Exclusive Graph Searching

Lélia Blin, Janna Burman, Nicolas Nisse

▶ **To cite this version:**

Lélia Blin, Janna Burman, Nicolas Nisse. Exclusive Graph Searching. 21st European Symposium on Algorithms (ESA 2013), Sep 2013, Sophia Antipolis, France. pp.181-192, 10.1007/978-3-642-40450-4_16 . hal-00845530

HAL Id: hal-00845530
https://hal.science/hal-00845530

Submitted on 17 Jul 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Exclusive Graph Searching

Lélia Blin[1], Janna Burman[2], and Nicolas Nisse[3]

[1] Université d'Evry Val d'Essonne and LIP6-CNRS, France, lelia.blin@lip6.fr
[2] LRI (CNRS/UPSud), Orsay, France, janna.burman@lri.fr
[3] COATI, Inria, I3S (CNRS/UNS), Sophia Antipolis, France, nicolas.nisse@inria.fr

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. 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 investigate *exclusive* graph searching, in which no two or more searchers can occupy the same node at the same time, and, 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*. Nevertheless, we design a polynomial-time algorithm which, given any tree T, computes the exclusive search number of T. Moreover, for any integer k, we provide a characterization of the trees T with exclusive search number at most k. This characterization allows us to describe a special type of exclusive search strategies, that can be executed in a distributed environment, i.e., in a framework in which the searchers are limited to cooperate in a distributed manner.

1 Introduction

Graph Searching was first introduced by Breisch [9,10] in the context of speleology, for solving the problem of rescuing a lost speleologist in a network of caves. Alternatively, graph searching can be defined as a particular type of cops-and-robber game, as follows. Given a graph G, modeling any kind of network, 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 [22]. 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 line, while two searchers are necessary in a ring: the search number of any line is 1, while the search number of any ring is 2.

The above variant of graph searching is actually called *mixed-search* [4]. Other classical variants of graph searching are *node-search* [3], *edge-search* [21, 22], *connected-search* [2], etc. All these variants suffer from two serious limitations as far as practical applications are concerned.

– First, they all 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: searcher 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 [2], 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. 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 $\mathtt{xs}(G)$.

We show 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 minor* taking (not even under subgraph). That is,

there are graphs G and H such that H is a subgraph of G, and $\mathtt{xs}(H) > \mathtt{xs}(G)$. 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 Sec. 2, we formally define exclusive graph searching and present basic properties for general graphs. Motivated by certain positive results for trees and inspired by the pioneering work of Parson [22] and Megiddo et al. [21], we are then essentially focussing 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 $O(\log n)$. Our main result (Sec. 3) is a polynomial-time algorithm which, given any tree T, computes the exclusive search number $\mathtt{xs}(T)$ of T, as well as an exclusive search strategy using $\mathtt{xs}(T)$ searchers for clearing T. Our algorithm is based on a characterization of the trees with exclusive search number at most k, for any given $k \leq n$.

The above characterization allows us to describe an important type of exclusive search strategies, that can be executed in a distributed environment, i.e., in a framework in which the searchers are restricted to cooperate in a distributed manner (Sec. 4). More specifically, we consider the classical (discrete) CORDA[4] (a.k.a. *Look-Compute-Move*) model [16, 24] for *autonomous* searchers moving in a network. We prove that, for any anonymous asymmetric tree T, as well as for any tree whose nodes are labeled with unique IDs, and for any $n \geq k \geq \mathtt{xs}(T)$, there exists a distributed protocol enabling k searchers to clear T.

Hence, an interesting outcome of this paper is that the minimum number of searchers needed to clear an (anonymous asymmetric or uniquely labeled) tree in a distributed manner is not larger than the one required when the searchers are coordinated and scheduled by a central entity. This is particularly surprising, especially when having in mind that, in the distributed setting, symmetry breaking becomes much more harder (even in an asymmetric network), and the scheduling of the searchers (i.e., which searchers are activated at any point in time) is under the full control of an adversary. Due to the lack of space, most of the proofs are omitted or sketched. All complete proofs can be found in [6].

Related Work. Graph searching has mainly been studied in the centralized setting for its relationship with the *treewidth* and *pathwidth* of graphs [4,18]. The problem of computing the search number of a graph is NP-hard [21]. However, this problem is polynomial in various graph classes [17, 19, 26]. In particular, it has been widely studied in the class of trees [14, 21–23, 25].

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 [4]. 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. Instead, connected graph searching, in which the set of clear edges must always induce a connected

[4] COordination of Robots in a Distributed and Asynchronous environment.

subgraph, is not monotone in general [27] and it is not known if connected search is in NP. Connected search is monotone in trees [2]. The connectivity constraint may increase the search number of any graph by a factor up to 2 [13].

Graph searching has been intensively studied in various distributed settings (see, e.g., [7,11,15,20]). Graph searching in the CORDA model has recently been studied for rings [12]. The exclusivity constraint has been already considered in the context of various coordination tasks for mobile entities in enhanced versions of the CORDA model (see, e.g., [1, 8, 12]). In the context of graph searching, the exclusivity constraint has been considered for the first time in our brief announcement [5]. Here, we present and improve some results announced in [5].

2 Exclusive Search

In this section, we provide the formal definition of exclusive graph searching, and present some basic general properties.

Given a connected 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 performed 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 $\texttt{xs}(G)$ is the smallest 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 n 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 (see [6]):

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

This result shows 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 $O(\log n)$ [22]. On the positive side, we show that, for any graph G with maximum degree Δ, $\mathtt{s}(G) \leq \mathtt{xs}(G) \leq (\Delta - 1)\mathtt{s}(G)$ [6]. To prove it, we consider a classical strategy $\mathcal{S}$ for G using $\mathtt{s}(G)$ searchers. To build an exclusive strategy $\mathcal{S}_{ex}$ for G, we mimic $\mathcal{S}$ using a team of $\Delta - 1$ searchers to "simulate" each searcher in $\mathcal{S}$.

We now turn our attention to the 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, when a searcher needs to cross another one, letting the former searcher pass may lead to recontaminate some edges. In spite of that, the goal of the winning strategy is to prevent an "uncontrolled" recontamination. In [6], we prove that:

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

Last, but not least, contrary to classical graph searching, exclusive graph searching is not closed under minor. Indeed, even taking a subgraph can decrease the connectivity which, surprisingly, may not help the searchers (due to the exclusivity constraint). That is, there exist a graph G and a subgraph H of G such that $\mathtt{xs}(H) > \mathtt{xs}(G)$ [6]. Nevertheless, exclusive-search is closed under subgraph in trees (see [6]):

Lemma 1. *For any tree T and any subtree T' of T, $\mathtt{xs}(T') \leq \mathtt{xs}(T)$.*

Contrary to classical graph searching, the proof of this result is not trivial because of the exclusivity property. To prove it, we have to transform an exclusive strategy $\mathcal{S}$ for T into a strategy $\mathcal{S}'$ for T' using the same number of searchers, and without violating the exclusivity property. The fact that $\mathcal{S}$ may be not monotone (i.e., some recontamination may occur during $\mathcal{S}$) makes the proof technical, because one has to "control" the recontamination of T' in $\mathcal{S}'$.

3 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. Given a node v in a tree T, a connected component

of $T \setminus \{v\}$ is called a *branch* at v. 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. More precisely, we prove that:

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

1. *v has degree at most $k+1$;*
2. *for any branch B at v, $\texttt{xs}(B) \leq k$;*
3. *for any even $i > 1$, at most i branches B at v have $\texttt{xs}(B) \geq k - i/2 + 1$.*

To prove the theorem, we first prove (Sec. 3.1) that, for any tree T and $k \geq 1$, $\texttt{xs}(T) \leq k$, only if the conditions of Th. 1 are satisfied. Then, we show that any tree satisfying these conditions can be decomposed in a particular way, depending on k (Fig. 1). Next, in Sec. 3.2, we describe an exclusive search strategy using at most k searchers, that clears any tree decomposed in such a way.

From the characterization of Th. 1, it follows that $\texttt{xs}(T)$ can be computed by dynamic programming on T. Moreover, such an algorithm computes the corresponding decomposition (see Section 3.2). Hence, the following result holds:

Theorem 2. *There exists a polynomial-time algorithm that computes $\texttt{xs}(T)$ and a corresponding exclusive search strategy for any tree T.*

We now prove Theorem 1 using the following notations. For a node $v \in T$, we denote by $N(v)$ the set of the neighbors of v. A *configuration* is a set of distinct nodes $C \subseteq V(T)$ that describes the positions of $|C|$ searchers in T.

3.1 Necessary Conditions for Theorem 1

We first show that the conditions of Theorem 1 are necessary. The fact that the first property is necessary directly follows from Claim 1. The second property is necessary by Lemma 1.

For proving that the third property is necessary, we first have to prove that, for any tree T, any branch B of T, and any exclusive strategy for T, there is a step of the strategy where at least $\texttt{xs}(B)$ searchers are occupying the nodes of B (see [6]). While such a result is trivial in classical graph searching, it is not the case anymore subject to exclusivity and internality properties. In particular, in classical graph searching, the result is true for any subtree (not necessarily a branch) while it is not the case for the exclusive variant (see [6]). Indeed, let us consider a sub-tree T' of tree T. If T' is given independently of T, the movements of searchers are more constrained because the searchers have less "space" in T'. On the contrary, when T' is inside the tree T, the searchers can use the "extra space" provides by T to clear T'.

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 $B = \{T_j : \texttt{xs}(T_j) \geq k - i/2 + 1\}$ of branches at v and $|B| > i$, then $\texttt{xs}(T) > k$.*

Proof. Let $\mathcal{S}$ be any exclusive strategy that clears T. By the remark above, 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 vertices 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 vertices 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 - 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 - a contradiction. □

Decomposition. Figure 1 presents a particular structure that we prove to exist for any tree T satisfying the properties of Theorem 1, for $k \geq 1$. Specifically, following [21], we prove that there is a unique path $A = (u_1, \cdots, u_p)$ in T called *avenue* 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., $\mathtt{xs}(T') < k$ (bold line in Fig. 1).

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.

3.2 Exclusive Search Strategy to Clear Trees

Let $k \geq 1$ and let T be any tree satisfying Theorem 1 and thus, given with the decomposition of Figure 1. In this section, we informally describe a search

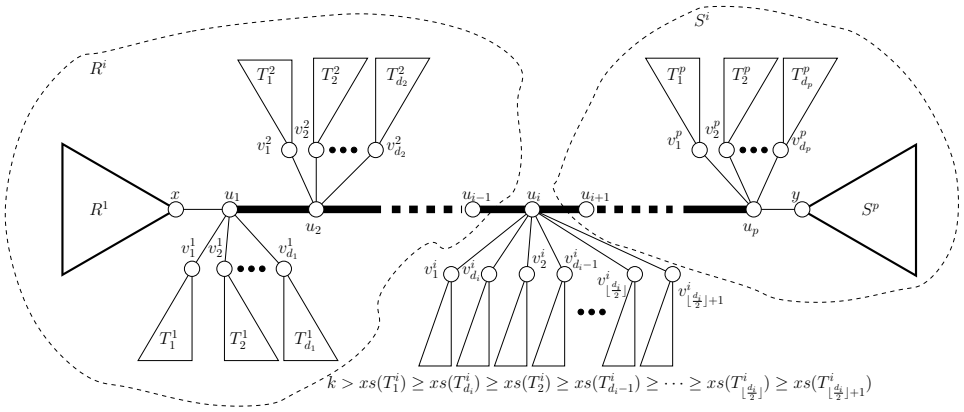


Fig. 1. A tree T with avenue $A = (u_1, \cdots, u_p)$. For any subtree X of $T \setminus A$, $\mathtt{xs}(X) < k$.

strategy that clears T using k searchers. By definition, the following strategy ensures that all moves are performed along paths free of searchers, satisfying the exclusivity and internally properties. 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. The formal description of the strategy and the complete correctness proof are provided in [6].

Strategy *ExclusiveClear*. For ease of description, let us assume that $|V(R^1)| \geq k-1$ (see Fig. 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. 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 and by the definition of A). 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.

Then, we aim at clearing the remaining subtrees of $T \setminus A$ that are adjacent to u_1 ($T_1^1, \cdots, T_{d_i}^1$, in Fig. 1). Moreover, after clearing such a subtree, we need to preserve it from recontamination. Notice that, during the clearing of a subtree, u_1 will always be occupied. However, to ensure that exclusivity property is satisfied when searchers go from one subtree to another (during the *bring_searchers* procedure explained later), we need other nodes being occupied.

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 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 a set S_1, the second one to a set 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 1 respects this order. The resulting $S_1 = \{T_1^1, \ldots, T_{\lceil d_i/2 \rceil}^1\}$ and $S_2 = \{T_{\lceil d_i/2+1 \rceil}^1, \ldots, T_{d_i}^1\}$

The clearing of the subtrees is then divided into two phases. The subtrees in S_1 are cleared first, in the non-increasing order of their `xs`. Then, the subtrees in S_2 are cleared in the non-decreasing order of their `xs`. Each time that a subtree $T_j^1 \in S_1$ has been cleared, one searcher is left on its *root* v_j^1 (its node adjacent to u_1). That is, once a new subtree is cleared, we somehow lose a searcher to clear the next one. This is balanced by the fact that the number of searchers needed to clear the next subtree *decrease*, according to the order of clearing established above, and provided by the properties of T (Th. 1).

After clearing the subtrees in S_1, there are searchers currently "blocked" in the roots of the cleared subtrees. In order to "re-use" these searchers to clear the remaining subtrees, the strategy changes. Now, the roots of the contaminated subtrees will be occupied to prevent recontamination of the cleared subtrees. 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.

Then, the subtrees in S_2 have to be cleared in the non-decreasing order of their `xs`. Each time that a subtree $T_j^1 \in S_2$ has been cleared, the searcher on its root v_j^1 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.

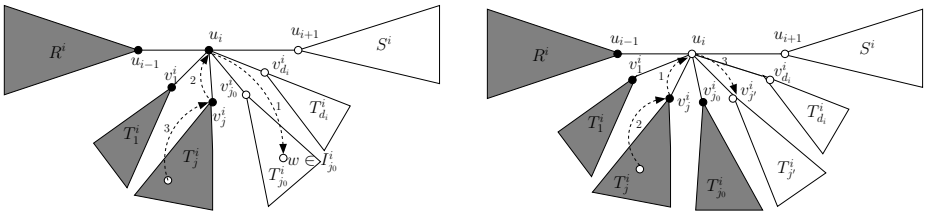
Once all subtrees of $T \setminus A$ adjacent to u_1 are cleared, the searcher at u_1 goes to u_2 (unless it is already occupied). Now, all the searchers in R^2 (see Fig. 1) became free. Then, a similar strategy is applied for the subtrees of $T \setminus A$ adjacent to u_2, and so on, until all the subtrees adjacent to u_p are cleared.

We now describe more precisely two sub-procedures that are used to implement the strategy we have sketched 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. 1) 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. 2(a)). There are two cases to be considered: $j_0 \leq \lceil \frac{d_i}{2} \rceil$ or otherwise.

Assume first that $j_0 \leq \lceil \frac{d_i}{2} \rceil$. 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. The process *bring_searchers*(i, j_0) is applied to bring $\texttt{xs}(T_{j_0}^i)$ searchers into $T_{j_0}^i$. The searchers are brought one by one, from the clear part to $T_{j_0}^i$, without recontamination and satisfying the exclusivity property.

Fig. 2(a) depicts one phase of this process. We prove that, before each phase (but the last one, which is slightly different), there is a free searcher at some node b, either in $R^i \setminus u_{i-1}$ or in $T_j^i \setminus v_j^i$ (for some $j < j_0$). First, the searcher at u_i goes to the furthest unoccupied node in $T_{j_0}^i$ (dotted line 1 in Fig. 2(a)). Second, the searcher at v_j^i (or at u_{i-1}) goes to u_i (dotted line 2 in Fig. 2(a)). Finally, the searcher at b goes to v_j^i (or to u_{i-1}) (dotted line 3 in Fig. 2(a)). Clearly, doing



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

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

Fig. 2. Black nodes are occupied. Grey subtrees are cleared. Steps are depicted by dotted arrows.

so, no recontamination occurs in the cleared subtrees (but in u_i) and exclusivity is satisfied. We apply similar techniques for $j_0 > \lceil \frac{d_i}{2} \rceil$.

Procedure *transfer*. Let $1 \leq i \leq p$ and $j_0 = \lceil d_i/2 \rceil$. Just after clearing $T^i_{j_0}$, we reach a configuration where the nodes in $\{u_i, v^i_1, \cdots, v^i_{j_0}\}$ are occupied, $T^i_{j_0}$ is clear, and all other searchers are at nodes of R^i or T^i_j $(j \leq j_0)$. 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.

Fig. 2(b) depicts one phase of this process. We prove that, before each phase, there is a free searcher at some node b either in $R^i \setminus \{u_{i-1}\}$ or in $T^i_j \setminus \{v^i_j\}$ (for some $j \leq j_0$). 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. 2(b)). Second, the searcher at b goes to v^i_j (or u_{i-1}) (dotted line 2 in Fig. 2(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. 2(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 Application to Distributed Graph Searching

In the previous section, we have described the *ExclusiveClear* strategy using $\texttt{xs}(T)$ searchers controlled by central scheduler. In this section, we briefly explain how this strategy can be adapted, and then used by the *autonomous* searchers operating in an asynchronous distributed manner (see [6] for more details).

We consider a version of the classical discrete CORDA model for autonomous mobile searchers. The searchers operate in *asynchronous* cycles of *Look-Compute-Move*. During its Look action, a searcher (instantaneously) takes a snapshot of

the network map together with the relative positions of all searchers. Based on this information, during the Compute action, it computes (deterministically) the next neighboring node where to move. During the Move action, the searcher (instantaneously) changes its position according to its computation. There is a finite but unbounded delay between any two actions. Moreover, the searchers are *anonymous*, *uniform* (i.e., each searcher executes the same algorithm), *oblivious* (i.e., memoryless to observations and to computations performed in previous cycles) and have *no sense of direction*.

Let us consider an asymmetric tree, i.e., a tree with no non-trivial automorphisms. We show that in such a tree, each searcher can assign distinct labels to the nodes such that each node of the tree is always given the same label by all searchers during any Compute action [6]. Hence, in CORDA, an anonymous asymmetric tree can be seen as uniquely labeled.

To clear a tree T in the distributed setting, at least $\texttt{xs}(T)$ searchers are placed in the specified different nodes of T, forming an initial configuration I. At each Compute action, a searcher computes a winning strategy S, starting from I, and using $\texttt{xs}(T)$ searchers. Given any achievable configuration, S describes the required move (of one of the searchers). S follows the same structure as *ExclusiveClear*, inductively clearing the subtrees of $T \setminus A$, where A is the avenue. However, in contrast with *ExclusiveClear*, the main difficulty is to ensure that all configurations in S are pairwise distinct. Otherwise, since searchers are oblivious, they could enter in a loop of configurations, and the clearing would fail. In particular, an attention should be paid to the case where searchers just have cleared a subtree of $T \setminus A$, and must go back towards A to clear the next subtree. Moreover, in *ExclusiveClear*, it may happen that a searcher slides back and forth along the same edge, while no other searchers have moved. This must be avoided in S. The case of a labeled line subtree of $T \setminus A$ is particularly tricky: an extra searcher is required to clear it compared to *ExclusiveClear*. Nevertheless, we succeed to use only $\texttt{xs}(T)$ searchers, even in the distributed case.

Theorem 3. *For any anonymous asymmetric or any uniquely labeled n-node tree T, and for any integer k with $\texttt{xs}(T) \leq k \leq n$, there exists a distributed protocol in the discrete CORDA model enabling k searchers to clear T.*

References

1. R. Baldoni, F. Bonnet, A. Milani, and M. Raynal. Anonymous graph exploration without collision by mobile robots. *Inf. Process. Lett.*, 109(2):98–103, 2008.
2. L. Barrière, P. Flocchini, F. V. Fomin, P. Fraigniaud, N. Nisse, N. Santoro, and D. M. Thilikos. Connected graph searching. *Inf. Comput.*, 219:1–16, 2012.
3. D. 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.
4. D. Bienstock and P. D. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.
5. L. Blin, J. Burman, and N. Nisse. Brief announcement: Distributed exclusive and perpetual tree searching. In *26th International Symposium on Distributed Computing (DISC)*, volume 7611 of *LNCS*, pages 403–404. Springer, 2012.

6. L. Blin, J. Burman, and N. Nisse. Exclusive graph searching. Technical report, INRIA, 2013. URL `http://hal.archives-ouvertes.fr/hal-00837543`.
7. L. Blin, P. Fraigniaud, N. Nisse, and S. Vial. Distributed chasing of network intruders. *Theor. Comput. Sci.*, 399(1-2):12–37, 2008.
8. L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. In *24th Int. Symp. on Distributed Computing (DISC)*, volume 6343 of *LNCS*, pages 312–327. Springer, 2010.
9. R. L. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, 6: 72–78, 1967.
10. R.L. Breisch. *Lost in a Cave-applying graph theory to cave exploration.* 2012.
11. D. Coudert, F. Huc, and D. Mazauric. A distributed algorithm for computing the node search number in trees. *Algorithmica*, 63(1-2):158–190, 2012.
12. G. D'Angelo, G. Di Stefano, A. Navarra, N. Nisse, and K. Suchan. A unified approach for different tasks on rings in robot-based computing systems. In *15th Workshop on Advances in Par. and Dist. Comp. Models (APDCM)*. IEEE, 2013.
13. Dariusz Dereniowski. From pathwidth to connected pathwidth. *SIAM J. Discrete Math.*, 26(4):1709–1732, 2012.
14. J. A. Ellis, I. H. Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Inf. Comput.*, 113(1):50–79, 1994.
15. P. Flocchini, M.J. Huang, and F. L. Luccio. Contiguous search in the hypercube for capturing an intruder. In *19th Int. Par. and Dist. Proc. Symp. (IPDPS)*, 2005.
16. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *10th Int. Symp. on Algorithms and Computation (ISAAC)*, volume 1741 of *LNCS*, pages 93–102. Springer, 1999.
17. F.V. Fomin, P. Heggernes, and R. Mihai. Mixed search number and linear-width of interval and split graphs. *Networks*, 56(3):207–214, 2010.
18. F.V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.
19. P. Heggernes and R. Mihai. Edge search number of cographs in linear time. In *3rd Int. Workshop on Frontiers in Algorithmics (FAW)*, volume 5598 of *LNCS*, pages 16–26. Springer, 2009.
20. D. Ilcinkas, N. Nisse, and D. Soguet. The cost of monotonicity in distributed graph searching. *Distributed Comp.*, 22(2):117–127, 2009.
21. 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.
22. 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.
23. S-L. Peng, C-W. Ho, T-S. Hsu, M-T. Ko, and C. Y. Tang. Edge and node searching problems on trees. *TCS*, 240(2):429–446, 2000.
24. G. Prencipe. Instantaneous actions vs. full asynchronicity : Controlling and coordinating a set of autonomous mobile robots. In *ICTCS*, pages 154–171, 2001.
25. K. Skodinis. Computing optimal linear layouts of trees in linear time. *J. Algorithms*, 47(1):40–59, 2003.
26. K. Suchan and I. 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.
27. B. Yang, D. Dyer, and B. Alspach. Sweeping graphs with large clique number. In *15th Int. Symp. on Algorithms and Computation (ISAAC)*, volume 3341 of *LNCS*, pages 908–920. Springer, 2004.