1 \leq i \leq \ell$. Otherwise, there would be a 1-vertex w with $I \cap T_w = \emptyset$ contradicting the induction hypothesis. Moreover, since S is monotone, $\hat{C}_X(v)$ can be cleared by k searchers starting from $X \cup p(v)$, without performing any query. Therefore, for any i, $1 \leq i \leq \ell$, there is at most one 1-vertex on the path between v_i and v. Hence, at most ℓ vertices labelled with 1 are in $C_X(v)$. Finally, by induction hypothesis, $T_v \setminus (C_X(v))$ contains at most $j - \ell$ vertices labelled with 1. This concludes the proof.

The fact that `OneQuery` performs in polynomial time directly follows from Theorem 6. ■

5 2-approximation of the q-limited search number of a tree

In this section, we introduce a restricted version of non deterministic graph searching. We prove that this new parameter, denoted by rs_q, provides a 2-approximation of s_q. Then we generalize the ideas presented in Section 4, to obtain a polynomial time algorithm for the restricted non-deterministic graph searching parameter in trees.

5.1 Restricted non-deterministic graph searching

We consider a particular kind of non-deterministic search strategy. A *restricted non-deterministic search strategy* is a monotone non-deterministic search strategy such that the moves are ordered in a particular way. Initially, the searchers can be placed at the vertices and the first query is performed. Then, while a query still can be performed, the strategy consists in first removing the searchers occupying a vertex all neighbours of which are clear, then placing searchers at some vertices in the contaminated part, and finally performing the query. When there is no query left, the strategy proceeds as usual. In other words, in this variant of non-deterministic graph searching, once some searchers have been placed at contaminated vertices, no searchers can be removed while the next query has not been performed.

The *restricted q-limited search number* of a graph G, denoted $rs_q(G)$, is the smallest number of searchers required to catch a fugitive performing at most q query steps, in a restricted non-deterministic way.

A *restricted q-branched tree* is a q-branched tree such that, for any vertex v that belongs to a path between two vertices of degree at least 3, either v is the root and have degree at least 2, or v has degree at least 3. A *restricted q-branched tree decomposition* of a graph G is a tree decomposition $(T, \mathcal{X})$ where T is a restricted q-branched tree. The *restricted q-branched treewidth*, $rtw_q(G)$, of a graph G, is the minimum width of any restricted q-branched tree decomposition of G. The proof of the following theorem is similar to the proof of Theorem 2 in [FFN05], and is omitted.

Theorem 14 *For any $q \geq 0$, for any graph G, $rtw_q(G) = rs_q(G) - 1$.*

Theorem 15 *For any $q \geq 2$, for any graph G, $rtw_q(G) \leq 2\ tw_q(G)$.*
For any $q \in \{0, 1\}$, for any graph G, $rtw_q(G) = tw_q(G)$.

Proof. Let $(T, \mathcal{X})$ be a q-branched tree decomposition of G. Let r be the root of T. Let us assume there exist v and w, vertices of $V(T)$ with degree at least 3 such that, there is a path $\{w, u_1, \cdots, u_l, v\}$ in T, and for any i, $1 \leq i \leq l$, u_i has degree 2, and is different from the root. W.l.o.g., let us assume that v is a descendant of w. For any i, $1 \leq i \leq l$, let $U_i = X_v \cap X_{u_i}$. We modify T by removing the edge $\{u_1, w\}$, and by adding an edge $\{v, w\}$. Moreover, we replace X_w by $X_v \cup X_w$, and X_{u_i} by U_i. By repeating this process, we will obtain a restricted q-branched tree decomposition of width at most twice the width of $(T, \mathcal{X})$.

For $q = 0$, the result is obvious, and, for $q = 1$, the result follows Lemma 3. ■

5.2 Polynomial time 2-approximation algorithm

This section is devoted to a polynomial time algorithm that computes $rs_q(T)$ for any tree T and any $q \geq 0$. Before presenting Protocol `Approx`, that is formally described on Algorithm 4, we need

Algorithm 4 Protocol `Approx`(T, q) that returns $rs_q(T)$

Require: A tree T

1: **for** k from 1 to $|V(T)|$ **do**
2: **for all** $r \in V(T)$ **do** {Suppose T rooted in r}
3: **while** it remains an unlabelled vertex $v \in V(T)$ **do**
4: Let v be an unlabelled vertex every children of which have a label
5: Let q be the greatest label among the labels of all descendants of v
6: Let b_j be the number of vertices labelled $\geq j$ in $C_{X(v,j+1)}(v)$, $1 \leq j \leq q$
7: Let $k' = k$ if v is the root and $k' = k - 1$ otherwise
8: Let $m \leq q$ be the greatest integer such that $b_m = \max_{1 \leq i \leq q} b_i$
9: Let $p > m$ be the smallest integer such that $b_p < k'$
10: Let $t \geq 1$ be the smallest integer such that $b_t < k'$
11: **if** v is a leaf **then**
12: **return** 0
13: **else if** $b_m \leq k'$ and $|X(v, m)| < k'$ **then**
14: **if** $s_0\{X(v, I) \cup \{p(v)\}\}(\hat{C}_{X(v,I)}(v)) \leq k$ **then**
15: label v with 0
16: **else**
17: label v with t
18: **else if** $b_m > k'$ **then**
19: label v with p
20: **else**
21: label v with $m + 1$
22: **if** a vertex is labelled more than q **then** Goto line 2
23: **return** k

to introduce again a new notation (this is the last one). Let us assume some vertices of a tree T are labelled with integers in $\{0, \cdots, q\}$, and $v \in V(T)$ is not labelled. Let $0 \leq i \leq q$. We set $X(v, i)$ to be the set $X(v, I)$ where I is the set of vertices of T labelled at least i.

Now we can define Protocol `Approx` that aims at computing $rs_q(T)$ for any tree T and any $q \geq 0$. $k \geq 1$ being fixed, Protocol `Approx` computes the smallest $q \geq 0$ such that there exists $X \subseteq V(T)$, with $|X| \leq k$, and for any connected component C of $T \setminus X$, $rs_{q-1}\{Y\}(C') \leq k$, with Y the set of vertices in X that are adjacent to a vertex in C, and C' the connected component of T induced by C and Y. Similarly to Protocol `OneQuery`, Protocol `Approx` aims at finding a subset $X \subseteq V(T)$ such that the connected components of $T \setminus X$ can be cleared in the way described above, and such that these components are largest as possible.

Intuitively, k being fixed, if `Approx` labels all vertices of T with non-negative integer smaller or equal to $q \geq 1$, it is easy to compute a restricted q-limited search strategy for T using at most k searchers: place the searchers at the vertices labelled q, then perform the first query. Before performing the ℓ^{th} query ($2 \leq \ell \leq q$), remove those searchers that are occupying a vertex adjacent to clear vertices only, place searchers at the vertices of the contaminated part that are labelled with $q - \ell + 1$, and performed the ℓ^{th} query. When all the queries have been performed, clear the remaining contaminated part starting from the current position of the searchers. This sketches the proof that if `Approx` returns $k \geq 1$, then $rs_q(T) \leq k$. The proof of the reverse inequality is similar to the proof of Theorem 13, in the case $q = 1$. The time complexity of the *while-loop* of Protocol `Approx` is dominated by the execution of line 14. By Theorem 6, `InitPos` performs the computation of $s_0\{X(v, I) \cup \{p(v)\}\}(\hat{C}_{X(v,I)}(v))$ in time polynomial in $n = |V(T)|$. Moreover, at each execution of the *while-loop*, at least one vertex is cleared, or a searcher is removed, unless line 18 is executed which completes the *while-loop*. Thus, `Approx` performs in time $O(n^3 t(n))$, where

$t(n)$ is the time complexity of `InitPos`. Hence, Protocol `Approx` performs in polynomial time in the size of the input tree (independent in q).

Let us now be more formal and let us assume that `Approx` returns $k \geq 1$, and the vertices of T have been labelled with labels $0 \leq \ell \leq q$. For any ℓ, $1 \leq \ell \leq q$, we will prove that it means that our labelling divides T into subtrees (as large as possible) covering T satisfying:

1. two distinct subtrees intersect in at most one vertex and this vertex is labelled at least $\ell + 1$;
2. any internal vertex of such a subtree is labelled at most ℓ;
3. if $\ell > 0$, any of these subtrees contains at most k vertices labelled at least ℓ;
4. any subtree intersects at most $k - 1$ other subtrees;
5. if $\ell = 0$, any subtree can be cleared by k searchers starting from its leaves labelled at least 1 and without performing a query (Note that, by Lemma 4, item 5. implies item 4. in case $\ell = 0$).

It is easy to see that if we have such a family of subtrees and no vertices labelled more than q, k searchers can clear T in a restricted way, performing at most q queries. The strategy consists in placing the searchers at vertices labelled q, then performing the first query. Then, for any ℓ, $1 < \ell \leq q$, remove those searchers that occupy a vertex all incident edges of which are clear, place the searchers at vertices labelled $q-\ell+1$ in the contaminated part, and then perform the ℓ^{th} query. After the last query, clear the remaining contaminated part C of T with k searchers starting from the vertices bordering C.

Remark: Note that, if $q = 1$, and Protocol `Approx` proceeds as `OneQuery` (Algorithm 3).

Theorem 16 *For any tree T and any integer $q \geq 0$, `Approx`(T, q) computes $rs_q(T)$ and a corresponding restricted q-limited search strategy in time polynomial in $|V(T)|$ (independent of q).*

Proof. Let us assume that `Approx`(T) computes k, and let $r \in V(T)$ be the root when it occurs. In order to prove that $rs_q(T) \leq k$, we prove that items $1 - 5$ are satisfied. Let $0 \leq \ell \leq q$. Let us consider the family $\mathcal{F}$ of subtrees of T, covering T and satisfying items 1. and 2.. We prove that items $3 - 5$ are satisfied as well. For $\ell = q$, 3. and 4. follows from line 13 applied to r. Let $S \in \mathcal{F}$, and $v \in V(S)$ be the vertex such that S is a subtree of T_v. Note that $S = C_{X(v,\ell+1)}(v)$. There are two cases to consider:

- Case v is labelled $\geq \ell + 1$. Thus, $\ell < q$. By 2., there is exactly one child u of v that belongs to S. Note that u is not the root. If u is labelled at least $\ell+1$, then $V(S) = \{u, v\}$ and items $3 - 5$ hold trivially. Let us assume that u is labelled at most ℓ. If $\ell = 0$, step 14. of `Approx` insures that item 5. is valid. We may assume that $\ell > 0$.

  If u has been labelled with 0, line 13 insures that there are at most $k - 1$ vertices labelled with at least ℓ in $S \setminus (\{u\} \cup \{v\})$. Therefore, item 3. holds. Let us assume that u is labelled more than 0.

  For purpose of contradiction, let us assume that more than $k-1$ vertices of $S\setminus\{v\}$ are labelled at least ℓ. Thus, $C_{X(u,\ell+1)}(u)$ contains $b_\ell \geq k - 1$ vertices labelled at least ℓ. If $b_\ell > k - 1$, lines 18-19 of `Approx` insures that u is labelled more than ℓ, a contradiction. Thus, we may

assume that $b_\ell = k-1$, and u cannot be labelled by lines 13-17 of `Approx` because, otherwise, u would have been labelled either with q, or with t such that $b_t < k-1$. Thus, u has been labelled by line 21 of `Approx`, a contradiction because it would have been labelled $m+1$ with $m \geq \ell$. Hence, if $\ell > 0$, item 3. holds.

It remains to prove item 4.. Assume for purpose of contradiction that S intersects more than $k-1$ other subtrees. Thus, $|X(v, \ell+1)| \geq k-1$. Again, $C_{X(u,\ell+1)}(u)$ contains $b_\ell \geq k-1$ vertices labelled at least ℓ. This leads to a contradiction as previously. Hence, item 4. holds.

- Case v is labelled $\leq \ell$. By 1., v is the root. The proof can easily be derived from the previous case.

Since the family of subtrees satisfy all properties $1-5$, it follows that $rs_q(T) \leq k$.

Let us prove that the converse holds. Let S be a monotone q-limited search strategy for T using at most k searchers. By Lemma 3, we may assume that S first places the at most k searchers at the vertices of $I \subseteq V(T)$ and then performs the first query. Let $k \geq 1$ and $r \in I$. Consider T as rooted in r. We consider the corresponding execution of `Approx`. Let $v \in V(T)$ and let $j = |I \cap V(T_v)|$. We prove by induction on j the following proposition:

Proposition 17 *For any $v \in V(T)$, there are at most j vertices labelled with q in T_v.*

Since $|I \cap V(T_r)| = |I| \leq k$, Proposition 17 implies that the number of vertices with level at least q is at most k. Thus, a vertex v would have been labelled more than q, only if $|X(v,q)| \geq k$ (condition of line 13). By Proposition 17, $|X(v,q)| \geq k$ implies that there is a connected component of $T \setminus I$ incident to at least k vertices of I, contradicting the Lemma 4. Hence, Proposition 17 implies that q is the largest level got by the vertices of T rooted in r, and at most k vertices got this level. It remains to prove Proposition 17 by induction on q. The above Remark and the proof of Theorem 13 ensure that Proposition 17 holds for $q = 1$. Let us assume that Proposition 17 holds for any q', $1 \leq q' < q$ (induction hypothesis 1). Let us prove it also holds for q.

Let us prove it by induction on j. If $j = 0$, T_v can be search using at most k searchers and $q-1$ queries. By induction hypothesis 1, no vertices of T_v are labelled more than $q-1$. Let $j > 0$. Let us assume that Proposition 17 holds if $|I \cap V(T_v)| < j$ (induction hypothesis 2) and let us prove it remains valid if $|I \cap V(T_v)| = j$. If $v \in I$, the result is straightforward by applying the induction hypothesis 2 on its children. Let us assume $v \notin I$. Let $X(v,I) = \{v_1, \cdots, v_\ell\}$. Let us remark that any vertices labelled with q in $\hat{C}_X(v,I)(v)$ must be on a path between v_i and v for some i, $1 \leq i \leq \ell$. Otherwise, there would be a vertex w labelled with q, and such that $X \cap T_w = \emptyset$, contradicting the induction hypothesis 2. Moreover, since S is monotone, $\hat{C}_X(v,I)(v)$ can be cleared by k searchers starting from $X(v,I) \cup p(v)$, performing at most $q-1$ queries in a restricted way. Therefore, for any i, $1 \leq i \leq \ell$, there is at most one vertex labelled with q on the path between v_i and v. Hence, at most ℓ vertices labelled with q are in $C_{X(v,I)}(v)$. Finally, by induction hypothesis, $T_v \setminus (C_{X(v,I)}(v))$ contains at most $j - \ell$ vertices labelled with q. This concludes the proof. ■

References

[BF02] H.L. Bodlaender and F. V. Fomin. Approximation of pathwidth of outerplanar graphs. *Journal of Algorithms*, 43(2):190–200, 2002.

[BFFS02] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 200–209, 2002.

[Bie91] D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Math. and Theoretical Computer Science*, 5:33–49, 1991.

[BK96] H.L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.

[BS91] D. Bienstock and P. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.

[CHS07] D. Coudert, F. Huc, and J.-S. Sereni. Pathwidth of outerplanar graphs. *Journal of Graph Theory*, 55(1):27–41, May 2007.

[EM04] J. Ellis and M. Markov. Computing the vertex separation of unicyclic graphs. *Inform. and Comput.*, 192(2):123–161, 2004.

[EST94] J. A. Ellis, I. H. Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Inform. and Comput.*, 113(1):50–79, 1994.

[FFN05] F.V. Fomin, P. Fraigniaud, and N. Nisse. Nondeterministic graph searching: From pathwidth to treewidth. In *30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Springer LNCS 3618, 2005.

[Gus93] J. Gustedt. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233–248, 1993.

[KP86] M. Kirousis and C.H. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, November 1986.

[MHG+88] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. Assoc. Comput. Mach.*, 35(1):18–44, 1988.

[MN07] F. Mazoit and N. Nisse. Monotonicity of non-deterministic graph searching. In *33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'07)*, 2007. to appear.

[Par78] T. D. Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Lecture Notes in Math., Vol. 642. Springer, Berlin, 1978.

[RS86] N. Robertson and P. Seymour. Graph minors ii, algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[Sko03] K. Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *J. Algorithms*, 47(1):40–59, 2003.

[ST93] P. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *J. Combin. Theory Ser. B*, 58:22–33, 1993.