Exclusive Graph Searching

Lélia Blin, Janna Burman, Nicolas Nisse

Exclusive Graph Searching*

Lélia Blin
Sorbonne Universités, UPMC Univ Paris 06, CNRS,
Université d'Evry-Val-d'Essonne.
LIP6 UMR 7606,
4 place Jussieu 75005, Paris, France
lelia.blin@lip6.fr

Janna Burman
LRI, Université Paris Sud, CNRS, UMR-8623, France.
janna.burman@lri.fr

Nicolas Nisse
Inria, France.
Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, Sophia Antipolis, France.
nicolas.nisse@inria.fr

February 2, 2016

Abstract

This paper tackles the well known *graph searching* problem, where a team of searchers aims at capturing an intruder in a network, modeled as a graph. This problem has been mainly studied for its relationship with the *pathwidth* of graphs. All variants of this problem assume that any node can be simultaneously occupied by several searchers. This assumption may be unrealistic, e.g., in the case of searchers modeling physical searchers, or may require each individual node to provide additional resources, e.g., in the case of searchers modeling software agents. We thus introduce and investigate *exclusive* graph searching, in which no two or more searchers can occupy the same node at the same time. As for the classical variants of graph searching, we study the minimum number of searchers required to capture the intruder. This number is called the *exclusive search number* of the considered graph. Exclusive graph searching appears to be considerably more complex than classical graph searching, for at least two reasons: (1) it does not satisfy the *monotonicity property*, and (2) it is not *closed under minor*. Moreover, we observe that the exclusive search number of a tree may differ exponentially from the values of classical search numbers (e.g., pathwidth). Nevertheless, we design a polynomial-time algorithm which, given any n-node tree T, computes the exclusive search number of T in time $O(n^3)$. Moreover, for any integer k, we provide a characterization of the trees T with exclusive search number at most k. Finally, we prove that the ratio between the exclusive search number and the pathwidth of a graph is bounded by its maximum degree.

1 Introduction

Graph Searching was first introduced by Breisch [5, 6] in the context of speleology, for solving the problem of rescuing a lost speleologist in a network of caves. Alternatively, graph searching

*An extended abstract of this work has been presented in ESA 2013 [4] can be defined as a particular type of cops-and-robber game, as follows. Given a graph G, modeling any kind of network, the goal is to design a *strategy* for a team of searchers moving in G resulting in capturing an intruder. There are no limitations on the capabilities of the intruder, who may be arbitrary fast, be aware of the whole structure of the network, and be perpetually aware of the current positions of the searchers. The objective is to compute the minimum number of searchers required to capture the intruder in G.

To be more formal regarding the behavior of the intruder, it is more convenient to rephrase the problem in terms of *clearing* a network of pipes contaminated by some gas [15]. In this framework, a team of searchers aims at clearing the edges of a graph, which are initially *contaminated.* Searchers stand on the nodes of the graph, and can *slide* along its edges. Moreover, a searcher can be removed from one node and then placed to any other node, i.e., a searcher can "jump" from node to another. Sliding of a searcher along an edge, as well as positioning one searcher at each extremity of an edge, results in clearing that edge. Nevertheless, if there is a path free of searchers between a *clear* edge and a *contaminated* edge, then the former is instantaneously *recontaminated.* Thus, to actually keep an edge clear, searchers must occupy appropriate nodes for avoiding recontamination to occur.

Informally, a *search strategy* is a sequence of movements executed by the searchers, resulting in all edges being eventually clear. The main question tackled in the context of graph searching is, given a graph G, compute a search strategy minimizing the number of searchers required for clearing G. This number, denoted by $\mathtt{s}(G)$, is called the *search number* of the graph G. For instance, one searcher is sufficient to clear a path graph, while two searchers are necessary (and sufficient) in a cycle: the search number of any path is 1, while the search number of any cycle is 2.

The above variant of graph searching is actually called *mixed-search* [3]. Other classical variants of graph searching are *node-search* [2], *edge-search* [14,15], *connected-search* [1], etc. All these variants suffer from two serious limitations as far as practical applications are concerned:

- First, all these works assume that any node can be simultaneously occupied by several searchers. This assumption may be unrealistic in several contexts. Typically, placing several searchers at the same node may simply be impossible in a physical environment in which, e.g., the searchers are modeling physical robots moving in a network of pipes. In the case of software agents deployed in a computer network, maintaining several searchers at the same node may consume local resources (e.g., memory, computation cycles, etc.). We investigate *exclusive* graph searching, i.e., graph searching bounded to satisfy the *exclusivity constraint* stating that no two or more searchers can occupy the same node at the same time.

- Second, most variants of graph searching also suffer from another unrealistic assumption: searchers are enabled to "jump" from one node of the graph, to another, potentially far away, node (e.g., see the classical mixed-search, defined above). We restrict ourselves to the more realistic *internal* search strategies [1], in which searchers are limited to move along the edges of the graph, that is, restricted to satisfy the *internality constraint.*

To sum up, we define *exclusive-search* as mixed-search with the additional exclusivity and internality constraints[1]. As for all classical variants of graph searching, we study the minimum number of searchers required to clear all edges of a graph G. This number is called the *exclusive search number*, denoted by $\mathtt{xs}(G)$.

[1]Note that, whenever exclusivity is not required, internality is not a strong constraint (difficult to overcome), since the jumping of one searcher from a node u to a node v can be replaced by sliding this searcher along a path from u to v. However, the combination of exclusivity and internality makes the problem much more difficult.

We demonstrate that exclusive graph searching behaves very differently from classical graph searching, for at least two reasons. First, it does not satisfy the *monotonicity property*. That is, there are graphs (even trees) in which every exclusive search strategy using the minimum number of searchers requires to let recontamination occurring at some step of the strategy. Second, exclusive graph searching is not *closed under taking minor* (not even under subgraph). That is, there are graphs G and H such that H is a subgraph of G, and $\texttt{xs}(H) > \texttt{xs}(G)$. The absence of these two properties (which will be formally established in the paper) makes exclusive-search considerably different from classical search, and its analysis requires introducing new techniques.

Our Results. First, in Section 2, we formally define exclusive graph searching and present basic properties for general graphs. Motivated by positive results for trees and inspired by the pioneering work of Parson [15] and Megiddo et al. [14], we essentially focus on trees. We observe that the exclusive search number of a graph can differ exponentially from the values of classical search numbers: in a tree, the former can be linear in the number of nodes n, while all classical search numbers of trees are at most $O(\log n)$. On the other hand, we prove that the ratio between the exclusive search number and other search numbers is bounded in the class of graphs with bounded maximum degree. Formally, we prove that $\texttt{xs}(G) \leq (\Delta - 1) \cdot (\texttt{s}(G) + 1)$ in any graph G with maximum degree Δ. Section 3 is devoted to proving some technical properties of exclusive graph searching in trees. Our main result (Section 4) is a polynomial-time algorithm which, given any n-node tree T, computes the exclusive search number $\texttt{xs}(T)$ of T in time $O(n^3)$. Our algorithm is based on a new characterization of the trees with exclusive search number at most k, for any given $k \leq n$.

Related Work. Graph searching has mainly been studied in the centralized setting for its relationship with the *treewidth* and *pathwidth* of graphs [3, 10]. In particular, the pathwidth of a graph and its mixed-search number differ by at most one [3, 12]. An important property of mixed-graph searching is the monotonicity property. A strategy is *monotone* if no edges are recontaminated once they have been cleared. For any graph G, there is an optimal winning monotone (mixed-search) strategy [3]. By extension, mixed-search is said monotone. This enables to prove that the number of steps of an optimal strategy is polynomially bounded by the number of edges. Hence, the problem to decide the mixed-search number of a graph belongs to NP.

The problem of computing the search number of a graph is NP-hard [14]. However, this problem is polynomial in various graph classes [9, 11, 18]. In particular, it has been widely studied in the class of trees [8, 14–17]. More precisely, Parsons has given a nice characterization of trees with search number k: for any $k \geq 1$, a tree T has search number at least $k + 1$ if and only if there is a node $v \in V(T)$ such that at least three components of $T \setminus v$ have search number at least k [14, 15]. This result and the relative simplicity of its proof mainly come from the monotonicity of classical graph searching and from the fact that search number is not increasing by taking subtrees.

Other variants of graph searching have been introduced to deal with practical applications. Connected graph searching, in which the set of clear edges must always induce a connected subgraph (in order to ensure safe communications between the searchers), is not monotone in general [19] and it is not known if connected search is in NP. Connected search is however monotone in trees [1]. The connectivity constraint may only increase the search number of any graph by a factor up to two [7].

Concerning exclusive graph searching, a recent work has shown that computing the exclusive search number is NP-hard in planar graphs with maximum degree 3 [13]. In the same paper, it is proved that the computational complexities of monotone exclusive graph searching and pathwidth cannot be compared. Precisely, monotone exclusive graph searching is NP-complete in split graphs where pathwidth is known to be solvable in polynomial time. Moreover, monotone exclusive graph searching is in P in a subclass of star-like graphs where pathwidth is known to be NP-hard [13].

2 Exclusive Search

In this section, we provide the formal definition of exclusive graph searching, and present some basic general properties. All graphs considered in this paper are undirected, simple and connected.

Given a connected n-node graph G, an *exclusive search strategy* in G, using $k \leq n$ searchers consists in (1) placing the k searchers at k different nodes of G, and (2) performing a sequence of *moves*. A move consists in sliding one searcher from one extremity u of an edge $e = \{u, v\}$ to its other extremity v. Such a move can be authorized only if v is free of searchers. That is, exclusive-search limits the strategy to place at most 1 searcher at each node, at any point in time. The edges of graph G are supposed to be initially contaminated. An edge becomes clear whenever either a searcher slides along it, or one searcher is placed at each of its extremities. An edge becomes recontaminated whenever there is a path free of searchers from that edge to a contaminated edge. A search strategy is *winning* if its execution results in all edges of the graph G being simultaneously clear. The exclusive-search number of G, denoted by $\mathtt{xs}(G)$ is the smallest number k for which there exists a winning search strategy in G.

Now, we state and explain the main differences between exclusive search and all classical variants of graph searching. These differences are mainly due to the combination of the two restrictions introduced in exclusive search: two searchers cannot occupy the same node (exclusivity) and a searcher cannot "jump" (internality). Intuitively, the difficulty occurs when a searcher has to go from one node u to a far away node v, and all paths from u to v contain an occupied node.

Consider a simple example of a star with central node c and f leaves. In the classical graph searching, one searcher can occupy c, while a second searcher will sequentially clear all leaves, either by jumping from one leaf to another, or by sliding from one leaf to another, and therefore occupying several times the already occupied node c. In exclusive graph searching, such strategies are not allowed. Intuitively, if a searcher r_1 has to cross a node v that is already occupied by another searcher r_2, the latter should step aside for letting r_1 pass. However, r_2 may occupy v to preserve the graph from recontamination, and moving away from v could lead to recontaminate the whole graph. To avoid this, it may be necessary to use extra searchers (compared to the classical graph searching) that will guard several neighbors of v to prevent from recontamination when r_2 gives way to r_1. It follows that, as opposed to all classical search numbers, which differ by at most some constant multiplicative factor, the exclusive search number may be arbitrary large compared to the mixed-search number, even in trees. For instance, it is easy to check that $\mathtt{xs}(S_n) = n-2$ for any n-node star S_n, $n \geq 3$. More generally,

Claim 1 *For any tree T with maximum degree $\Delta \geq 2$, $\mathtt{xs}(T) > \Delta - 2$.*

Proof. We prove the following more general result. Let G be any connected graph with a cut-vertex v (so G has at least two edges) and let $cc(v) \geq 2$ be the number of the connected components in $G \setminus \{v\}$, then $\mathtt{xs}(G) > cc(v) - 2$. The result clearly holds if $cc(v) = 2$ since any strategy must use at least one searcher to clear any graph with at least one edge. Therefore, we may assume that $cc(v) > 2$.

Let v be a cut-vertex of a graph G and consider any strategy using at most $cc(v) - 2$ searchers. Initially, at least two components U and W of $G \setminus \{v\}$ are unoccupied and so all edges in these components are contaminated. Let us, consider the first step when a searcher occupies a node in one of these components, say U. That is, let us consider the first step when a searcher slides from v to a node in U. But after this step, no searchers are occupying a node in W which is still contaminated, no searcher is occupying v and there is a connected component C of $G \setminus (\{v\} \cup U \cup W)$ that contains no searchers. Hence, C is fully (re)contaminated. Hence, at any step of the strategy, at least two connected components of $G \setminus \{v\}$ remain contaminated.□

We proof on Claim 1 that an exponential increase in the number of searchers used to clear a graph since the mixed-search number of n-node trees is at most $O(\log n)$ [15]. On the positive side, we show that, for any graph G with maximum degree Δ, $\mathtt{xs}(G) \leq (\Delta - 1)(s(G) + 1)$. To prove it, we consider a classical strategy $\mathcal{S}$ for G using $\mathtt{s}(G)$ searchers. To build an exclusive strategy $\mathcal{S}^*$ for G, we mimic $\mathcal{S}$ using a team of $\Delta - 1$ searchers to "simulate" each searcher in $\mathcal{S}$. For details, see the proof below.

Let $N(v)$ denote the set of neighbors of a node $v \in V(G)$ and let $N[v] = N(v) \cup \{v\}$ be the *close neighborhood* of node v.

Theorem 1 *For any connected graph G with maximum degree Δ,*

$$\mathtt{s}(G) \leq \mathtt{xs}(G) \leq (\Delta - 1) \cdot (\mathtt{s}(G) + 1).$$

Moreover, if $\Delta \leq 3$, $\mathtt{s}(G) \leq \mathtt{xs}(G) \leq \mathtt{s}(G) + 1$.

Proof. $\mathtt{s}(G) \leq \mathtt{xs}(G)$ is a direct consequence of the definition since exclusive-search is a variant of mixed-search where searchers are more constrained.

To prove the second inequality, it is more convenient to deal with node-search. A *node-search strategy* is a search strategy where edges become clear only if both their ends are occupied simultaneously. The *node-search number*, denoted by $\mathtt{ns}(G)$, of a graph G is the smallest number of searchers needed to clear G in this setting. It is well known that $\mathtt{ns}(G) \leq \mathtt{s}(G) + 1$ for any graph G and that recontamination does not help for node-search [2, 3]. Hence, we will prove here that $\mathtt{xs}(G) \leq (\Delta - 1) \cdot \mathtt{ns}(G)$ for any graph G with maximum degree Δ.

Let $\mathcal{S}$ be a winning monotone node-search strategy of G using $k \geq \mathtt{ns}(G)$ searchers. Let $(s_1, \cdots, s_r)$ be the corresponding sequence of place and remove steps. Note that we can assume that a searcher is removed as soon as possible, i.e., if after some step s_i, a searcher occupies a node incident only to clear edges, no other searcher can be placed before this searcher is removed. We also may assume that there are no useless steps, i.e., no step consists in placing a searcher at a node incident to only clear edges nor in placing a searcher at an already occupied node. Last remark, before each step s_i that consists in placing a searcher at a node v, all edges incident to v are contaminated.

For any i, $1 \leq i \leq r$, let E_i be the set of clear edges after step s_i and let V_i be the set of occupied nodes after step s_i. By remark above, if s_{i+1} consists in placing a new searcher, then every node of V_i is incident to at least one contaminated edge after step s_i. We set $V_0 = E_0 = \emptyset$. Because the strategy is monotone, $E_{i-1} \subseteq E_i$ for any $1 < i \leq r$. Moreover, any path from an edge in E_i and an edge in $E \setminus E_i$ must contain a node in V_i.

1. Let us first assume that $\Delta \leq 3$. In this case, we show that $\mathtt{xs}(G) \leq \mathtt{ns}(G)$. We use $\mathcal{S}$ to define an exclusive strategy $\mathcal{S}^*$ to clear G using k searchers. Initially, the k searchers are placed at any arbitrary k nodes. Then, the strategy consists of r phases described below, where Phase i of $\mathcal{S}^*$ depends on step s_i of $\mathcal{S}$ ($i \leq r$). For any $1 \leq i \leq r$, we show that, if before Phase i, the nodes of V_{i-1} were occupied and the edges of E_{i-1} were clear, then after Phase i, the nodes of V_i are occupied and the edges of E_i are clear. In particular, this implies that after Phase r, the edges of $E_r = E$ are clear. Note that, before Phase 1,

the induction hypothesis holds since $V_0 = E_0 = \emptyset$. Now, let $1 \leq i \leq r$ and assume that the induction hypothesis holds before Phase i.

The easy case is when s_i is a remove-step. In that case, Phase i consists in doing nothing and the induction still holds before Phase $i+1$.

Let us now assume that step s_i consists in placing a searcher at some node $v \in V$. Therefore, $|V_{i-1}| < k$ and there are searchers that are not occupying a node in V_{i-1} just before Phase i. We call these searchers "*free searchers*". Let us consider the free searcher f in node x that is the closest to v just before Phase i. The Phase i consists in "moving" f from w to v. We show how it can be done without recontaminating any edge in E_{i-1} and then we show that the edges of E_i are clear at the end of the Phase i.

Let $P = (w, w_2, \cdots, u_{n_1}, v)$ be a shortest path from w to v.

(a) First assume that P contains no nodes in V_{i-1}. Then, no nodes in $\{u_2, \cdots, u_{n_1}, v\}$ are occupied because we have considered the free searcher f that is closest to v. Therefore, Phase i consists in sliding f from w to v along P. Let F be the set of edges that are incident to some node of P. To show that no edges in E_{i-1} is recontaminated during these moves, it is sufficient to point out that either $F \subseteq E_{i-1}$ or $F \subseteq E \setminus E_{i-1}$. Indeed, we already mentioned that any path between an edge in E_{i-1} and an edge in $E \setminus E_{i-1}$ contains a node in V_{i-1}. In other words, an edge of E_{i-1} might be recontaminated only by moving a searcher from a node in V_{i-1}.

(b) Otherwise, there exist $2 \leq j \leq \ell < h$ such that, for any $q < j$, $u_q \notin V_{i-1}$; for any $j \leq q \leq \ell$, $u_q \in V_{i-1}$ and $u_{\ell+1} \notin V_{i-1}$. In that case, f first slides from w to u_{j-1}. No edges of E_{i-1} are recontaminated for the same reason as in the previous item. Then, we simulate the "jump" of the free searcher from u_{j-1} to $u_{\ell+1}$ in the following way. For t from ℓ down to j, the searcher at u_t slides to u_{t+1}. Finally, the free searcher occupying u_{j-1} slides to u_j.

When a searcher moves from u_t to u_{t+1}, the only two edges that might be recontaminated are $\{u_{t-1}, u_t\}$ and $\{u_{t+1}, u_t\}$ because u_t has degree at most three. However, $\{u_{t+1}, u_t\}$ is clear as soon as the next searcher moves from u_{t-1} to u_t.

Therefore, after these $\ell - j + 2$ moves, the edges of E_{i-1} are still clear, the nodes of V_{i-1} are still occupied and a (new) free searcher is occupying the node $u_{\ell+1}$, i.e., the free searcher has progressed along P.

The same process is done until a searcher reaches v (always following P).

To conclude, after Phase i, a searcher is on v and all nodes of V_{i-1} are occupied, hence the nodes of $V_i = V_{i-1} \cup \{v\}$ are occupied. Moreover, we showed above that the edges of E_{i-1} are still clear after Phase i. Since $E_i \setminus E_{i-1}$ is the set of edges that are incident to v and to some node in V_{i-1}, they are clear as well at the end of this phase.

Hence, the induction hypothesis holds after Phase i and $\mathcal{S}^*$ is a winning strategy satisfying exclusivity for clearing G with k searchers. Hence, $\texttt{xs}(G) \leq \texttt{ns}(G)$.

2. Now, let us consider the case of an arbitrary Δ. We use $\mathcal{S}$ to define an exclusive strategy $\mathcal{S}^*$ to clear G using $(\Delta - 1)k$ searchers. As previously, $\mathcal{S}^*$ consists of r phases such that, after Phase i, $i \leq r$, we ensure that the edges in E_i are clear. The difference with previous case is that at each step, we ensure that all nodes of V_i are occupied after Phase i and, moreover, for any node $v \in V_i$ at most two neighbors of v are not occupied.

As previously, all the searchers are placed on arbitrarily different nodes initially. Then, for each step s_i $(1 \leq i \leq r)$, if s_i is a remove-step then Phase i consists in doing nothing.

Otherwise, if s_i consists in placing a searcher at some node v, this means that $|V_{i-1}| < k$ and therefore there are at least $\Delta - 1$ free searchers. Here, each node in V_{i-1} is *assigned* $\leq \Delta - 1$ searchers, the remaining searchers being the free searchers. Intuitively, a searcher assigned to a node v is dedicated to protect v from recontamination, either by occupying v or by occupying some neighbor of v. To ensure this, Phase i consists in moving $\Delta - 1$ free searchers from their current positions either to v or to some neighbor of v until v is occupied and at most two neighbors of v are not occupied.

More precisely, while v has not been yet assigned $\Delta - 1$ searchers and there is still a node in $N[v]$ that is unoccupied, we do the following. Let us consider a free searcher f not occupying a node in $N[v]$ and that is closest to an unoccupied node x in $N[v]$. f is assigned to v in the following manner. Let w be the node occupied by f at the end of Phase $i-1$, and let $P = (w = u_1, \cdots, u_h = x)$ be a shortest path from w to x. As previously, if no nodes in $\{u_2, \cdots, u_h\}$ are occupied, f simply slides along the path until it reaches x.

Otherwise, w.l.o.g., assume that $\{u_2, \cdots, u_j\}$ are occupied and u_{j+1} is not for some $2 \leq j < h$. Then, the strategy is the following. For t from j down to 1, if $u_t \notin V_{i-1}$, the searcher at u_t slides to u_{t+1}. Otherwise, if $u_t \in V_{i-1}$ there are two cases. If there are two neighbors $z \neq u_{t+1}$ and $y \neq u_{t+1}$ of u_t that are not occupied, then the searcher at u_t slides to y. Note that such a case may occur only if $u_{t+1} \notin V_{i-1}$. Otherwise, the searcher at u_t slides to u_{t+1}.

After Phase i, all edges of E_i are clear and all nodes of V_i and their neighbors (but at most two per node in V_i) are occupied. Hence, the theorem follows.

$\square$

Because the pathwidth of a graph G and its search number $s(G)$ differ by at most one, Theorem 1 also proves that the ratio between the exclusive search number and the pathwidth of G is bounded by the degree of G.

Note that the example of the n-node star (S_n) shows that the inequalities in Theorem 1 are tight up to a constant ratio, since $\mathtt{s}(S_n) = 2$ and $\mathtt{xs}(S_n) = n - 2$ for any $n \geq 4$.

We now focus on monotonicity property. Indeed, another important difference of exclusive search compared to classical graph searching is that it is not monotone. As explained in the example of a star (at the beginning of the section), when a searcher needs to cross another one, letting the former searcher pass (to satisfy exclusivity) may lead to a recontamination of some edges. The goal of the winning strategy is to prevent a possible "uncontrolled" recontamination.

Claim 2 *Exclusive graph searching is not monotone, even in trees.*

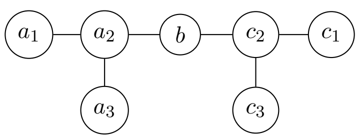


Figure 1: A tree where no optimal exclusive search strategy is monotone.

Proof. Let T be a tree with a node set $\{a_1, a_2, a_3, b, c_1, c_2, c_3\}$ such that (a_1, a_2, b, c_2, c_1) is a path and a_3 is adjacent to a_2 and c_3 is adjacent to c_2 see Figure 1.

We first prove that $\mathtt{xs}(T) = 2$. A winning strategy using two searchers is defined as follows: choose $\{a_1, a_3\}$ as initial positions and place searchers X and Y on these nodes respectively. Then, Y slides along the edges of the path from a_3 to c_2, X slides along the edges from a_1 to b, Y goes to c_3 (here the edge $\{b, c_2\}$ is recontaminated), and finally, X slides to c_1. The fact that $\mathtt{xs}(T) > 1$ is obvious and follows Claim 1.

Finally, consider any winning strategy using two searchers, we show that there is a step with recontamination. There are two cases to be considered. Either the set of initial positions contains no nodes in $C = \{c_1, c_2, c_3\}$ (or symmetrically in $A = \{a_1, a_2, a_3\}$), or one searcher initially occupies a node in C and the other searcher occupies a node in A.

In the first case, i.e., no nodes in C are occupied initially, consider the first step when c_1 or c_3, w.l.o.g., say c_1, is occupied. This can be done only by moving a searcher along the edge $\{c_2, c_1\}$. But then, the edge $\{b, c_2\}$ is recontaminated by $\{c_2, c_3\}$ (since c_3 has never been occupied yet).

In the second case, consider the first searcher to reach b, w.l.o.g., it comes from a_2. Then, it is easy to verify that $\{a_2, b\}$ is recontaminated because a single searcher cannot have cleared $\{a_1, a_2\}$ and $\{a_3, a_2\}$ simultaneously. □

Last, but not least, contrary to classical graph searching, exclusive graph searching is not closed under minor. We show below that though taking a subgraph can decrease the connectivity and "help" the searchers in the classical case, it does not do so in general in the exclusive search. Formally, we show the following.

Claim 3 *For any $\Delta \geq 3$, there exists a graph G with $2\Delta + 1$ nodes and an induced subgraph H such that $\mathtt{xs}(H) = \Delta - 1$ and $\mathtt{xs}(G) \leq 3$.*

Proof. Let H be a $(\Delta + 1)$-node star. By Claim 1, $\mathtt{xs}(H) \geq \Delta - 1$ and it is easy to check that $\mathtt{xs}(H) = \Delta - 1$. Let G be a graph built from a cycle with a set of nodes $(a_1, \cdots, a_{2\Delta})$ (in this order), and by adding a node c adjacent to a_{2i}, for any $1 \leq i \leq \Delta$. H is an induced subgraph of G. Finally, $\mathtt{xs}(G) \leq 3$: place searchers at each node in $\{c, a_1, a_2\}$. The searcher at a_2 then slides to $a_{2\Delta}$ along the path induced by $V(G) \setminus \{a_1, c\}$. □

Actually, there is an even more surprising behaviour of exclusive graph searching. Consider any (mixed) search strategy $\mathcal{S}$ that clears a graph G, let H be any subgraph and let k be the maximum number of searchers that occupy the nodes of H at some step of $\mathcal{S}$. Then, $\mathtt{s}(H) \leq k$ [folklore]. In other words, since $\mathtt{s}(H)$ searchers are required to clear H, there must be a step of $\mathcal{S}$ where at least $\mathtt{s}(H)$ searchers are occupying simultaneously some nodes of H. Note that this property is one of the key points of the proof of the Parsons' characterization [15].

Unfortunately, this property is not true anymore for exclusive graph searching even restricted to trees. Indeed, let $\Delta > 3$ and let T be the tree obtained from a star S with leaves $(a_1, \cdots, a_\Delta)$ by adding, for any $3 \leq i \leq \Delta$, two neighbors b_i and c_i to a_i. Note that $T \setminus S$ induces a graph that is not connected. Consider the following exclusive search strategy: start with one searcher at any node b_i, $3 \leq i \leq \Delta$ and one searcher at a_1; the searcher at a_1 slides to the center of S; then, sequentially for i from 3 to Δ, the searcher at b_i goes to a_i and then to c_i; finally, the searcher at the center goes to a_2. At any step of the described strategy, at most 2 searchers are occupying the nodes in S, while $\mathtt{xs}(S) = \Delta - 1$.

Nevertheless, we prove in Lemma 1 that this property remains true for some particular subtrees of trees (roughly, when $T \setminus S$ is connected). As a corollary, it follows that exclusive-search is closed under subgraph in trees.

3 Exclusive search number is closed under taking subtrees

This section is devoted to proving that the exclusive search number of trees is closed under taking subtrees. Theorem 2 will be used to prove the necessary conditions for our main result (Theorem 3 in the next section).

Theorem 2 *For any tree T and any subtree R of T, $\mathtt{xs}(R) \leq \mathtt{xs}(T)$.*

Contrary to classical graph searching, the proof of Theorem 2 is not trivial, due to of the exclusivity property. To prove it, we will transform an exclusive strategy $\mathcal{S}$ for a tree T into a strategy $\mathcal{S}'$ for a subtree R using the same number of searchers, and without violating the exclusivity property. Before going into the details of the proof and to point out the main difficulties, let us briefly recall the proof of the fact that the mixed-search number is closed under taking subgraphs. Let $\mathcal{S}_m = (s_1, \cdots, s_r)$ be a mixed-search strategy that clears a graph G using $k \geq \mathtt{s}(G)$ searchers and let H be any subgraph of G. Then, a mixed-search strategy clearing H can be obtained as follows. For any $i \leq r$ such that s_i is a step that does not concerns the nodes of H[2], remove s_i from $\mathcal{S}_m$. For any $i \leq r$, if s_i consists in sliding a searcher along $\{u, v\} \in E(G)$ (from u to v) where $u \in V(G) \setminus V(H)$ and $v \in V(H)$, then replace s_i by the step placing a searcher at v. Finally, for any $i \leq r$, if s_i consists in sliding a searcher along $\{u, v\} \in E(G)$ (from u to v) where $u \in V(H)$ and $v \in V(G) \setminus V(H)$, then replace s_i by the step removing a searcher from u. It is easy to check that the obtained sequence of moves is a mixed-search strategy that clears H using at most k searchers.

A key point when designing the mixed-search strategy for H is that some searchers may be removed from the graph when they are useless and placed at some node later when they become useful (for instance when a step of $\mathcal{S}_m$ slides a searcher from a node in $V(G) \setminus V(H)$ to a node of H). Note that when a searcher is removed from H, it does not interfere with the moves of other searchers. Such removals are not allowed in exclusive search strategy. On the contrary, in an exclusive search strategy, any searcher must occupy some node of H since the beginning of the strategy and without disturbing the motions of other searchers. Moreover, notice that, a searcher may occupy some node that must be crossed by some other searcher (according to mixed-search strategy for H). Then, in the exclusive strategy, the moves should be adapted to avoid violation of the exclusivity constraint.

The next two Propositions are technical results that will be used to deal with these problems in trees and finally to prove Lemma 1. Informally, these propositions state that, if there exists a sequence of moves in a tree, then we can initially add a searcher without disturbing the whole motion of the searchers. That is, we are able to clear the same part of the tree even with the extra searcher "in the middle". Moreover, in some cases, we can keep some control on the final position of the extra searcher with respect to its initial position.

Notations. Let $G = (V, E)$ be a graph and $\mathcal{S} = (s_1, \cdots, s_r)$ be a sequence of moves. A *state* after step s_i (or before step s_{i+1}) consists of a pair (I, F) where I is the set of nodes occupied by some searchers after step s_i and F is a subset of cleared edges (after s_i). Moreover, we impose that, for any two incident edges $e \in F$ and $f \notin F$, their common end is in I[3]. Note that F does not necessarily contain all clear edges. We say that $\mathcal{S}$ is a *sequence from state $\mathcal{C}$ to state $\mathcal{C}'$* if $\mathcal{C}$ is the state before the first step of $\mathcal{S}$ and $\mathcal{C}'$ is the state after its last step.

We recall that for any $v \in V(G)$, $N(v)$ denotes the set of the neighbors of v and $N[v] = N(v) \cup \{v\}$.

[2] That is, s_i consists in either placing a searcher at a node of $V(G) \setminus V(H)$, or removing a searcher from a node of $V(G) \setminus V(H)$, or sliding a searcher along an edge in $\{u, v\} \in E(G)$ where $u, v \in V(G) \setminus V(H)$

[3] This constraint is only to avoid considering pathological cases when a sequence $\mathcal{S}$ starts from a state (I, F) and some edges of F are recontaminated before the first move of $\mathcal{S}$.

Proposition 1 *Let T be a tree, $\mathcal{S} = (s_1, \cdots, s_r)$ be a sequence of exclusive-search moves from state $\mathcal{C} = (I, F)$ to state $\mathcal{C}' = (I', F')$ and let $v \in V(T) \setminus I$. There exists a sequence $\mathcal{S}'$ of moves in T from $(I \cup \{v\}, F)$ to $(I' \cup \{w\}, F')$, where $w \in V(T) \setminus I'$. Moreover, if $v \notin I'$ then $w = v$.*

Proof. If $|V(T)| = 1$, the claim clearly holds. Indeed, since $v \in V(T) \setminus I$, then $I = \emptyset$. In that case, $\mathcal{S}'$ consists only in placing a searcher at v. Assume by induction on $n > 1$ that the claim holds for any tree T with $|V(T)| < n$. We now show that it holds if $|V(T)| = n$ by induction on $r > 0$.

If the first step s_1 of $\mathcal{S}$ consists of sliding a searcher from $x \in I$ to $y \neq v$, let $\bar{F}$ be the set of clear edges after this step in $\mathcal{S}$.

- If $r = 1$, then $\mathcal{S}$ goes from state (I, F) to state $(I \cup \{y\} \setminus \{x\}, F')$ (where $\bar{F} = F'$ and $I' = I \cup \{y\} \setminus \{x\}$). Then $\mathcal{S}'$ only executes the step s_1 and the result clearly holds.

  More precisely, $\mathcal{S}'$ starts from $(I \cup \{v\}, F)$ (possible since $v \notin I$) and then moves the searcher at x along the edge $\{x, y\}$ (possible since $v \neq y$). The set of edges that are cleared at the end of such strategy must be a super-set of (or equal to) the subset $\bar{F}$ of edges cleared at the end of $\mathcal{S}$ (in $\mathcal{S}'$, there is one searcher more, so no edge can be recontaminated by $\mathcal{S}'$ while it is not recontaminated by $\mathcal{S}$). Hence, $\bar{F} = F'$ is a subset of cleared edges after $\mathcal{S}'$. Therefore, the state $(I' \cup \{v\}, F')$ is achieved by $\mathcal{S}'$.

  In other words, $\mathcal{S}'$ goes from $(I \cup \{v\}, F)$ to $(I' \cup \{v\}, F')$.

- Otherwise, the sequence $(s_2, \cdots, s_r)$ goes from $((I \cup \{y\}) \setminus x, \bar{F})$ to (I', F'). By induction, there is a sequence $\mathcal{S}^*$ of moves from $((I \cup \{y, v\}) \setminus x, \bar{F})$ to $(I' \cup \{w\}, F')$ where $w \notin I'$ and $w = v$ if $v \notin I'$. The sequence $(s_1, \mathcal{S}^*)$ starting from $(I \cup \{v\}, F)$ satisfies the requirements.

Otherwise, let us assume that the first step s_1 of $\mathcal{S}$ consists in sliding a searcher from $x \in I \cap N(v)$ to v.

If $r = 1$, then the sequence that starts from $(I \cup \{v\}, F)$ and does nothing satisfies the requirements.

Let us assume that $r > 1$. There are two cases: either v remains occupied during the whole strategy $\mathcal{S}$, or let s_i, $1 < i \leq r$, be the first step such that the searcher at v slides to some node $y \in N(v)$. Let $T_1, \cdots, T_p$ be the connected components of $T \setminus \{v\}$ where T_p is the component of x.

1. Consider the first case, i.e., from s_1, v is always occupied in $\mathcal{S}$. We may assume (up to reordering the steps in $\mathcal{S}$) that $\mathcal{S}$ applies move s_1, and then applies the moves in $T_1, \cdots, T_p$ sequentially, i.e., any move in T_i is performed before any move in T_{i+1} for any $1 \leq i < p$.

   Let $\mathcal{S}^*$ be the restriction of $(s_2, \cdots, s_r)$ to T_p (that is the subsequence of the moves that consists in sliding along an edge of T_p). The sequence $\mathcal{S}^*$ starts from $((I \cap V(T_p)) \setminus \{x\}, \bar{F})$, where $\bar{F}$ is the set of edges of $E(T_p)$ that are cleared after step s_1 of $\mathcal{S}$. Moreover, $\mathcal{S}^*$ terminates in $(I' \cap V(T_p), F' \cap E(T_p))$.

   Since $|V(T_p)| < |V(T)|$, using induction, there is a sequence of moves $\mathcal{S}^p$ in T_p that starts from $((I \cap V(T_p)) \cup \{x\}, \bar{F})$ and terminates in $(I' \cap V(T_p) \cup \{w\}, F' \cap E(T_p))$ where $w \notin I' \cap V(T_p)$ and $w = x$ if $x \notin I' \cap V(T_p)$.

   Then, the following sequence of moves proves the claim. Starting from $(I \cup \{v\}, F)$, apply sequentially the moves of $\mathcal{S}$ in $T_1, \cdots, T_{p-1}$ and finally apply the sequence of moves $\mathcal{S}^p$.

2. Now, let s_i, $1 < i \leq r$, be the first step of $\mathcal{S}$ such that the searcher at v slides to some node $y \in N(v)$. Let (I^i, F^i) be the state reached by $\mathcal{S}$ after this step. Note that $v \notin I^i$.

Therefore, the sequence of moves $(s_{i+1}, \cdots, s_r)$ allows to go from state (I^i, F^i) to state (I', F'). By induction on r, there is sequence of moves $\mathcal{S}^{**}$ that allows to go from $(I^i \cup \{v\}, F^i)$ to $(I' \cup \{w\}, F')$, where $w \notin I'$ and $w = v$ if $v \notin I'$.

It only remains to prove that there is a sequence $\mathcal{S}^*$ of moves that starts in $(I \cup \{v\}, F)$ and terminates in $(I^i \cup \{v\}, F^i)$. The sequence of moves that consists of applying the moves in $\mathcal{S}^*$ and then the moves in $\mathcal{S}^{**}$ will then satisfy the requirements.

Let $\mathcal{S}^{i-1}$ be the sequence of moves $(s_2, \cdots, s_{i-1})$ and let (I^{i-1}, F^{i-1}) be the state reached by $\mathcal{S}^{i-1}$. We may assume (up to reordering the steps in $\mathcal{S}^{i-1}$) that $\mathcal{S}^{i-1}$ applies move s_1, and then applies the moves in $T_1, \cdots, T_p$ sequentially.

Let us define $\mathcal{S}^*$ as follows. First, apply sequentially the moves of $\mathcal{S}^{i-1}$ in $T_1, \cdots, T_{p-1}$. Then,

(a) if $y \neq x$, move the searcher at v to y and the searcher at x to v and then apply the moves of $\mathcal{S}^{i-1}$ in T_p.

(b) Otherwise, as in previous item, by induction since $|V(T_p)| < |V(T)|$, from the sequence of moves of $\mathcal{S}^{i-1}$ restricted to T_p, we can define a sequence $\hat{\mathcal{S}}$ of moves in T_p that starts in the same state as in $\mathcal{S}^{i-1}$ (restricted to T_p) plus one extra searcher at x and that reach the same final state as in $\mathcal{S}^{i-1}$ (restricted to T_p) with the extra searcher still at x. Apply $\hat{\mathcal{S}}$ to terminate $\mathcal{S}^*$.

□

Proposition 2 *Let T be a tree and let $\mathcal{S} = (s_1, \cdots, s_r)$ be a sequence of moves from $\mathcal{C} = (I, F)$ to $\mathcal{C}' = (I', F')$. Assume that there is $v \in I$ such that v is occupied only during the first $r-1$ steps of $\mathcal{S}$ and s_r consists in sliding the searcher at v to one of its neighbors. Note that $v \notin I'$. There exists $w \in V(T) \setminus I$ and a sequence $\mathcal{S}'$ of moves in T starting from $(I \cup \{w\}, F)$ and ending in state $(I' \cup \{v\}, F')$.*

Proof. We build a strategy $\mathcal{S}' = (s'_1, \cdots, s'_p)$ with an extra searcher f(compared with $\mathcal{S}$) that starts in some node $w \notin I$ and ends in v such that, at the end of $\mathcal{S}'$, all edges in F' are cleared and the set of occupied nodes is $I' \cup \{v\}$.

We need to find an unoccupied initial position w for t f, such that it ensures that v will be occupied at the end of the sequence of moves. Let $\{v = w_\ell, \cdots, w_1 = w\}$, $\ell > 1$, be the path defined as follows. $w \in V(T) \setminus I$, i.e., w_1 is not initially occupied in $\mathcal{S}$. For any $1 < j \leq \ell$, w_j is initially (in $\mathcal{S}$) occupied by a searcher, i.e., $w_j \in V(T) \cap I$, and the first move of this searcher in $\mathcal{S}$, at step s^*_j, is to slide from w_j to w_{j-1}. Such a path clearly exists and $s^*_j > s^*_{j-1}$ (meaning step s^*_j occurs before step s^*_{j+1} in $\mathcal{S}$) for any $1 \leq j < \ell$. In particular, note that $s^*_\ell = s_r$.

Strategy $\mathcal{S}'$ initially places f at $w_1 = w$ and other searchers are placed at the nodes of I.

Let (I_2, F_2) be the state reached by $\mathcal{S}$ just before step s^*_2 (when the searcher at w_2 moves for the first time and goes to $w_1 = w$). Note that $w \notin I_2 \cup I$. Therefore, by Proposition 1, there is sequence $\mathcal{S}_2$ of moves that goes from state $(I \cup \{w\}, F)$ to $(I_2 \cup \{w\}, F_2)$.

Therefore, starting with f in w and executing $\mathcal{S}_2$, we obtain the same state as in $\mathcal{S}$ before s^*_2, but with f still in w (possibly more edges may have been cleared). Then s^*_2 moves a searcher from $w_2 \in I_2$ to w_1. Actually, since both w_2 and w_1 are occupied after executing $\mathcal{S}_2$, we have reached the state after step s^*_2 in $\mathcal{S}$, having f in w_2.

Assume that we have executing the sequence of moves $(\mathcal{S}_2, \cdots, \mathcal{S}_i)$, $i < \ell$, and that we have reached the state $(I_i \cup \{w_i\}, F_i)$ where (I_i, F_i) is the state reached by $\mathcal{S}$ after step s^*_i (in particular, $w_i \notin I_i$). Let (I_{i+1}, F_{i+1}) be the state reached by $\mathcal{S}$ just before step s^*_{i+1}. Note that $w_{i+1} \notin I_{i+1}$. By Proposition 1, there is sequence $\mathcal{S}_{i+1}$ of moves that goes from state $(I_i \cup \{w_i\}, F_i)$ to $(I_{i+1} \cup \{w_i\}, F_{i+1})$. Then s^*_{i+1} moves a searcher from $w_{i+1} \in I_{i+1}$ to w_i. Actually, since both w_{i+1} and w_i are occupied after executing $\mathcal{S}_{i+1}$, we have reached the state after step s^*_{i+1} in $\mathcal{S}$, having f in w_{i+1}.

Going on in that way, the sequence of moves $(\mathcal{S}_2, \cdots, \mathcal{S}_\ell)$ satisfies the desired requirements. □

Definition 1 *Given a node v in a tree T, a connected component of $T \setminus \{v\}$ is called a* branch *at v.*

We are now ready to prove the main lemma.

Lemma 1 *Let T be any tree and R be a branch at $v \in V(T)$. Let $\mathcal{S}' = (s'_1, \cdots, s'_r)$ be any exclusive strategy for T and let k be the maximum number of searchers occupying simultaneously (at the same step) some nodes of R. Then $\mathtt{xs}(R) \leq k$.*

Proof. Let u be the node of R that is adjacent to v and let $e = \{u, v\} \in E(T)$.

Note that, for all $i \leq r$, s'_i consists of the sliding of a searcher along an edge of $E(T)$. We consider the restriction of $\mathcal{S}'$ to R and we build the sequence $\mathcal{S}$ of moves as follows. Let I be the set of nodes of R that are initially occupied by a searcher in $\mathcal{S}'$. In $\mathcal{S}$, let us start by placing one searcher at each node of I. Then, for any step s'_i of $\mathcal{S}'$, if s'_i consists of sliding a searcher along an edge $\{x, y\} \in E(R)$, we keep this move; if s'_i consists of sliding a searcher along an edge $\{x, y\} \in E(T) \setminus (E(R) \cup \{e\})$ (i.e., $v \notin \{x, y\}$), we remove s'_i; if s'_i consists in sliding a searcher from v to u, we add a *particular move* that consists in placing a searcher at u; and if s'_i consists in sliding a searcher from u to v, we add a *particular move* that consists in removing a searcher at u.

Therefore, $\mathcal{S} = (s_1, \cdots, s_p)$ is a sequence of moves that are either sliding a searcher along an edge of R, or a particular step which is either placing a searcher at u or removing a searcher at u. It is easy to check that this sequence results in the clearing of all edges in R, no more than k nodes are occupied at any step, and any node is never occupied by more than one searcher.

We prove by induction on the number of particular steps of $\mathcal{S}$ (removing or placing a searcher) that $\mathtt{xs}(R) \leq k$. First assume that $\mathcal{S}$ contains no particular steps. Clearly, it is an exclusive strategy, using k searchers and clearing all edges of R. Therefore, the result holds.

If $\mathcal{S}$ contains some particular steps, we build a new strategy $\mathcal{S}^*$ with same properties and one less particular step. The result then follows by induction. There are three cases to be considered.

1. Assume that the last particular step s_i $(i \leq p)$ of $\mathcal{S}$ is a removal step. Let (I, F) be the state after this step and let (I', F') be the state at the end of $\mathcal{S}$. Note that $u \notin I$ and $|I| < k$.

   By Proposition 1, there is a sequence $\mathcal{S}^*$of moves from $(I \cup \{u\}, F)$ to $(I' \cup \{w\}, F')$ with $w \in V(R) \setminus I'$. Hence, the sequence of moves $(s_1, \cdots, s_{i-1}, \mathcal{S}^*)$ clears all edges in R and no more than k nodes are occupied at some step. Moreover, it has less particular moves than $\mathcal{S}$, therefore the result holds by induction.

2. Assume that the first particular step s_i $(i \leq p)$ of $\mathcal{S}$ is a placing step. Let (I, F) be the state at the beginning of $\mathcal{S}$ and let (I', F') be the state before this step s_i. Note that $u \notin I'$, that $|I| < k$ and that $(I' \cup \{u\}, F')$ is the state just after s_i.

   By Proposition 2 and Proposition 1, there is a sequence $\mathcal{S}^*$of moves from $(I \cup \{w\}, F)$ to $(I' \cup \{u\}, F')$ with $w \in V(R) \setminus I$. Hence, the sequence of moves $(\mathcal{S}^*, s_{i+1}, \cdots, s_p)$ clears all edges in R and no more than k nodes are occupied at some step. Moreover, it has less particular moves than $\mathcal{S}$, therefore the result holds by induction.

3. If none of the previous cases hold, then there are $1 \leq i < j \leq p$ such that s_i is a removal step, s_j is a placing step and s_ℓ is not a particular step for all $i < \ell < j$. Let (I, F) be the state after step s_i and let (I', F') be the state before step s_j. Note that $u \notin I \cup I'$, that $|I| = |I'| < k$ and that $(I' \cup \{u\}, F')$ is the state just after s_j.

   From Proposition 1, there is a sequence $\mathcal{S}^*$ of moves from $(I \cup \{u\}, F)$ to $(I' \cup \{u\}, F')$. Hence, the sequence of moves $(s_1, \cdots, s_{i-1}, \mathcal{S}^*, s_{j+1}, \cdots, s_p)$ clears all edges in R and no more than k nodes are occupied at some step. Moreover, it has less particular moves than $\mathcal{S}$, therefore the result holds by induction.

□

Note that, in particular, previous lemma implies that $\mathtt{xs}(R) \leq \mathtt{xs}(T)$ for any branch R of any tree T.

Proof of Theorem 2: We only need to prove the result when $V(T) \setminus V(R)$ is a leaf of T and then the result follows by induction on $|V(T) \setminus V(R)|$. Hence, let v be the leaf of T such that $R = T \setminus \{v\}$. Then, R is a branch at v of T and by Lemma 1, it follows that $\mathtt{xs}(R) \leq \mathtt{xs}(T)$.

4 Exclusive Search in Trees

This section is devoted to our main result. We present a polynomial-time algorithm which, given any tree T, computes the exclusive search number $\mathtt{xs}(T)$ of T and an exclusive search strategy enabling $\mathtt{xs}(T)$ searchers to clear T. Our algorithm is based on a characterization of the trees with exclusive search number at most k, for any given k. Our characterization establishes a relationship between the exclusive-search number of T and the exclusive-search number of some of the branches adjacent to any node in T.

Theorem 3 *Let $k \geq 1$. For any tree T, $\mathtt{xs}(T) \leq k$ if and only if, for every node v in T, the following three properties hold:*

1. *$\delta(v) \leq k + 1$ where $\delta(v)$ is the degree of v in T.*
2. *for any branch B at v, $\mathtt{xs}(B) \leq k$;*
3. *for any even $i > 1$, at most i branches B at v have $\mathtt{xs}(B) \geq k - i/2 + 1$.*

To prove the theorem, we first prove (Section 4.1) that, for any tree T and $k \geq 1$, $\mathtt{xs}(T) \leq k$, only if the conditions of Theorem 3 are satisfied. Then, we show that any tree satisfying these conditions can be decomposed in a particular way, depending on k (Figure 2). Next, in Section 4.3, we describe an exclusive search strategy using at most k searchers, that clears any tree decomposed in such a way.

From the characterization of Theorem 3, it follows that $\mathtt{xs}(T)$ can be computed by dynamic programming on T in polynomial-time. Moreover, such an algorithm computes the corresponding decomposition (see Section 4.3 and Section 4.4). Hence, the following result holds:

Theorem 4 *There exists a polynomial-time algorithm that computes $\mathtt{xs}(T)$ and a corresponding exclusive search strategy for any tree T.*

Definition 2 *A* configuration *is a set of distinct nodes $C \subseteq V(T)$ that describes the positions of $|C|$ searchers in T.*

4.1 Necessary Conditions for Theorem 3

We first show that the conditions of Theorem 3 are necessary. The fact that the first condition is necessary directly follows from Claim 1. The second condition is necessary by Theorem 2. The fact that the third condition is necessary mainly relies on Lemma 1.

Lemma 2 *Let $k \geq 1$. For any tree T, if there exist $v \in V(T)$ and an even integer $i > 1$ such that there is a set $B = \{T_j : \mathtt{xs}(T_j) \geq k - i/2 + 1\}$ of branches at v and $|B| > i$, then $\mathtt{xs}(T) > k$.*

Proof. Let $\mathcal{S}$ be any exclusive strategy that clears T. By Lemma 1, for any $j \leq |B|$, there is a step of the strategy $\mathcal{S}$ such that at least $k - i/2 + 1$ searchers occupy simultaneously nodes in T_j. Let s_j be the last such step of $\mathcal{S}$ that occurs in T_j. W.l.o.g. assume that $s_{j-1} < s_j$, for any $1 < j \leq |B|$, and we may assume that, before step s_j, T_j is not completely clear (this means that $\mathcal{S}$ uses $k - i/2 + 1$ searchers in T_j only if it is really needed). Then, at step $s_{i/2+1}$, at least $k - i/2 + 1$ searchers are in $T_{i/2+1}$, some nodes have been cleared in T_j for any $j \leq i/2$, and T_j cannot become fully contaminated anymore until the end of the strategy (otherwise there would be another step after s_j where $k - i/2 + 1$ searchers are in T_j).

For the sake of contradiction, let us assume that $\mathcal{S}$ uses at most k searchers. Then, at step $s_{i/2+1}$, at least $k - i/2 + 1$ searchers are in $T_{i/2+1}$ and there are at most $i/2 - 1$ searchers outside $T_{i/2+1}$. That is, at that moment, there is at least one branch $X \in \{T_{i/2+2}, \ldots, T_{|B|}\}$ ($|B| > i$) at v with still contaminated edges, and at least one branch $Y \in \{T_1, \ldots, T_{i/2}\}$ at v with (at least) some clear edges that must not be recontaminated and no searchers occupy nodes in both these branches. If there is no searcher at v, Y is fully recontaminated. Therefore, we obtain a contradiction.

Otherwise, there is a searcher in v. However, since there is at least one non cleared yet branch without any searcher in it, it has to be cleared by moving there at least one searcher. For that, the searcher from v have to move. However, if this searcher moves (no matter where), there will be still at most $i/2 - 1$ searchers outside $T_{i/2+1}$ and hence, at least one cleared and one uncleared branch without any searcher, and no searcher in v. The cleared branch will be fully recontaminated. Therefore, we obtain a contradiction. $\square$

4.2 Avenue, Decomposition and Notations

In this section, we show that any tree satisfying the conditions of Theorem 3 admits a particular shape. Moreover, for the purpose of clarity, we introduce some others notations. Figure 2 depicts a particular structure that we prove to exist for any tree T satisfying the conditions of Theorem 3, for $k \geq 1$. Specifically, following [14], we prove that there is a path $A = (u_1, \cdots, u_p)$ in T called *avenue* (bold line in Figure 2) such that $p \geq 1$ and, for any component T' of $T \setminus A$, there is an exclusive strategy that clears T' using $< k$ searchers, i.e., $\mathtt{xs}(T') < k$. The proof of the existence of such a path A is very similar to the one given in [14] in the case of edge graph searching. We give the proof for our case of exclusive search below.

Claim 1 *Let a tree T satisfy the conditions of Theorem 3 for $k \geq 1$. Then, there is a subpath A in T such that, for any connected component S of $T \setminus A$, $\mathtt{xs}(S) < k$.*

Proof. If T is a path, then let $A = T$. Otherwise, if $\mathtt{xs}(S) < k$ for every branch S at every node of T, then choose any node $u_1 \in V(T)$ and set $V(A) = \{u_1\}$. Otherwise, let $u_1 \in V(T)$ be a node with the maximum number of branches having search number k. Note that, by the condition 3 of Theorem 3 (for $i = 2$), there are at most two such branches. Then, there are two cases to consider:

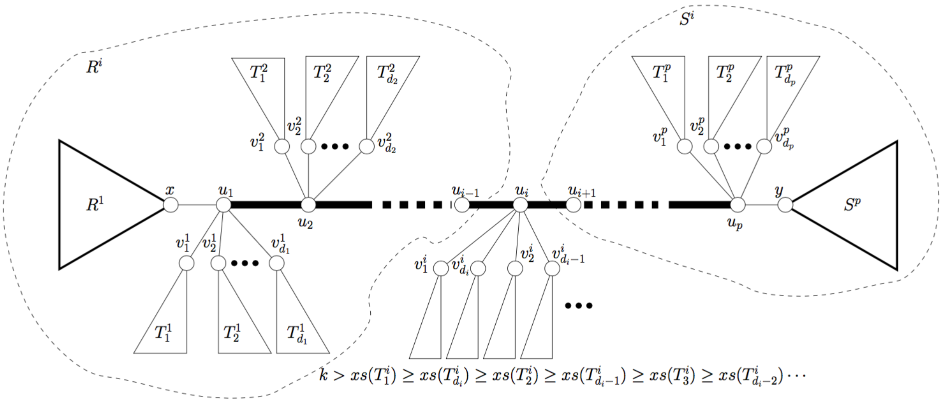


Figure 2: A tree T with avenue $A = (u_1, \cdots, u_p)$. For any subtree X of $T \setminus A$, $\mathtt{xs}(X) < k$.

- First, if there is only one branch S at u_1 with $\mathtt{xs}(S) = k$, let u_2 be the neighbor of u_1 in S, i.e., $\{u_2\} = N(u_1) \cap S$. Then, either all connected components $S \in T \setminus \{u_1, u_2\}$ with $\mathtt{xs}(S) < k$ and then, $V(A) = \{u_1, u_2\}$. Or, there is a branch S' at u_2, $u_1 \notin S'$ with $\mathtt{xs}(S') = k$. In this case, continue this iterative process of decomposition with u_2 and its neighbor $u_3 \in N(u_2) \cap S'$ as before with u_1 and u_2. Proceed till two adjacent nodes u_{p-1} and u_p are found such that all connected components of $T \setminus A$, $A = \{u_1, \dots u_p\}$, are such that each of them, S'', has $\mathtt{xs}(S'') < k$.

- Second, if there are two branches S and S' at u_1 such that $\mathtt{xs}(S) = \mathtt{xs}(S') = k$, let $u_2 \in N(u_1) \cap S$ and $u_2' \in N(u_1) \cap S'$. Then, $\{u_1, u_2, u_2'\} \in A$. If R and R' are two branches at u_2 with $\mathtt{xs}(R) = \mathtt{xs}(R') = k$, then by Lemma 2, one of these branches, say R', has to contain u_1 (otherwise, $\mathtt{xs}(T) > k$). Let $u_3 \in N(u_2) \cap R$. Then, $\{u_1, u_2, u_3, u_2'\} \in A$. Let us continue this process iteratively, till for every connected component $S \in T \setminus A$, $\mathtt{xs}(S) < k$. The process clearly terminates since, at each step, there are at most two components of $T \setminus A$ with $\mathtt{xs} = k$ and their size strictly decreases.

$\square$

Notations. Let a tree T satisfy the conditions of Theorem 3 for $k \geq 1$ and let A be an avenue in T. Hence, each node of the avenue A satisfies the condition 3 of Theorem 3. We introduce some notations depicted in Figure 2. In particular, the ordering of the branches incident to the nodes of the avenue will be crucial in next sections.

Let $A = \{u_1, \cdots, u_p\}$ be the avenue in T. Let $R^1 \subseteq T \setminus A$ be a branch at u_1 maximizing $ws(R^1) < k$ (out of all the branches in $T \setminus A$). Let $S^p \subseteq T \setminus A$ be the branch at u_p (different from R^1 if $p = 1$), and maximizing $ws(S^p)$ (out of the branches at u_p). Let $x \in N(u_1) \cap R^1$ and $y \in N(u_p) \cap S^p$.

For every $1 \leq i \leq p$, let $v_1^i, \cdots, v_{d_i}^i$ be the neighbors of u_i not in $A \cup \{x, y\}$ (by the condition 1 of Theorem 3, $d_i \leq k - 1$ for any $1 \leq i \leq p$). Let T_j^i be the branch at u_i containing v_j^i, for any $1 \leq j \leq d_i$. For any $1 \leq i \leq p$, the nodes $v_1^i, \cdots, v_{d_i}^i$, are ordered such that $\mathtt{xs}(T_1^i) \geq \mathtt{xs}(T_{d_i}^i) \geq \mathtt{xs}(T_2^i) \geq \mathtt{xs}(T_{d_i-1}^i) \geq \mathtt{xs}(T_3^i) \geq \mathtt{xs}(T_{d_i-2}^i) \cdots$. By definition of A, for any $1 \leq i \leq p$ and any $1 \leq j \leq d_i$, $\mathtt{xs}(T_j^i) \leq k - 1$. For $i > 1$, let R^i be the branch at u_i containing u_{i-1} and for $i < p$, let S^i be the branch at u_i containing u_{i+1}. Set $R^{p+1} = T \setminus S^p$.

In the next section, we describe a strategy, called *ExclusiveClear*, based on this decomposition and allowing k searchers to clear T in an exclusive way. The strategy consists in clearing the subtrees of $T \setminus A$, starting with the subtrees that are adjacent to u_1, then the ones adjacent to u_2 and so on, finishing in u_p. To clear a subtree T' of $T \setminus A$, we proceed in a recursive way. That is, we recursively use *ExclusiveClear* on T' using $k' < k$ searchers. The first difficulty is to ensure that no subtrees that have been cleared are recontaminated. For this purpose, when clearing T', the remaining $k - k'$ searchers that are not needed to clear it are used to prevent recontamination. The second difficulty is to ensure exclusivity: while these $k - k'$ searchers are protecting from recontamination, k' searchers should be able to enter T' to clear it.

4.3 Exclusive Search Strategy to Clear Trees

Let $k \geq 1$ and let T be any tree satisfying the conditions of Theorem 3. That is, for any $v \in V(T)$, v has degree at most $k+1$, for any branch B at v, $\mathtt{xs}(B) \leq k$ and, for any even $i > 1$, at most i branches B at v have $\mathtt{xs}(B) \geq k - i/2 + 1$.

By the previous claim, T admits an avenue as described in the previous subsection. We use the same notations as in previous subsection and in Figure 2. In this section, we describe a search strategy that clears T using k searchers.

By definition, our following recursive strategy *ExclusiveClear* ensures that all moves are performed along paths free of searchers, satisfying the exclusivity and internally properties. Moreover, it is easy to check that it actually clears T. To prove its correctness, it is sufficient to show that it uses at most k searchers (in particular, when applying the sub-procedures *bring_searchers* or *transfer* defined below). The formal proof mainly relies on the properties of the decomposition.

A formal description of *ExclusiveClear* is given in Algorithm 1 and the detailed description follows. Before that, let us introduce some new notations. For $u, v \in V(T)$ and $U \subseteq V(T)$, let $u \rightsquigarrow v$ denote the sequence of slidings bringing the searcher at u to v; and let $U \rightsquigarrow v$ denote the sequence of slidings bringing a searcher, occupying some closest to v node of U, to v, along a path free of searchers (in both cases).

Strategy *ExclusiveClear*. For ease of description, let us assume that $|V(R^1)| \geq k-1$. Let I be a subset of u_1 and $k-1$ distinct nodes in R^1. The strategy starts by placing the searchers at the nodes of I[4]. By definition of A, $\mathtt{xs}(R^1) \leq k-1$. Then, the $k-1$ searchers in R^1 apply *ExclusiveClear*(R^1) (such a strategy exists by induction). It is important to mention that the searcher at u_1 preserves R^1 from being recontaminated by the rest of T. After this sequence of moves, all edges in $E(R^1 \cup \{(x, u_1)\})$ are cleared. Finally, a searcher in R^1 that is closest to x goes to x. After this step, R^1 is clear and u_1 and x are occupied. Moreover, all searchers are at nodes in $R^1 \cup \{u_1\}$.

Then, we aim at clearing the remaining subtrees of $T \setminus A$ that are adjacent to u_1, that is, the subtrees $T^1_1, \cdots, T^1_{d_i}$ (see Figure 2). Moreover, after clearing any subtree T^1_i, we need to preserve it from recontamination. Notice that, during the clearing of a subtree T^1_i, u_1 will always be occupied. However, to ensure that exclusivity property is satisfied when searchers go from one subtree T^1_i to the next one T^1_{i+1} (during the *bring_searchers* procedure explained later), we need to free u_1 and then other nodes must be occupied to avoid recontamination.

In order to use as few searchers as possible, the cleaning of the subtrees adjacent to u_1 must be done in a specific order. The order used to clear the subtrees is defined by partitioning these subtrees into two sets S_1 and S_2 built as follows. Each subtree is considered one after the other, in the non-increasing order of $\mathtt{xs}$. In this order, we assign the first subtree to S_1, the second

[4] If $|V(R^1)| < k-1$, then the strategy starts by placing one searcher at u_1, then as much as possible searchers at the nodes of R^1, then at the nodes of T^1_1, of T^1_2, and so on, until the k searchers are placed.

Algorithm 1 *ExclusiveClear* strategy

Require: Tree T satisfying the conditions of Theorem 3 for $k \geq 1$. Notations are recalled in Figure 2.

Initially, the searchers occupy $k-1$ nodes of R^1 and u_1 (if $|V(R^1)| < k-1$, all nodes of R^1 are occupied and then the nodes of T^1_1, T^1_2, etc., by $k-1$ searchers).
ExclusiveClear(R^1)
$R^1 \rightsquigarrow x$
for all $1 \leq i \leq p$ **do**
  for all $1 \leq j_0 \leq d_i + 1$ **do**
    if $i = p \wedge j_0 = d_p + 1$ **then**
      $u_p \rightsquigarrow y$
      $R^{p+1} \rightsquigarrow u_p$
      bring_searchers$(p, d_p + 1)$
      ExclusiveClear(S^p) /* end of the strategy */
    else if $j_0 = d_i + 1 \wedge i < p$ **then**
      $u_i \rightsquigarrow u_{i+1}$
      $R^{i+1} \rightsquigarrow u_i$
    else
      if $j_0 = 1$ **then**
        bring_searchers$(i, 1)$
      else if $1 < j_0 \leq \lceil d_i/2 \rceil$ **then**
        if there is a searcher in R^i **then**
          $R^i \rightsquigarrow u_{i-1}$
        else
          $u_i \rightsquigarrow u_{i-1}$ $(u_0 = x)$
          let $j < j_0$ such that there is a searcher in $T^i_j \setminus v^i_j$.
          $v^i_j \rightsquigarrow u_i$
          $T^i_j \rightsquigarrow v^i_j$
        bring_searchers(i, j_0)
      else if $\lceil d_i/2 \rceil < j_0 < d_i + 1$ **then**
        $u_i \rightsquigarrow u_{i+1}$ $(u_{p+1} = y)$
        $R^i_{j_0} \rightsquigarrow u_i$
        bring_searchers(i, j_0)
      ExclusiveClear$(T^i_{j_0})$
      $T^i_{j_0} \rightsquigarrow v^i_{j_0}$
      if $j_0 = \lceil d_i/2 \rceil$ **then**
        transfer(i)

one to S_2, the third one to S_1, the fourth one to S_2, and we continue to divide the subtrees until each of them is assigned to one of the two sets. Note that the formula given in Figure 2 respects this order. The resulting $S_1 = \{T^1_1, \ldots, T^1_{\lceil d_1/2 \rceil}\}$ and $S_2 = \{T^1_{\lceil d_1/2 \rceil+1}, \ldots, T^1_{d_1}\}$ such that $\mathtt{xs}(T^1_1) \geq \mathtt{xs}(T^1_{d_1}) \geq \mathtt{xs}(T^1_2) \geq \mathtt{xs}(T^1_{d_1-1}) \geq \mathtt{xs}(T^1_3) \geq \mathtt{xs}(T^1_{d_1-2}) \ldots$

The clearing of the subtrees is then divided into three phases. The subtrees in S_1 are cleared first, in the non-increasing order of their $\mathtt{xs}$. Then, the searchers are moved in a particular way (Using Procedure *transfer*). Finally, the subtrees in S_2 are cleared in the non-decreasing order of their $\mathtt{xs}$.

Let us detail each phase.

- Let $1 \leq i < \lceil d_1/2 \rceil$. Let us assume that $R^1, T^1_1, \cdots, T^1_{i-1}$ have been cleared, that $u_1, x, v^1_1, \cdots, v^1_{i-1}$ are occupied by searchers and all other searchers are occupying nodes in $R^1, T^1_1, \cdots, T^1_{i-1}$ and u_1.

  First, Procedure *bring_searchers* is used to move $k' = \mathtt{xs}(T^1_i)$ searchers to nodes of T^1_i without recontaminating the subtrees that have already been cleared. When k' searchers are in T^1_i and $u_1, x, v^1_1, \cdots, v^1_{i-1}$ are occupied, *ExclusiveClear*(T^1_i) is applied recursively to clear T^1_i using the k' searchers. Finally, a searcher in T^1_i reaches v^1_i.

  Below, we detail Procedure *bring_searchers* and we prove that $k' = \mathtt{xs}(T^1_i)$ searchers are actually available to clear T^1_i.

  Each time that a subtree $T^1_j \in S_1$ has been cleared, one searcher is left on its *root* v^1_j (its node adjacent to u_1). That is, once a new subtree is cleared, we somehow loose one searcher to clear the next one. This is balanced by the fact that the number of searchers needed to clear the next subtree does not increase, according to the order of clearing established above, and provided by the properties of T.

- After clearing the subtrees in S_1, there are searchers currently "blocked" in the roots of the cleared subtrees. More precisely, $R^1, T^1_1, \cdots, T^1_{\lceil d_1/2 \rceil}$ have been cleared and $u_1, x, v^1_1, \cdots, v^1_{\lceil d_1/2 \rceil}$ are occupied by searchers.

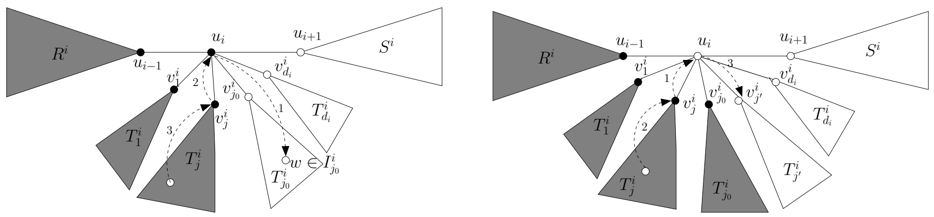
  In order to "re-use" these searchers to clear the remaining subtrees, the strategy changes. Now, the roots of the still contaminated subtrees (set S_2) will be occupied to prevent recontamination of the cleared subtrees (set S_1). Procedure *transfer* (explained later) is used to occupy these nodes, ensuring no recontamination of the subtrees and satisfying the exclusivity property. After *transfer*, the searchers at the roots of the cleared subtrees become *free*, i.e., it is possible now to use them to clear the next subtrees.

  More precisely, after this phase, $R^1, T^1_1, \cdots, T^1_{\lceil d_1/2 \rceil}$ are still clear and $u_1, u_2, v^1_{\lceil d_1/2 \rceil}, \cdots, v^1_{d_1}$ are occupied by searchers.

- Then, the subtrees in S_2 have to be cleared in the non-decreasing order of their $\mathtt{xs}$. Each time that a subtree $T^1_j \in S_2$ has been cleared, the searcher on its root v^1_j becomes free. That is, we somehow gain a searcher to clear the next subtree, whose search number may *increase*, according to the properties of T.

  More precisely, let $\lceil d_1/2 \rceil + 1 \leq i \leq d_1$. Let us assume that $R^1, T^1_1, \cdots, T^1_{i-1}$ have been cleared, that $u_1, u_2, v^1_i, \cdots, v^1_{d_1}$ are occupied by searchers and all other searchers are occupying nodes in $R^1, T^1_1, \cdots, T^1_{i-1}$ and u_1, u_2.

  First, Procedure *bring_searchers* is used to move $k' = \mathtt{xs}(T^1_i)$ searchers to nodes of T^1_i without recontaminating the subtrees that have already been cleared. When k' searchers are in T^1_i and $u_1, u_2, v^1_{i+1}, \cdots, v^1_{d_1}$ are occupied, *ExclusiveClear*(T^1_i) is applied recursively to clear T^1_i using the k' searchers.



(a) Description of one phase of *bring_searchers*(i, j_0).

(b) Description of one phase of *transfer*(i).

Figure 3: Black nodes are occupied. Grey subtrees are cleared. Steps are depicted by dotted arrows.

Once all subtrees of $T \setminus A$ adjacent to u_1 are cleared, the searcher at u_1 goes to u_2 (unless it is already occupied). Now, all the searchers in R^2 (see Figure 2) become free. Then, a similar strategy is applied for the subtrees of $T \setminus A$ adjacent to u_2, and so on, until all the subtrees adjacent to u_p are cleared.

We now describe more precisely two sub-procedures that are used to implement the strategy we have given above.

Procedure *bring_searchers*. It remains to detail how the searchers, once a subtree has been cleared, go to the next subtree, satisfying exclusivity and preventing recontamination. To do so, let $1 \leq i \leq p$ and let us consider the step of the strategy when the branch R^i (see Fig. 2) and all subtrees $T_1^i, \cdots, T_{j_0-1}^i$ ($1 < j_0 \leq d_i$) are cleared (the grey subtrees in Fig. 3(a)).

We describe the procedure in the case when $j_0 \leq \lceil \frac{d_i}{2} \rceil$. The case when $\lceil \frac{d_i}{2} \rceil + 1 \leq j_0 \leq d_i$ is similar.

As explained before, at this step, the nodes in $\{u_i, v_1^i, \cdots, v_{j_0-1}^i\}$ are occupied, and all other searchers are free and occupy nodes of R^i and T_j^i, for $j < j_0$. It is ensured that also u_{i-1} (if $i = 1$, set $u_{i-1} = x$) will be occupied.

Let $k - j_0$ be the number of free searchers, i.e., searchers not occupying $\{u_i, v_1^i, \cdots, v_{j_0-1}^i\}$. We first prove that $k - j_0 \geq \mathtt{xs}(T_{j_0}^i)$. Indeed, by definition of the ordering of $T_1^i, \cdots, T_{d_i}^i$, there are at least $2(j_0 - 1) + 3$ branches at u_i with exclusive search number at least $\mathtt{xs}(T_{j_0}^i)$. Namely, $\mathtt{xs}(R^i) \geq \mathtt{xs}(T_{j_0}^i)$, $\mathtt{xs}(S^i) \geq \mathtt{xs}(T_{j_0}^i)$, and $\mathtt{xs}(T_1^i) \geq \mathtt{xs}(T_{d_i}^i) \geq \mathtt{xs}(T_2^i) \geq \mathtt{xs}(T_{d_i-1}^i) \geq \mathtt{xs}(T_3^i) \geq \mathtt{xs}(T_{d_i-2}^i) \geq \cdots \geq \mathtt{xs}(T_{d_i-j_0+1}^i) \geq \mathtt{xs}(T_{j_0}^i)$. By the condition 3 of Theorem 3, we get that $\mathtt{xs}(T_{j_0}^i) \leq k - j_0$ (since the number of branches at u_i with exclusive search number at least $k - j_0 + 1$ is at most $2j_0$).

The process *bring_searchers*(i, j_0) is applied to bring $\mathtt{xs}(T_{j_0}^i)$ searchers into $T_{j_0}^i$. The searchers are brought one by one, from the clear part to $T_{j_0}^i$, without recontamination (but possibly the edges incident to u_i) and satisfying the exclusivity property.

Figure 3(a) depicts one phase of this process. Before each phase (but the last one, which is slightly different), there is a free searcher at some node b, either in $T_j^i \setminus v_j^i$ (for some $j < j_0$) or in $R^i \setminus u_{i-1}$. First, the searcher at u_i goes to the furthest unoccupied node in $T_{j_0}^i$ (dotted line 1 in Fig. 3(a)). Second, the searcher at v_j^i (or at u_{i-1}) goes to u_i (dotted line 2 in Fig. 3(a)). Finally, the searcher at b goes to v_j^i (or to u_{i-1}) (dotted line 3 in Fig. 3(a)). Clearly, doing so, no recontamination occurs in the cleared subtrees (but in u_i) and exclusivity is satisfied.

Procedure *transfer*. Let $1 \leq i \leq p$ and $j_0 = \lceil d_i/2 \rceil$. Just after clearing $T_{j_0}^i$, we reach a configuration where the nodes in $\{u_i, v_1^i, \cdots, v_{j_0}^i\}$ are occupied, $T_{j_0}^i$ is clear, and all other searchers are at nodes of R^i or T_j^i ($j \leq j_0$). First, the searcher at u_i goes to u_{i-1} unless it is already occupied.

As explained before, the nodes in $\{v^i_{j_0+1}, \cdots, v^i_{d_i}\}$ must now be occupied before clearing any subtree T^i_j, for $j > j_0$. This is the role of sub-process *transfer*(i). The searchers are brought one by one, from the clear part to $\{v^i_{j_0+1}, \cdots, v^i_{d_i}\}$, without recontamination and satisfying exclusivity.

Figure 3(b) depicts one phase of this process. By the condition 1 of Theorem 3, $d_i+2 \leq k+1$. This ensures that, before each phase, there is a free searcher at some node b either in $T^i_j \setminus \{v^i_j\}$ (for some $j \leq j_0$) or in $R^i \setminus \{u_{i-1}\}$. First, the searcher at v^i_j (if $b \in V(T^i_j)$) or at u_{i-1} (otherwise) goes to u_i (unless u_i is occupied) (dotted line 1 in Fig. 3(b)). Second, the searcher at b goes to v^i_j (or u_{i-1}) (dotted line 2 in Fig. 3(b)). Finally, the searcher at u_i goes to an unoccupied node in $\{v^i_{j_0+1}, \cdots, v^i_{d_i}\}$ (dotted line 3 in Fig. 3(b)). Once all these nodes are occupied, the searcher at u_{i-1} goes back to u_i. Clearly, doing so, exclusivity is satisfied and no recontamination occurs in the cleared subtrees. This, in particular, since either all the nodes $\{u_{i-1}, v^i_1, \cdots, v^i_{j_0-1}\}$, or u_i, are always occupied during *transfer*(i).

4.4 Polynomial-time algorithm

From the characterization of Theorem 3, it follows that $\texttt{xs}(T)$ can be computed recursively for any tree T. This section is devoted to the design of such a polynomial-time algorithm. The algorithm to compute $\texttt{xs}$ follows the one designed in [14] to compute the edge-search number of trees. The main difference between our algorithm and the one of [14] comes from the third item of our characterization (Theorem 3) for which we require our dynamic programming algorithm to keep more information (fields “neighbors” and “branches” below).

Avenue and types of rooted trees. First let us refine the definition of *avenue* in order to ensure its unicity. Let T be a tree with $\texttt{xs}(T) = k$. An avenue is a subpath $A = (u_1, \cdots, u_p)$ in T such that $p > 1$, u_1 and u_p have exactly one branch with exclusive search number k (containing v_2 and v_{p-1} respectively) and, for any $2 \leq i < p$, u_i has exactly two branches with exclusive search number k (containing v_{i-1} and v_{i+1} respectively). Note that the definition of a branch in our work (Definition 1) differs a bit from the one in [14]. We adapt the algorithm below according to this definition.

The proof of Claim 1 shows that

Claim 2 *Let T be a tree with $\texttt{xs}(T) = k$. Either T has a vertex v (called hub) such that all branches at v have exclusive search number $< k$, or T has a unique avenue $(u_1, \cdots, u_p)$ and $p > 1$.*

The algorithm takes a tree T rooted in $r \in V(T)$ and recursively computes $\texttt{xs}(T)$. The location of the root relatively to the avenue is important, therefore we need the following notations (borrowed from [14]). By Claim 2, there are 4 possible types:

Type H. All branches at r have exclusive search number $< k$. In that case, r is called the *hub* of T.

Type E. T has a unique avenue $(u_1, \cdots, u_p)$ and r belongs to a branch at u_1 or u_p with exclusive search number $< k$, or $r \in \{u_1, u_p\}$ (in the latter case, we moreover impose that $p > 1$, otherwise this is the case of Type H).

Type I. T has a unique avenue $(u_1, \cdots, u_p)$, $p > 1$ and $r \in \{u_2, \cdots, u_{p-1}\}$.

Type M. T has a unique avenue $(u_1, \cdots, u_p)$ and r belongs to a branch M at u_i $(2 \leq i < p)$ with exclusive search number $< k$.

Recursive algorithm. The algorithm recursively computes the following information record, denoted by

$$info(T,r) = [type, \mathtt{xs}, branches, M\text{-}info, neighbors],$$

associated with tree T (whose root r is chosen arbitrarily).

Note that the only difference with the algorithm designed in [14] comes from the fields *neighbors* and *branches*.

The five fields of $info(T,r)$ are defined as follows

type. It is the type (H, E, I or M) of the tree T rooted at r.

$\mathtt{xs}$. It is the exclusive search number $\mathtt{xs}(T)$ of T.

branches. For any branch B at r, *branches* contains the value of $\mathtt{xs}(B)$.

M-info. If *type* $= M$, let $(u_1, \cdots, u_p)$ be the avenue of T and let B be the branch at u_i that contains r. This field contains the information record $info(B,r)$ associated with the branch B at u_i containing r. Otherwise, it is *nil*.

neighbors. If r has degree one in T, then *neighbors* $=$ *nil*.

Otherwise, *neighbors* is a set of information records, one record for every branch B at r. More precisely, for any branch B at r, *neighbors* contains the record $[type(S), \mathtt{xs}(S), branches(S), M\text{-}info(S), nil]$ associated to the subtree $S = B \cup \{r\}$ rooted at r.

The algorithm (the computation of $info(T,r)$) proceeds as follows. If T is an edge, then $info(T,r) = [H, 1, \{0\}, nil, nil]$. Otherwise,

- If r has degree at least 2, then procedures *merge* (see below) and *re-root* are applied to compute $info(T,r)$. Precisely, let $r_1, \cdots, r_d$ be the neighbors of r. For every $1 \leq i \leq d$, let $T_i = B_i \cup \{r\}$ be the subtree of T that consists of r plus the branch B_i at r in T that contains r_i. The algorithm first recursively computes $info(T_i, r)$ for every $i \leq d$. Then, $info(T,r) = merge(info(T_1,r), \cdots, info(T_d,r))$.
- If r has degree 1, then procedure *re-root* (see below) is applied to compute $info(T,r)$. Precisely, let r_1 be the neighbor of r. First $info(T, r_1)$ is computed and then $info(T,r) = re\text{-}root(info(T_1, r_1))$.

4.4.1 Procedure *merge*

Procedure *merge* is used to compute $info(T,r)$ from the information records of the branches of T. Namely, $info(T,r) = merge(info(T_1,r), \cdots, info(T_d,r))$. The procedure *merge* is defined as follows.

When Procedure *merge* has more than two arguments ($d > 2$), it is defined recursively by $merge(X_1, \cdots, X_d) = merge(X_1, merge(X_2, \cdots, X_d))$. The case $d = 2$ is defined as follows.

Let T be any rooted tree obtained from two rooted trees T_1 and T_2 by identifying their roots into a single vertex r, the root of T. Let $info(T_1,r) = [type_1, \mathtt{xs}_1, branches_1, M\text{-}info_1, neighbors_1]$ and $info(T_2,r) = [type_2, \mathtt{xs}_2, branches_2, M\text{-}info_2, neighbors_2]$. Then, $info(T,r) = merge(info(T_1,r), info(T_2,r))$ is computed as follows.

We may assume that $\mathtt{xs}_1 \geq \mathtt{xs}_2$.

Computation of *neighbors***.** For $i \in \{1,2\}$, if r has degree 1 in T_i, set $neighbors_i^* = \{info(T_i, r)\}$ and set $neighbors_i^* = neighbors_i$ otherwise. Then, $neighbors = neighbors_1^* \cup neighbors_2^*$.

Computation of *branches***.** Let *branches* of $info(T,r)$ be $branches_1 \cup branches_2$.

Computation of xs, *type* **and** *M-info***.** Let ℓ' be the smallest integer such that, for any even $i > 1$, at most i branches B at r in T have $\texttt{xs}(B) \geq \ell' - i/2 + 1$, and $\texttt{xs}(B) \leq \ell'$. Clearly, ℓ' can be computed using *branches*. Let $\ell = \max\{\ell', \texttt{xs}_1, d\}$ where d equals the degree of r in T minus one.

1. If $\ell > \texttt{xs}_1$, then $info(T,r) = [H, \ell, branches, nil, neighbors]$.

   Now, we may assume that $\ell = \texttt{xs}_1$.

2. If $\texttt{xs}_1 = \texttt{xs}_2$, there are the following cases.

   (a) If $type_1 = type_2 = H$, then $info(T,r) = [H, \ell, branches, nil, neighbors]$.

   (b) If $type_1 = H$ and $type_2 = E$ or vice versa, then $info(T,r) = [E, \ell, branches, nil, neighbors]$.

   (c) If $type_1 = type_2 = E$, then $info(T,r) = [I, \ell, branches, nil, neighbors]$.

   (d) If $type_1 = I$ and $type_2 = H$, or vice versa, then $info(T,r) = [I, \ell, branches, nil, neighbors]$.

   (e) Otherwise (at least one of T_1 or T_2 is of type M, or one is of type I and the other of type I or E), then at least 3 branches at r in T have exclusive search number at least $\texttt{xs}_1$. Therefore, in that case, $\ell \geq \ell' > \texttt{xs}_1$. Hence, this case is treated in the previous item.

3. $\texttt{xs}_1 > \texttt{xs}_2$.

   (a) If $type_1 = H, E$ or I, then $info(T,r) = [type_1, \ell, branches, nil, neighbors]$.

   (b) Otherwise, $type_1 = M$. In that case, Procedure *merge* is called on T_2 and M (the branch where r stands in T_1), using $info(T_2, r)$ and $M\text{-}info_1 = info(M, r)$. Let $[type', \texttt{xs}', branches', M\text{-}info', neighbors'] = merge(info(T_2, r), info(M, r))$ be the result of this call.

   - If $\texttt{xs}' < \texttt{xs}_1$, then $info(T,r) = [M, \ell, branches, [type', \texttt{xs}', branches', M\text{-}info', neighbors'], neighbors]$.
   - Otherwise, $info(T,r) = [H, \ell, branches, nil, neighbors]$.

Lemma 3 *The procedure merge computes $info(T,r)$ from $info(T_1,r)$ and $info(T_2,r)$, where T is the n-node tree obtained by identifying the roots of two trees T_1 and T_2. Moreover, its time-complexity is $O(n)$.*

Proof. The proof is by case analysis, corresponding to the cases in the description of the procedure.

1: $\ell > \texttt{xs}_1$. In this case, all branches at r have exclusive search number at most $\texttt{xs}_1 < \ell$. Moreover, ℓ is the smallest integer that satisfies the conditions of Theorem 3. Hence, the result follows.

2a: $\ell = \texttt{xs}_1, \texttt{xs}_1 = \texttt{xs}_2, type_1 = type_2 = H$. In this case, both trees have the root as hub, and their union is a tree in which all branches at the root have search number less than $\texttt{xs}_1 = \texttt{xs}_2$. The new tree has the root as a hub, and its exclusive search number is ℓ, by Theorem 3.

2b: $\ell = \mathtt{xs}_1, \mathtt{xs}_1 = \mathtt{xs}_2, type_1 = H, type_2 = E$. The new tree can be searched with ℓ, by Theorem 3. T has an avenue of length at least one, with r as an endpoint, and hence it is type E.

2c: $\ell = \mathtt{xs}_1, \mathtt{xs}_1 = \mathtt{xs}_2, type_1 = type_2 = E$. The avenue of T is the shortest path that contains both the avenues of T_1 and T_2, and this avenue can be used to search T with ℓ searchers. Since r is one of the interior points on the avenue, T is type I.

2d: $\ell = \mathtt{xs}_1, \mathtt{xs}_1 = \mathtt{xs}_2, type_1 = I, type_2 = H$. The root of T_1 is a node on the interior of its avenue, and after combination with T_2 the off-avenue branches at that root will all continue to have a search number less than $\mathtt{xs}_1 = \mathtt{xs}_2$. Thus, T has the same type as T_1 and $\mathtt{xs} = \ell$.

2e: As already said, this case is treated in Case 1.

3a: $\ell = \mathtt{xs}_1, \mathtt{xs}_1 > \mathtt{xs}_2$, $type_1$ is H, E, or I. If $type_1$ is H, then the root continues to have only branches with search number less than $\mathtt{xs}_1$ in T, so T is of type H and has search number ℓ. If $type_1$ is E, T has an avenue of length at least one, with r on a branch at an endpoint (possibly it is the endpoint), so T is of type E. If $type_1 = I$, the argument of Case 2d applies, so T is of type I and has search number ℓ.

3b: $\ell = \mathtt{xs}_1, \mathtt{xs}_1 > \mathtt{xs}_2$ and $type_1 = M$. The search number for T depends on the search number $\mathtt{xs}(T')$ of the union of the M-tree for T_1 with T_2. If $\mathtt{xs}' = \mathtt{xs}(T') < \mathtt{xs}_1$, then the avenue of T is exactly the same as the avenue of T_1, the same search number $\mathtt{xs}_1 = \ell$ suffices, and T' is now the M-tree of T. If $\mathtt{xs}' \geq \mathtt{xs}_1$, then T is type H with search number ℓ, by Theorem 3.

Every case of the function takes a constant time, but the case 3b which makes the recursive call $merge(info(T_2, r), info(M, r))$ on a tree obtained from the identification of r in T_2 and M which has strictly smaller exclusive search number. The time-complexity $O(\mathtt{xs}(T)) = O(n)$ follows. □

4.4.2 Procedure *re-root*

The procedure *re-root* is defined as follows. Let us assume that r has a unique neighbor r' in T.

Let $info(T, r') = [type', \mathtt{xs}', branches', M\text{-}info', neighbors']$ (that has been computed recursively). Then, $info(T, r) = [type, \mathtt{xs}, branches, M\text{-}info, neighbors]$ is computed as follows.

Computation of xs. Since the tree is not modified (only the root changes), $\mathtt{xs} = \mathtt{xs}'$.

Computation of *branches*. Let $\{r'_1, \cdots, r'_h\}$ be the set of neighbors of r' but r. For any $i \leq h$, let B'_i be the branch at r' containing r'_i. By definition, for any $i \leq h$, $neighbors'$ contains the information record $info(B'_i \cup \{r'\}, r')$ associated to the subtree $B'_i \cup \{r'\}$ rooted in r'. Applying the sub-procedure *merge* allows us to compute

$$info(T \setminus \{r\}, r') = merge(info(B'_1 \cup \{r'\}), \cdots, info(B'_h \cup \{r'\}, r'))$$

that contains $\mathtt{xs}(T \setminus \{r\})$. Note that no recursive call of the main algorithm is done at this step.

The field *branches* of $info(T, r)$ is set to $\{\mathtt{xs}(T \setminus \{r\})\}$.

Computation of *type* and *M-info*.

1. If $type' = E$, then $type = E$ and $M\text{-}info = nil$.
2. If $type' = H$, then $type = E$ and $M\text{-}info = nil$.
3. If $type' = I$ and $\mathtt{xs}' = 1$ (i.e., T is a path with end r) then $type = E$ and $M\text{-}info = nil$. If $type' = I$ and $\mathtt{xs}' > 1$, then $type = M$ and $M\text{-}info = [H, \{0\}, \emptyset, nil, nil]$.
4. If $type' = M$, then $type = M$, $M\text{-}info = re\text{-}root(M\text{-}info')$.

Computation of *neighbors*. Because r has degree 1 in T, $neighbors = nil$.

Lemma 4 *For any nnode tree T that is not just a single edge, the procedure re-root computes $info(T, r)$ from $info(T, r')$, where T is rooted at a leaf r and r' is the neighbor of r. Moreover, its time-complexity is $O(n^2)$.*

Proof. The correctness of *re-root* directly follows from the correctness of *merge* by Lemma 3.

All steps are executed in constant-time but, the application of $merge(info(B'_1 \cup \{r'\}), \cdots, info(B'_h \cup \{r'\}, r'))$ which takes time $O(n)$ by Lemma 3, and the recursive call $re\text{-}root(M\text{-}info')$ to a tree with smaller exclusive search number. Overall, the time-complexity is $O(n^2)$. □

4.4.3 Time-complexity

Theorem 5 *The algorithm described in this section computes $info(T, r)$ for any tree T rooted in r and having n nodes in $O(n^3)$ time.*

Proof. The correctness of the algorithm follows immediately from Lemmas 3 and 4.

The worst case is when r has a unique neighbor r'. Let $B'_1, \cdots, B'_d$ be the branches at r' and let $B_i = B'_i \cup \{r'\}$ for any $1 \leq i \leq d$. We may assume that T is not an edge, so $d \geq 2$ (r' has degree at least 2). Let n_i be the size of B_i and note that $\sum_{1 \leq i \leq d} n_i = n + d - 1$. First $info(B_i, r')$ is computed for any $1 \leq i \leq d$. Then, $merge(B_1, \cdots, B_d)$ is computed, which consists of $O(n)$ merging of two subtrees taking $O(n)$-time each. Finally, the algorithm executes $re\text{-}root(T, r')$ which takes time $O(n^2)$.

Let $g(n)$ be the complexity of the algorithm applied to an n-node tree. By previous paragraph, we get that $g(n) = \sum_{1 \leq i \leq d} g(n_i) + O(n^2)$. Since $\sum_{1 \leq i \leq d} n_i = O(n)$, we get by induction that $g(n) = O(n^3)$. □

5 Conclusion

In this paper, we have defined and study a new graph searching parameter, namely the exclusive search number of graphs. It appears that, contrary to all previous variants of graph searching games, the exclusive search number is not related to pathwidth. An interesting open-problem is to determine if exclusive graph searching is related to other graph parameters related to vertices layouts such as cutwidth or bandwidth.

It has been recently proved that computing the exclusive search number is NP-hard [13]. Is this problem Fixed Parameter Tractable? Do approximation algorithms exist?

References

[1] Lali Barrière, Paola Flocchini, Fedor V. Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, and Dimitrios M. Thilikos. Connected graph searching. *Information and Computation*, 219:1–16, 2012.

[2] Daniel Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discr. Maths and Theoretical Comp. Sc.*, 5:33–49, 1991.

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

[4] Lélia Blin, Janna Burman, and Nicolas Nisse. Exclusive graph searching. In *21st Annual European Symposium on Algorithms (ESA)*, volume 8125 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2013.

[5] Richard L. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, 6:72–78, 1967.

[6] Richard L. Breisch. *Lost in a Cave-applying graph theory to cave exploration.* National Speleological Society, 2012. 298 pages.

[7] Dariusz Dereniowski. From pathwidth to connected pathwidth. *SIAM J. Discrete Math.*, 26(4):1709–1732, 2012.

[8] Jonathan A. Ellis, Ivan Hal Sudborough, and Jonathan S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

[9] Fedor V. Fomin, Pinar Heggernes, and Rodica Mihai. Mixed search number and linear-width of interval and split graphs. *Networks*, 56(3):207–214, 2010.

[10] Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[11] Petr A. Golovach, Pinar Heggernes, and Rodica Mihai. Edge search number of cographs. *Discrete Applied Mathematics*, 160(6):734–743, 2012.

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

[13] Euripides Markou, Nicolas Nisse, and Stéphane Pérennes. Exclusive Graph Searching vs. Pathwidth. Research Report RR-8523, INRIA, 2014. URL `https://hal.inria.fr/hal-00980877`.

[14] Nimrod Megiddo, Seifollah L. Hakimi, Michael R. Garey, David S. Johnson, and Christos H. Papadimitriou. The complexity of searching a graph. *J. Assoc. Comput. Mach.*, 35(1):18–44, 1988.

[15] T. D. Parsons. The search number of a connected graph. In *9th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*, Congress. Numer., XXI, pages 549–554. Utilitas Math., Winnipeg, Man., 1978.

[16] Sheng-Lung Peng, Chin-Wen Ho, Tsan-sheng Hsu, Ming-Tat Ko, and Chuan Yi Tang. Edge and node searching problems on trees. *Theoretical Computer Science*, 240(2):429–446, 2000.

[17] Konstantin Skodinis. Computing optimal linear layouts of trees in linear time. *J. Algorithms*, 47(1):40–59, 2003.

[18] Karol Suchan and Ioan Todinca. Pathwidth of circular-arc graphs. In *33rd Int. Workshop on Graph-Theoretic Concepts in Computer Sc. (WG)*, volume 4769 of *LNCS*, pages 258–269. Springer, 2007.

[19] Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, 2009.