The Cost of Monotonicity in Distributed Graph Searching

David Ilcinkas, Nicolas Nisse, David Soguet

Noname manuscript No.
(will be inserted by the editor)

The Cost of Monotonicity in Distributed Graph Searching

David Ilcinkas · **Nicolas Nisse** · **David Soguet**

the date of receipt and acceptance should be inserted later

Abstract Blin et al. [5] (TCS 2008) proposed a distributed protocol enabling the smallest possible number of searchers to clear any unknown graph in a decentralized manner. However, the strategy that is actually performed lacks of an important property, namely the monotonicity. This paper deals with the smallest number of searchers that are necessary and sufficient to monotonously clear any unknown graph in a decentralized manner. The clearing of the graph is required to be connected, i.e., the clear part of the graph must remain permanently connected, and monotone, i.e., the clear part of the graph only grows. We prove that a distributed protocol clearing any unknown n-node graph in a monotone connected way, in a decentralized setting, can achieve but cannot beat competitive ratio of $\Theta(\frac{n}{\log n})$, compared with the centralized minimum number of searchers. Moreover, our lower bound holds even in a synchronous setting, while our constructive upper bound holds even in an asynchronous setting.

Keywords Graph searching · Mobile agent · Monotonicity · Competitive ratio

A preliminary version of this paper appeared in the proceedings of the 11th International Conference On Principles Of Distributed Systems (OPODIS'07), LNCS 4878, pages 105-118.

D. Ilcinkas (corresponding author)
CNRS & Université de Bordeaux (LaBRI)
bât. A30, 351 cours de la Libération, 33405 Talence cedex, France
Tel.: +33 540-006-912, Fax: +33 540-006-669
E-mail: david.ilcinkas@labri.fr

N. Nisse
MASCOTTE, INRIA, I3S, CNRS, UNS, Sophia Antipolis, France. E-mail: nicolas.nisse@sophia.inria.fr

D. Soguet
LRI, Université Paris-Sud, Orsay, France
E-mail: david.soguet@lri.fr

1 Introduction

The graph searching problem [6,17] consists, for a team of *searchers*, in capturing an invisible arbitrarily fast fugitive hidden in a graph (see [10] for a recent survey). Equivalently, an undirected connected graph can be seen as a system of tunnels contaminated by a toxic gas. In this latter setting, the searchers have to clear, i.e., to decontaminate, the graph. We will use this latter formulation in the paper.

The *search problem* has been widely studied in the design of distributed protocols for clearing a network in a decentralized manner [5,7–9,14,16]. Initially, all edges are contaminated and all searchers are placed at a particular vertex of the graph, called the *homebase*. Subsequently the searchers stand at vertices of the graph and move along the edges. An edge is *cleared* when it is traversed by a searcher. A clear edge e is *recontaminated* as soon as there exists a path P between e and a contaminated edge such that no searchers are occupying any vertex or any edge of P. A *search strategy* is a sequence of moves of the searchers along the edges of the graph, such that *recontamination* never occurs, that is, a clear edge always remains clear. A search strategy is aiming at clearing the whole network. Given a graph G and a homebase $v_0 \in V(G)$, the search problem consists in designing a distributed protocol that allows the smallest possible number of searchers to clear G starting from v_0. The search strategy performed by the searchers is thus computed online by the searchers themselves.

In this paper, we define a search strategy as satisfaying two important properties. Firstly, a search strategy is *monotone* [4,13]. That is, the contaminated part of the graph never grows. This ensures that the clearing of the graph can be performed using a polynomial

number of steps (moves). Secondly, a search strategy is *connected* [1,2], in the sense that, at any step of the strategy, the clear part of the graph induces a connected subgraph. This latter property ensures safe communications between the searchers. In the following, the *search number* $\mathbf{mcs}(G,v_0)$ of a graph G with homebase $v_0 \in V(G)$ denotes the smallest number of searchers required to clear the graph in a monotone connected way, starting from v_0, in centralized settings.

Several distributed protocols have been proposed to solve the search problem [1,5,7–9,14,16]. Two main approaches have been used in the previous works. On one hand, Blin et al. proposed a distributed protocol that enables $\mathbf{mcs}(G,v_0)+1$ searchers to clear any *unknown* asynchronous graph G, starting from any homebase v_0, in a connected way [5]. That is, the clearing of the graph is performed without the searchers being provided any information about the graph. However, the search strategy that is actually performed is not monotone and may use an exponential number of steps, which is not surprising since the problem of computing $\mathbf{mcs}(G,v_0)$ is NP-complete [15]. On the other hand, the distributed protocols that are proposed in [7–9,14,16] enable $\mathbf{mcs}(G,v_0)$ searchers to monotonously clear a graph G, starting from a homebase v_0, such that the searchers are given some a priori information about it. In all these works, an extra searcher is required as soon as the network is asynchronous. In this paper, we consider the problem from another point of view. More precisely, we address the problem of the minimum number of searchers necessary to solve the search problem (again, the performed strategy must be connected and monotone) without any a priori information about the graph.

1.1 Model and definitions

We model the searchers by autonomous mobile computing entities with distinct IDs in $\{1,\cdots,|V(G)|\}$. A network is modeled by an undirected connected and simple graph. To strengthen our bounds, we assume that the searchers as well as the network are synchronous when proving our lower bound and asynchronous when proving our upper bound. The network is anonymous, that is, nodes are not labeled. The $\deg(u)$ edges incident to any node u are labeled from 1 to $\deg(u)$, so that the searchers can distinguish the different edges incident to a node. These labels are called *port numbers*. Every node of the network has a zone of local memory, called *whiteboard*, in which searchers can read, erase, and write symbols. Moreover, it is assumed that searchers can access these whiteboards in fair mutual exclusion.

We define a *search protocol* $\mathscr{P}$ as a distributed protocol that solves the search problem: for any connected graph G and any homebase $v_0 \in V(G)$, a team of searchers executing $\mathscr{P}$ can clear G in a monotone connected way, starting from v_0. In these settings, the searchers do not know in advance in which graph they are launched in. The number of searchers used by $\mathscr{P}$ to clear G is the maximum number of searchers that stand at the vertices of G over all steps of the execution of $\mathscr{P}$. The *cost* of a search protocol $\mathscr{P}$ in a graph G with homebase v_0 is measured by the ratio between the number of searchers it uses to clear G and the search number $\mathbf{mcs}(G,v_0)$ of G. This ratio, maximized over all graphs and all starting nodes, is called the *competitive ratio* $r(\mathscr{P})$ of the protocol $\mathscr{P}$.

1.2 Our results

We prove that any search protocol for clearing n-node graphs has competitive ratio $\Omega(\frac{n}{\log n})$. Moreover, we propose an optimal search protocol that has competitive ratio $O(\frac{n}{\log n})$. More precisely, we prove that for any distributed protocol $\mathscr{P}$, there exists a constant c such that for any sufficiently large n, there exists a n-node graph G with a homebase $v_0 \in V(G)$, such that $\mathscr{P}$ requires at least $c\frac{n}{\log n}\,\mathbf{mcs}(G,v_0)$ searchers to clear G, starting from v_0. Note that this n-node graph G is a tree with maximum degree 3. On the other hand, we propose a distributed search protocol that uses at most $O(\frac{n}{\log n})\,\mathbf{mcs}(G,v_0)$ searchers to clear any connected graph G in a monotone connected way, starting from any homebase $v_0 \in V(G)$. Moreover, our protocol performs clearing of n-node graphs using searchers with at most $O(\log n)$ bits of memory, and whiteboards of size $O(n)$ bits. Note that the lower bound holds even in a synchronous setting, while our protocol can be implemented even in an asynchronous setting.

1.3 Related work

In the problem of connected graph searching [1,2,11,12,18], the clear part must remain connected during all steps of the search strategy. This property is very useful as soon as we want to ensure secure communications between the searchers. Contrary to the non-connected graph searching [3,4,6,13,17] where monotonicity can be ensured for free, monotonicity in the connected version of the problem generally requires more searchers. Indeed, Alspash et al. proved that *recontamination does help* in the case of connected graph searching [18] (see also [12]). That is, they describe a class of graphs for which the smallest number of

searchers required to connectedly clear these graphs is strictly less than the number of searchers necessary to clear them in a monotone connected way. This result has an important impact since it is not known whether the decision problem corresponding to the connected search number of a graph, i.e., the smallest number of searchers required to clear a graph in a connected way, belongs to NP. Moreover, monotone strategies are of particular interest since, first, they perform in a polynomial number of steps, and second, it is a priori difficult to design non-monotone search strategies.

Several distributed protocols have been proposed to solve the search problem for particular graph's topologies. More precisely, Barrière et al. designed protocols for clearing trees [1], Flocchini, Luccio and Song considered chordal rings and tori [7] and meshes [9], Flocchini, Huang and Luccio considered hypercubes [8], and Luccio dealt with Sierpinski's graphs [14]. Assuming the searchers know the topology of the asynchronous network G they must clear, these protocols enable $\mathbf{mcs}(G,v_0)+1$ searchers to clear G in a monotone connected way, starting from any homebase $v_0 \in V(G)$. The extra searcher, in comparison with the centralized case, is necessary and due to the asynchrony of the network [9]. In [5], Blin et al. proposed a distributed protocol allowing $\mathbf{mcs}(G,v_0)+1$ searchers to clear any unknown asynchronous graph G in a connected way, starting from any homebase $v_0 \in V(G)$. In this case, the searchers do not need any a priori information about the graph in which they are placed. However, the search strategy actually performed is not monotone and may be performed using an exponential number of steps. In [16], Nisse and Soguet proposed to give to the searchers some information about the graph by putting short labels on the nodes of the graph. They proved that $\Theta(n\log n)$ bits of information are necessary and sufficient to solve the search problem for any n-node asynchronous graph G, starting from a homebase v_0 and using $\mathbf{mcs}(G,v_0)+1$ searchers.

2 Lower Bound on the Competitive Ratio

In this section, we assume that the searchers and the network are synchronous. This section is devoted to prove a lower bound on the competitive ratio of any search protocol. For this purpose, we consider a game between an arbitrary search protocol and an adversary. Roughly speaking, the adversary gradually builds the graph, which is actually a ternary tree, as the search protocol clears it in a monotone connected way. The role of the adversary is to force the protocol to use the maximum possible number of searchers to clear the graph. The fact that the adversary can build the graph during the execution of the search protocol is possible since the searchers have no information on the graph they are clearing.

We need the following definition. A *partial graph* is a simple connected graph which can have edges with only one end. Edges with one single end (resp., two ends) are called *half-edges* (resp., *full-edges*). Let $G=(V,H,F)$ be a partial graph, where V is the vertex-set of G, H its set of half-edges and F its set of full-edges. Let G^- be the graph (V,F), with the same vertex-set than G and edge-set F. Let G^+ be the graph obtained by adding a degree-one end to any half-edge of G.

Let us give some definitions and results that will be used in the following. A *ternary tree* is a tree of maximum degree at most three. A search strategy that is not constrained to satisfy neither the connected property, nor the monotone property is simply a sequence of moves of the searchers along the edges of a graph resulting in clearing the whole graph. Let $\mathbf{s}(G)$ denote the smallest number of searchers that are necessary to clear a graph G in such a way. A lot of research has been done regarding the graph searching problem in the class of trees. In particular, the following results are known.

Theorem 1 *Let T be a tree with $n \geq 2$ vertices,*

- $\mathbf{s}(T) \leq 1+\log_3(n-1)$ **(Megiddo et al. [15])**
- $\forall v_0 \in V(T)$, $\mathbf{mcs}(T,v_0) \leq 2\mathbf{s}(T)-1$ **(Barrière et al. [2])**

The remaining part of this section is devoted to the proof of Theorem 2. Recall that a search protocol has been defined as a distributed algorithm for clearing a graph in a monotone connected way.

Theorem 2 *Any search protocol for clearing n-node graphs has competitive ratio $\Omega(\frac{n}{\log n})$.*

Proof Let $\mathscr{P}$ be a (successful) search protocol. We prove that there exists a constant $c>0$, such that for any $n \geq 5$, there exists a ternary n-node tree T (actually, T has exactly one internal vertex of degree two if n is odd, and none otherwise), such that $\mathscr{P}$ uses at least q searchers to clear T in a monotone connected way, starting from any homebase $v_0 \in V(T)$, with $q \geq c\,\frac{n}{\log n}\,\mathbf{mcs}(T,v_0)$.

Fix $n \geq 5$. We will construct an n-node ternary tree T, that $\mathscr{P}$ has to clear starting from $v_0 \in V(T)$. Let us describe the game executed turn by turn by $\mathscr{P}$ and the adversary $\mathscr{A}$. This game progressively constructs a partial graph T_p that ends up being the tree T for which the cost of the search protocol $\mathscr{P}$ is high.

Initially, the partial graph T_p consists of a single vertex, the homebase v_0, incident to three half-edges.

All searchers are placed at v_0. Then, $\mathscr{P}$ and $\mathscr{A}$ play alternatively, starting with $\mathscr{P}$. At each round, $T_p = (V,H,F)$ corresponds to the part of T that $\mathscr{P}$ currently knows. At each round, the search protocol $\mathscr{P}$ chooses a searcher and it moves this searcher along an edge e of T_p in such a way that recontamination does not occur. Such a move is always possible since $\mathscr{P}$ is a successful search protocol, and thus, it eventually clears T in a monotone connected way. Note that e may be a half-edge or a full-edge. If e is a full-edge, then $\mathscr{A}$ skips its turn. Otherwise, two cases must be considered. Either $|V(T_p^+)| < n-1$, or $|V(T_p^+)| = n-1$. In the first case, $\mathscr{A}$ adds a new end v to e such that v is incident to two new half-edges f and h. That is, the partial graph becomes $T_p = (V \cup \{v\}, H_{new}, F_{new})$, with $H_{new} = (H \setminus \{e\}) \cup \{f\} \cup \{h\}$ and $F_{new} = F \cup \{e\}$. In the second case, $\mathscr{A}$ adds a new end v to e such that v is incident to only one new half-edge f. Again, this is possible since $\mathscr{P}$ does not know the graph in advance. The game ends when $|V(T_p^+)| = n$. At such a round, $\mathscr{A}$ decides that the graph T is actually T_p^+.

Let us first do the following easy remarks. At each round of the game, T_p^- is a ternary tree, and T_p^+ is a ternary tree with at least $\lfloor (n'+2)/2 \rfloor$ leaves, where n' is the number of vertices of T_p^+ (this can be easily proved by induction on the number of rounds). Moreover, T_p^- is exactly the clear part of T at this step of the execution of $\mathscr{P}$. In other words, the half-edges of T_p corresponds to the contaminated edges that are incident to the clear part of T. Since the execution of $\mathscr{P}$ ensures that the performed strategy is monotone, it follows that, at any round of the game, the vertices incident to at least one half-edge are occupied by a searcher. Let us consider the last round r, that is when $|V(T_p^+)|$ equals n. We show that at this round the number of vertices of T_p^+ occupied by searchers is at least $\lfloor n/4 \rfloor$. From the previous remarks, it follows that T_p^+ at round r, that is T, is a ternary tree with at least $\lfloor (n+2)/4 \rfloor$ vertices occupied by a searcher. Indeed, every parent of a leaf in T must be occupied by a searcher, and every vertex is parent of at most two leaves. Thus, $\mathscr{P}$ uses at least $q \geq \lfloor n/4 \rfloor$ searchers. By Theorem 1, $\mathbf{mcs}(T,v_0) \leq 1 + 2\log_3(n-1)$. Therefore,

$$q \geq \frac{\mathbf{mcs}(T,v_0)}{1+2\log_3(n-1)} \times \lfloor n/4 \rfloor \ .$$

It follows easily that there is a constant $c > 0$ such that for any $n \geq 5$ we have

$$q \geq c \, \frac{n}{\log n} \, \mathbf{mcs}(T,v_0) \ ,$$

which concludes the proof of the theorem. □

3 An Algorithm of Optimal Competitive Ratio

In this section, we assume that both the searchers and the network are asynchronous. We propose a search protocol named `mc_search` (for monotone connected search) having competitive ratio of $O(\frac{n}{\log n})$ for any n-node graph. The lower bound we proved in Section 2 shows that this distributed search protocol has thus an optimal competitive ratio of $\Theta(\frac{n}{\log n})$.

Before describing the search protocol `mc_search`, we need some definitions. In the following, the *depth* of a rooted tree T is the maximum length of the paths between the root and any leaf of T. Let v be a vertex of the rooted tree T that is not the root, and let u be the parent of v. The edge $\{u,v\}$ is called the *parent-edge* of v.

A *complete ternary tree* is defined as follows. The complete ternary tree T_0, of depth 0, consists of a single vertex, called its root. For any $k \geq 1$, a complete ternary tree T_k, of depth k, is a ternary tree in which all internal vertices have degree exactly three, and there exists a vertex, called its root, that is at distance exactly k from all leaves.

Finally, for any graph G, we define $\mathbf{mcs}(G)$ to be $\min_{v \in V(G)} \mathbf{mcs}(G,v)$.

Theorem 3 (Barrière et al. [2])
For any $k \geq 0$, $\mathbf{mcs}(T_k) = k+1$.

A graph H is a *minor* of a graph G if H is a subgraph of a graph obtained by a succession of edge contractions[1] of G. A well known result is that, for any graph G and any minor H of G, $\mathbf{s}(G) \geq \mathbf{s}(H)$ (folklore). Note that this result is not valid for the search number **mcs**, i.e., there exist some graphs G, and H minor of G such that $\mathbf{mcs}(H) > \mathbf{mcs}(G)$ [2].

3.1 General ideas of protocol `mc_search`

Before going through the description of our protocol, let us first consider some general characteristics of the clearing of a connected graph by some simple search protocol. At every step, the clear part of the graph induces a connected subgraph containing the homebase. Any vertex of the clear part that is incident to a contaminated edge is occupied by a searcher, preserving the clear part from recontamination. The set of such vertices is called the *border* of the clear part. Thus, after having cleared a contaminated edge, a searcher checks whether it is preserving the clear part of the graph from

[1] The *contraction* of the edge e with endpoints u,v is the replacement of u and v with a single vertex whose neighbors are the vertices that were neighbors of u or v.

recontamination. If its current vertex is incident to at least one contaminated edge and if it is not guarded by another searcher, then the searcher has to stay at the current vertex to prevent recontamination.

The main issue of the search protocol consists in deciding the next contaminated edge to be cleared. Note that, because of the connectedness of the strategy, the next edge to be cleared must be an edge incident to a vertex in the clear part. Another issue of the search protocol is to ensure that a searcher not standing at the border of the clear part of the graph is always able to reach the chosen edge through the clear part. We now briefly describe how our protocol `mc_search` deals with these issues.

Let G be a connected n-node graph and $v_0 \in V(G)$. Throughout the execution of the algorithm, `mc_search` dynamically maintains a rooted ternary subtree T of the (current) clear part of G. The root of T will be used to host all the currently "unused" searchers. More precisely, the tree T is required to cover all vertices occupied by at least one searcher. Thanks to this property, T will be used by the searchers to go from the root of T to a vertex of the border of the clear part in order to clear a contaminated edge. After having cleared a contaminated edge, and if the new position of the searcher does not lie at the border of the clear part or is already occupied by another searcher, the former searcher will also use T to go back to the root of T. This can easily be done by performing a DFS of T.

Furthermore, our protocol `mc_search` maintains the property that a searcher lies, at least, at every vertex of the border of the clear part, and at every vertex of degree 3 of T. In addition, a goal of our protocol is to keep T small and to use few agents. As a consequence, if a searcher occupies a vertex v not incident to any contaminated edge and of degree at most two, then our protocol relieves this useless searcher and send it to the root. Of course, if v was the root, then the root is also moved elsewhere in the tree. Additionally, if v is a leaf in T, then this leaf is removed together with the longest path without searchers leading to it, because this branch is of no use anymore to cover the vertices occupied by searchers.

The protocol `mc_search` also maintains a second rooted subtree S that is defined as a minor of T. More precisely, S is obtained from T by contracting all edges $\{u,v\}$ of T such that u is the parent of v and v is not occupied by a searcher. In other words, S represents the structure of T with respect to the searchers. That is, if there is a path in T without searchers but connecting two vertices occupied by some searchers, then this path is contracted to a single edge in S. Thus, in some sense, every vertex of S is occupied by a searcher. We define the root of S to be the root of T.

The subtree S is used by Protocol `mc_search` to decide the next contaminated edge to be cleared. Indeed, at each step, Protocol `mc_search` decides to clear an edge of G that is chosen such that S becomes as close as possible to a complete ternary tree. More precisely, at each step, Protocol `mc_search` will choose the next contaminated edge to be cleared in such a way that S remains of degree at most three and such that the depth of S may be increased from $k \geq 0$ to $k+1$ only if S was isomorphic to T_k at a previous step.

The intuitive reason of this choice is that the complete ternary tree is the tree requiring the (asymptotic) largest number of searchers compared to the size of the tree, even for a centralized algorithm. Thus, if the adversary forces our protocol to use a lot of searchers by choosing a graph for which almost every cleared edge leads to a new vertex (basic idea of the proof of the lower bound), then our protocol forces the chosen graph to have a large complete ternary tree as a minor, and thus even a centralized algorithm needs a logarithmic number of agents to clear the graph. This is the intuitive reason why our protocol `mc_search` achieves the optimal competitive ratio $\Theta(n/\log n)$.

Figure 1 shows a state of a graph at some step of the execution of Protocol `mc_search`. The light gray part represents the clear part of the graph at this step. The tree T rooted in r is depicted using bold edges. Dotted edges represent those edges of the clear part that belong to both T and S. (That is, S is obtained from T by contracting the non-dotted edges of T.) Dark gray vertices are those occupied by searchers at this step, i.e., the vertices of S. In this example, the next edge to be cleared must be an edge incident to e or f.

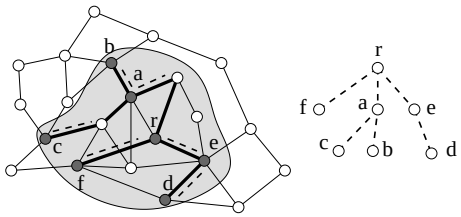

Fig. 1 Protocol `mc_search` maintains two subtrees: a subgraph T of the clear part covering the searchers (bold lines) and a minor S of T (dotted lines). The clear part appears in light gray. S is represented to the right.

3.2 Protocol `mc_search`

In this section, we describe the main features of Protocol `mc_search` that is also described in a more com-

pact way in Figure 6. For the purpose of simplifying the presentation, we assume in Figure 6 that searchers are able to communicate by exchanging messages of size $O(\log n)$ bits. This assumption is satisfied by using an additional searcher. This extra searcher is used to schedule the moves of the other searchers and to transmit few information between the searchers. For this purpose, the extra searcher performs a DFS of the tree T that enables it to reach any other searcher. Using this extra searcher enables Protocol `mc_search` to clear both synchronous and asynchronous networks. First, we describe the data structure used by `mc_search`.

The whiteboard of every vertex $v \in V(G)$ contains one vector $status_v$. For any edge $e \in E(G)$ incident to v, $status_v[e]$ takes a value in $L = \{Contaminated, Removed, Tree, Minor\}$. Initially, for any edge e with end v, $status_v[e] = Contaminated$. To simplify the presentation, we assume that each edge $e = \{u,v\} \in E(G)$ has only one label $\ell(e) = status_v[e] = status_u[e]$. This simplification may easily be implemented by the extra searcher. Indeed, each time an edge e is relabeled, the extra searcher does a return trip through e to synchronize the labels of both its ends. Moreover, the whiteboard of every vertex v contains a boolean $root_v$ which is true if and only if v is the current root of S and T.

The protocol is divided in $O(|E(G)|)$ phases. At each phase, Protocol `mc_search` relabels at least one edge. Moreover, an edge cannot be labeled twice using the same label. More precisely, an edge labeled *Contaminated* can only be relabeled *Minor* or *Removed*. Similarly, an edge labeled *Minor* (resp., *Tree*) can only be relabeled *Tree* or *Removed* (resp., *Removed*). Finally, the edges labeled with *Removed* are never relabeled. This proves that Protocol `mc_search` terminates.

Let us define some notations. At any step, T is the subgraph of G induced by the edges labeled *Minor* or *Tree*. In the next section, we prove that T is indeed a tree. S is the minor of T obtained by contracting all edges labeled *Tree*. Initially, T and S are rooted at v_0, the homebase. Finally, for any vertex $v \in V(G)$, m_v, t_v, r_v, c_v denote the number of edges incident to v that are respectively labeled *Minor*, *Tree*, *Removed*, *Contaminated*.

Every searcher has an integer state variable $\ell evel\ell$ in $\{0,\cdots,n\}$. Roughly, this variable indicates the distance between the vertex v currently occupied by the searcher and the root, in the tree S. Since S is obtained from T by contracting the edges labeled *Tree* and keeping the edges labeled *Minor*, this variable gives precisely the number of edges labeled *Minor* in the path between v and the root in the tree T. Initially, every searcher occupies the root v_0 and has $\ell evel\ell = 0$.

Let us describe a phase of the execution of Protocol `mc_search`. A phase starts by the election of the searcher that will perform moves and/or labellings of edges. The purpose of this searcher is to make the tree S as close as possible to a complete ternary tree. The elected searcher is an arbitrary searcher with minimum $\ell evel\ell$ and that occupies a vertex $v \in V(G)$ satisfying one of the following four conditions. Each of the four cases will be described in detail below.

Case a: $m_v + t_v \leq 2$ and $c_v \geq 1$,
Case b: $m_v + t_v = 1$ and $c_v = 0$,
Case c: $m_v + t_v = 2$, $c_v = 0$ and v is not the root,
Case d: $m_v + t_v = 2$, $c_v = 0$ and v is the root.

Roughly speaking, the goal in Case a is to make the ternary tree S (and thus T) grow by adding new incident edges to nodes of degree at most two, by clearing an incident contaminated edge. The other rules, Cases b, c and d, are designed to prune the tree S (but not necessarily T) in nodes where it cannot grow anymore, that is where the degree is less than three but where there are no incident contaminated edges to be added.

We will prove that, at any phase, any searcher actually occupies a vertex of S (more precisely, either the root or a vertex whose parent-edge is labeled *Minor*). Therefore, this election can easily be implemented by the extra searcher performing a DFS of T. Moreover, that can be done with $O(\log n)$ bits of memory, since the extra searcher only needs to remember the minimum $\ell evel\ell$ of a searcher satisfying one of the above conditions that it meets during this DFS.

Once the extra searcher has performed this DFS and has gone back to the root of T, let k be the minimum $\ell evel\ell$, satisfying one of the above conditions, it has met. Then, the extra searcher performs a new DFS to reach a searcher A with $\ell evel\ell = k$ satisfying one of the conditions. Let v be the vertex occupied by the searcher A. We now go further into the details of the four conditions listed above. In the following, when we refer to the root, we mean the current root of T, and thus, of S.

Case a. $m_v + t_v \leq 2$ and $c_v \geq 1$. This case is depicted in Figure 2.
In this case, v has degree at most two in T and is incident to at least one contaminated edge e. The purpose of this case is to clear the edge e. If this edge leads to a new vertex, then e is added to both T and S.
More precisely, the extra searcher leads an additional searcher B from the root to the vertex v during its second DFS. The searcher B, followed by the extra searcher, clears e and reaches its other end $u \in V(G)$. If there is an other searcher at u, then

the extra searcher labels e with *Removed*, i.e., e is clear but does not belong to T. Then B and the extra searcher go back to the root. Otherwise, i.e., if u is a newly discovered vertex, the extra searcher labels e with *Minor*, i.e., e is added to both S and T. Then B remains at u to guard it and takes $\ell evel\ell = k+1$. The extra searcher goes back to the root.

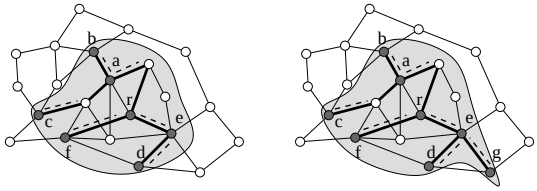

Fig. 2 Situation when Case a will be executed at vertex e (left). Situation after the execution of Case a (right).

In Figure 2, the light gray part represents the clear part of the graph, the dark gray vertices are those occupied by the searchers. The tree T is rooted in r and its edges are depicted in bold lines, and the dotted lines represent the edges of S. This case is applied to the vertex e. Note that the newly cleared edge could have been chosen incident to f as well (that is, this case could have been applied to f because f and e have the same level).

Case b. $m_v + t_v = 1$ and $c_v = 0$. This case is depicted in Figure 3.

In this case, v is a leaf in T and S, and is incident to no contaminated edge. In other words, all edges incident to v are labeled *Removed* except one edge, say e, that is labeled *Minor* or *Tree*. (The latter case occurs only if v is the root.) It means that the searcher occupying v does not protect the clear part of the graph from recontamination and thus, either it can go back to the root, or it can move away with the root, if v is the root. Moreover, recall that we want the tree T to be a small subtree spanning the vertices occupied by searchers. Thus, e can be pruned from T. For this purpose, the edge e is relabeled *Removed*. However, the tree T may be pruned more. Indeed, let u be the end of e different from v and assume that, before relabeling e, we had $m_u + t_u = 2$, $c_u = 0$, and u was not occupied by any searcher. Then after relabeling e, the vertex u is a leaf of T that is of no use to cover the vertices occupied by the searchers. Therefore this leaf can be pruned as well. In this case, the edge of T incident to u and different from e is pruned by being relabeled *Removed*. This process is executed recursively until a vertex w satisfying at least one of the following three conditions is reached: (1) the vertex is occupied, (2) the vertex is not of degree 2 in T, (3) the vertex is incident to at least one contaminated edge. (Note that the condition (1) is in fact sufficient. Indeed, we will prove later that each of (2) and (3) implies (1).) Finally, if v was the root, then w becomes the new root. The level of all searchers are updated if necessary.

In other words, if $P = (v, v_1, \cdots, v_r, w)$, $r \geq 0$, is the longest path of T such that, for any i, $1 \leq i \leq r$, v_i is not occupied by any searcher, v_i has degree two in T (i.e., $m_{v_i} + t_{v_i} = 2$), and v_i has no contaminated incident edge (i.e., $c_{v_i} = 0$), then all edges of P are relabeled *Removed*. This process corresponds, in T, to prune the branch containing v and, in S, to simply remove the leaf v.

More precisely, the pruning operation is performed in the following way: the searcher A occupying vertex v traverses the edge $e = \{v, v_1\}$ labeled *Minor* or *Tree*, relabeling it *Removed*. If v was the root, then v_1 becomes the new root, i.e., the booleans $root_v$ and $root_{v_1}$ are updated and all searchers occupying v go to v_1. Once e has been removed from T, if v_1 has degree one in T, is incident to no contaminated edge, and was not occupied before the removal of e, then the searcher A traverses $\{v_1, v_2\}$ relabeling it *Removed*. If v_1 was the root, then the root is moved to v_2, and all searchers that were occupying v_1 go to v_2. This process is done recursively while it is possible. Then, the extra searcher and searcher A go back to the root and takes $\ell evel\ell = 0$. Again, it is possible thanks to a DFS of T.

Finally, if, at the beginning of this phase, v was the root, then the level of any searcher that was not standing at v is decreased by one. (Their distance to the root in S has been decreased by one in the operation. Indeed we prove later that exactly one edge labeled *Minor* is pruned.) To do so, the extra searcher can perform a DFS of T.

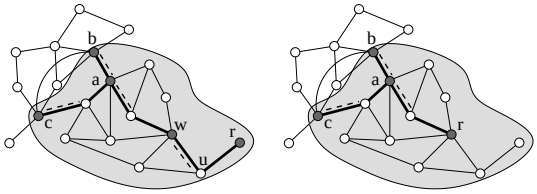

Fig. 3 Situation when Case b will be executed at vertex r (left). Situation after the execution of Case b (right).

In Figure 3, the light gray part represents the clear part of the graph, the dark gray vertices are those occupied by the searchers. The tree T is rooted in r and its edges are depicted in bold lines, and the dotted lines represent the edges of S. This case is applied to the leaf r. The path P is the path of T

between r and w. At the end of the process, w is the new root of S and T.

Case c. $m_v + t_v = 2$, $c_v = 0$ and v is not the root. This case is depicted in Figure 4.

In this case, v is not the root, has degree two in T, and is incident to no contaminated edge. Note first that the parent-edge e of v is labeled *Minor* because a vertex different from the root and occupied by a searcher always has its parent-edge labeled *Minor* (this will be proved by Claim 3 in the proof of Lemma 1). The purpose of this case is to remove v from S because the searcher A is not used to prevent recontamination and is not at a degree-3 vertex of T. This is done by sending A back to the root and by contracting the edge e in S. That corresponds to relabeling *Tree* the parent-edge e of v in T that was labeled *Minor*. Finally the level of some searchers are updated.

More precisely, the searcher A traverses the parent-edge e of v, labeling it *Tree*. Then, it goes back to the root and takes $\ell evel = 0$. Since this case corresponds to the contraction of e in S, we need to update, i.e., to decrease by one, the level of any searcher standing at a descendant of v. For this purpose, the extra searcher can perform a DFS of T_v, the subtree of T rooted at v. Finally, the extra searcher goes back to the root.

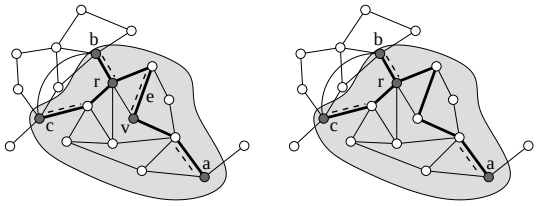

Fig. 4 Situation when Case c will be executed at vertex v (left). Situation after the execution of Case c (right).

In Figure 4, the light gray part represents the clear part of the graph, the dark gray vertices are those occupied by the searchers. The tree T is rooted in r and its edges are depicted in bold lines, and the dotted lines represent the edges of S. This case is applied to the vertex v. The searcher occupying v goes back to the root and the edge e is contracted (i.e., relabeled *Tree*). Moreover, the level of the searcher occupying a is decreased by one.

Case d. $m_v + t_v = 2$, $c_v = 0$ and v is the root. This case is depicted in Figure 5.

In this case, v is the root, has degree two in T, and is incident to no contaminated edge. Let e be the first edge labeled *Minor* to be traversed when performing some DFS traversal of T from v. Let u be the vertex such that e is its parent-edge. The purpose of this case is to move the root to u because the current root v is not used to prevent recontamination and is not a degree-3 vertex of v. This is done by moving the root to u and contracting the edge e in S. That corresponds to relabeling the edge e with *Tree*. Finally the level of some searchers are updated.

More precisely, all searchers standing at v (the root) perform an arbitrary but common DFS traversal of T until traversing an edge e labeled *Minor*. While traversing it, they relabel it with *Tree*. Let u be their current position. The vertex u becomes the new root, i.e., the booleans $root_v$ and $root_u$ are updated. Again, we need to update, i.e., to decrease by one, the level of any searcher that was standing at a descendant of v in the subtree containing u (when v was the root of T). This can be done by the extra searcher by a DFS traversal, as in the previous cases. Finally, the extra searcher goes back to the new root.

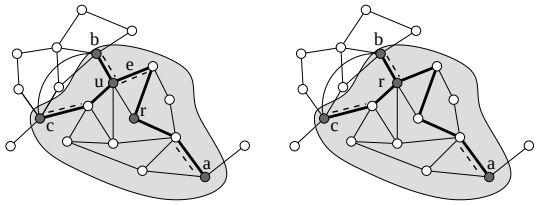

Fig. 5 Situation when Case d will be executed at vertex r (left). Possible situation after the execution of Case d (right).

In Figure 5, the light gray part represents the clear part of the graph, the dark gray vertices are those occupied by the searchers. The tree T is rooted in r and its edges are depicted in bold lines, and the dotted lines represent the edges of S. This case is applied to the root r that is moved to u. Note that the root could have been moved to a as well, depending on the order in which the arbitrary DFS of T is performed.

3.3 Correctness of Protocol `mc_search`

This section is devoted to prove the correctness of Protocol `mc_search`. For this purpose, we first prove the following technical lemma.

Lemma 1 *Let us consider the end of a phase of the execution of Protocol* `mc_search`*. Let T be the subgraph of G induced by the edges labeled Minor or Tree. Let S be the minor of T when all edges labeled Tree have been contracted.*

Initially all searchers stand at the root v_0 with $\ell evel\ell = 0$.
During the execution of `mc_search`, T is the tree that consists of edges labeled *Tree* or *Minor*. At the beginning, we have $T = (\{v_0\}, \emptyset)$.

Description of the execution of any phase of `mc_search`.

While there exists an edge labeled *Contaminated* **do**

1. Election of a searcher A occupying a vertex v, with minimum $\ell evel\ell$, say L, such that one of the four following conditions is satisfied.
   (Case a) $m_v + t_v \leq 2$, $c_v \geq 1$
   (Case b) $m_v + t_v = 1$, $c_v = 0$
   (Case c) $m_v + t_v = 2$, $c_v = 0$ and v is not the root
   (Case d) $m_v + t_v = 2$, $c_v = 0$ and v is the root

2. (Case a)
   An additional searcher B from the root goes to v.
   Let e be an edge incident to v and labeled *Contaminated*; B clears e.
   Let u be the other end of e.
   if u is occupied by another searcher **then**
   Label e *Removed*.
   Searcher B goes back to the root.
   else Label e *Minor*; Searcher B sets $\ell evel\ell = L + 1$ **endif**

   (Case b)
   Let e be the edge incident to v labeled *Minor* or *Tree*.
   Label e *Removed*.
   Let u be its other end.
   if v is the root **then** u becomes the new root; any searcher not standing at v decreases its level by one; all searchers standing at v go to u; **endif**
   (m_u and t_u have been updated when relabeling e)
   While $m_u + t_u = 1$, $c_u = 0$ and u was not occupied **do**
   Let f be the edge *Minor* or *Tree* incident to u.
   Label f *Removed*.
   Let u' be the other end of f. A goes to u'.
   if u is the root **then** u' becomes the new root and all searchers standing at u go to u' **endif**
   $u \leftarrow u'$ (again, m_u and t_u have been updated)
   EndWhile
   Searcher A goes to the root.

   (Case c)
   Let e be the parent-edge of v and let u be its other end.
   Label e with *Tree*.
   Let T_v be the subtree of T obtained by removing e and containing v.
   Any searcher occupying a vertex of T_v decreases its $\ell evel\ell$ by one.
   Searcher A goes to the root.

   (Case d)
   Let e be the first edge labeled *Minor* traversed when performing some DFS of T from v and let u be the vertex such that e is its parent-edge.
   Label e with *Tree*.
   Let T' be the subtree of T obtained by removing e and containing u. Any searcher occupying a vertex of T' decreases its $\ell evel\ell$ by one.
   u becomes the new root.
   All searchers that were standing at v go to u.

endWhile

Fig. 6 Protocol `mc_search`

1. *T and S are rooted trees with maximum degree at most three;*
2. *the set of vertices of G occupied by a searcher exactly consists of: the root, and the vertices whose parent-edge is labeled Minor;*
3. *if S has depth $k \geq 1$, then there exists a previous phase when S was the complete ternary tree T_{k-1}.*

Proof The proof is by induction on the phase number. Let $p \geq 1$ be the number of a phase of the execution of `mc_search` and let us assume that the result is valid at the beginning of phase p. Trivially this induction hypothesis holds when $p = 1$, since T ans S are restricted to the one-vertex tree that consists of the homebase where all searchers are standing. Let T' be the subgraph of G induced by the edges labeled *Minor* or *Tree* at the beginning of phase p, and let S' be the minor corresponding to the contraction of edges labeled *Tree*. T and S are the corresponding objects at the end of phase p. The proof of Lemma 1 proceeds in four claims. First we prove that S and T are trees.

Claim. 1 S and T are trees, and T has maximum degree at most three.

Proof. Note that, by definition, for any vertex $v \in V(G)$, $m_v + t_v$ is the degree of v in T'. According to the induction hypothesis, T' is a tree with maximum degree at most three. Let us show that at the end of phase p, T is a tree with maximum degree three. We consider the four cases a, b, c and d.

Case a. Either an edge $e = \{v, u\}$ is added to T', i.e., $T = (V(T') \cup \{u\}, E(T') \cup \{e\})$, or T' remains unchanged, i.e., $T = T'$. In the first case, we have $v \in V(T')$ and $u \notin V(T')$. Thus T is a tree in both cases. Moreover, $m_v + t_v$ was at most two, thus v has degree at most two in T'. Thus T has maximum degree at most three.

Case b. T is obtained from T' by recursively removing leaves of T'. Thus, T, as T', is a tree of maximum degree three.

Cases c and d. At most one edge of T' may be relabeled *Tree*, thus $T' = T$.

It follows that T is a tree with maximum degree at most three. Since S is obtained from T by edge contractions, S is also a tree. ◇

We now prove a structural property that will be used in the subsequent claims.

Claim. 2 Any vertex belonging to the tree T but not occupied by any searcher has degree exactly 2 in T.

Proof. First of all, when a vertex appears for the first time in T, it is occupied (initialization or case a). We

then note that when a searcher leaves a vertex unoccupied, either this vertex is removed from T (case b), or it is of degree 2 (cases c or d). Moreover, the degree in T of a vertex can only increase in case a, and thus only if this vertex is occupied. Thus the degree of an unoccupied vertex is at most 2.

Besides, the degree in T of a vertex can only decrease in case b, if it is the vertex w incident to the last pruned edge. Thus assume that the degree of w goes from 2 to 1 by application of case b at some vertex v. By definition of case b, the vertex w satisfies at least one of the following two conditions: (1) the vertex is occupied, (3) the vertex is incident to at least one contaminated edge. If w satisfies (3), then it is occupied, because our protocol maintains a searcher at any vertex belonging to the border of the clear part. Thus in both cases, w is occupied. As a consequence, the degree of an unoccupied node cannot be one.

This concludes the proof of the claim. ◇

Before proving that the maximum degree of S is three, we prove the second property.

Claim. 3 The set of vertices occupied by a searcher exactly consists of: the root, and the vertices whose parent-edge is labeled *Minor*.

Proof. We consider the four cases a, b, c and d. Let V'_M, resp. V_M, be the set of vertices such that their parent-edge are labeled *Minor* at the beginning, resp. end, of phase p.

Case a. The edge $e = \{v,u\}$, labeled *Contaminated* at the beginning of phase p, is the only edge to be relabeled. It is relabeled either *Removed* or *Minor*. In the first case, $V_M = V'_M$ and the searchers occupy exactly the same vertices than at the beginning of phase p, thus the property holds. In the second case, u is a new leaf of T (and S), and e is the parent edge of u. Thus $V_M = V'_M \cup \{u\}$. In both cases the vertices occupied by a searcher are exactly the root and the elements of V_M. Thus the property holds.

Case b. Let $P = (v, \cdots, w)$ be the path removed from the tree T at this phase. We first prove that w is occupied. By construction, the vertex w satisfies at least one of the following three conditions: (1) the vertex is occupied, (2) the vertex is not of degree 2 in T, (3) the vertex is incident to at least one contaminated edge. By Claim 2, we now that (2) implies (1). Moreover, as we already noticed before, if w satisfies (3), then it is occupied, because our protocol maintains a searcher at any vertex belonging to the border of the clear part. Thus, in any case w is occupied.

Assume first that v was the root at the beginning of the phase. This means that the only edge of P that was labeled *Minor* was the parent-edge of w, that is the edge incident to w and belonging to P. Thus, we have $V_M = V'_M \setminus \{w\}$. Moreover, w becomes the new root. On the other hand, the set of occupied vertices remains the same except for v that was occupied but is not anymore. Therefore, the vertices occupied by a searcher are exactly the root and the elements of V_M.

Assume now that v was not the root at the beginning of the phase. This means that the only edge of P that was labeled *Minor* was the edge incident to v, that is the parent-edge of v. Thus, we have $V_M = V'_M \setminus \{v\}$. On the other hand, the set of occupied vertices remains the same except for v that was occupied but is not anymore. Therefore, the vertices occupied by a searcher are exactly the root and the elements of V_M.

Thus the property holds in both subcases.

Case c. The parent-edge e of v is the only edge relabeled. According to the induction hypothesis, edge e is labeled *Minor* at the beginning of the phase because v is occupied by a searcher at this time. Hence, e is relabeled from *Minor* to *Tree*. Thus $V_M = V'_M \setminus \{v\}$. Since the searcher leaves v to go to the root, the property holds.

Case d. Let e be the edge relabeled in phase p. Let u be the vertex such that e is its parent-edge. The edge e is the only edge relabeled, and it is relabeled from *Minor* to *Tree*. Thus $V_M = V'_M \setminus \{u\}$. Moreover, all searchers from the old root go to the new root u. Finally, although the root changes, the child extremity of any edge other than e labeled *Minor* does not change. Thus the vertices occupied by a searcher are exactly $V_M \cup \{u\}$ $(= V'_M)$ and the property holds.

Therefore, at the end of phase p, the second property holds. ◇

We prove now that the maximum degree of S is at most three.

Claim. 4 S has maximum degree at most three.

Proof. By Claim 3, the child extremity v of any edge $\{u,v\}$ labeled *Tree* is not occupied by any searcher, and thus by Claim 2, it has degree exactly two. Thus, when this edge $\{u,v\}$ is contracted to obtain S from T, the vertex resulting from the fusion of u and v has the same degree as u had before the contraction. Therefore, S has maximum degree at most three, like T. ◇

To conclude the proof of the lemma, let us prove the third property.

Claim. 5 If S has depth $k \geq 1$, then there exists a previous phase when S was the complete ternary tree T_{k-1}.

Proof. First, for any searcher occupying a vertex v, its level is the number of edges labeled *Minor* between v and the root. This can be easily proved by induction. Let $k \geq 1$ and let us consider the first phase p' at which the depth of S becomes k. The phase p' consists of the clearing of a contaminated edge $e = \{u, v\}$ with $v \in V(T)$ occupied by a searcher with level $k-1$, and $u \in V(G) \setminus V(T)$. Since the move performed at phase p' is executed by a searcher with level $k-1$, it means that no searcher with level less than $k-1$ can move according to the rules. That is, all internal vertices of S have degree exactly three (because of cases c and d) and all leaves of S are at distance $k-1$ from the root (because of cases a and b), i.e., $S = T_{k-1}$. ◇

This concludes the proof of the lemma. □

We can now prove the main theorem.

Theorem 4 *Let G be a connected n-node graph and let v_0 be one of its vertices. Protocol* `mc_search` *enables $O(\frac{n}{\log n}\,\mathbf{mcs}(G, v_0))$ searchers to clear G in a monotone connected way, starting from v_0.*

Proof Let us first prove that the protocol `mc_search` clears G in a monotone connected way. Initially, all edges are labeled *Contaminated* and the label of an edge e becomes *Minor* or *Removed* as soon as e is traversed by a searcher. Moreover, after this traversal, each of its ends is occupied by a searcher (Case a). The strategy is obviously monotone since a searcher is removed from a vertex v if either v is occupied by an other searcher (Case a), or no contaminated edge is incident to v, i.e., $c_v = 0$, (Cases b, c and d). Furthermore, the strategy is connected since it is monotone and starts from a single vertex v_0. Finally, Protocol `mc_search` eventually clears G. Indeed, at each step, an edge is labeled, and any edge is relabeled at most three times: *Minor*, *Tree*, and *Removed* in this order. Thus, no loop can occur. Moreover, we proved above that T is a tree. Therefore, at any step, at least the searchers occupying its leaves satisfy the conditions of one of the cases a, b, c, or d. Thus, while there remains a contaminated edge, a searcher will eventually be called to clear this edge.

It remains to show that Protocol `mc_search` uses $q = O(\frac{n}{\log n}\,\mathbf{mcs}(G, v_0))$ searchers. Let us consider k to be the maximum depth of S during the clearing of G. By the three properties of Lemma 1 we have

$$q \leq |V(T_k)| = \frac{|V(T_k)|}{\log |V(T_k)|} \times \log |V(T_k)| \,.$$

Moreover, by the third property of Lemma 1, T_{k-1} is a minor of G, thus

$$\mathbf{s}(T_{k-1}) \leq \mathbf{s}(G) \leq \mathbf{mcs}(G, v_0) \text{ and } |V(T_{k-1})| \leq |V(G)|.$$

By Theorem 3, we have

$$\log |V(T_k)| = O(k) = O(\mathbf{mcs}(T_{k-1}))$$

and by Theorem 1,

$$\mathbf{mcs}(T_{k-1}) \leq 2\mathbf{s}(T_{k-1}).$$

Thus we have $\log |V(T_k)| = O(\mathbf{mcs}(G, v_0))$.

Finally, since the function $\frac{x}{\log x}$ is strictly increasing, and

$$|V(T_k)| = 3\,|V(T_{k-1})| + 1 \leq 3\,|V(G)| + 1 = 3n + 1 \,,$$

we obtain:

$$q = O\left(\frac{n}{\log n} \times \mathbf{mcs}(G, v_0)\right) \,,$$

which concludes the proof of the theorem. □

To conclude this section, let us estimate the number of moves done by the searchers during the execution of Protocol `mc_search`. As we already mentioned, at each phase, at least one edge is relabeled, and each edge is relabeled at most three times. This proves that there are $O(m)$ phases during the clearing of any graph with m edges. During any phase, one "move consuming" operation consists of the DFSs performed by the searchers. In the worst case, the extra searcher will execute a constant number of DFSs of T, while any other searcher will follow the extra searcher during at most one DFS. The other "move consuming" operation consists of the moves of all the searchers currently standing at the root that must follow the root when it is moved, leading to possibly $O(q \cdot n)$ moves in a single phase. This leads to an upper bound of $O(q \cdot n \cdot m)$ moves executed by the searchers to clear a n-node m-edge graph using q agents.

4 Further Work

It would be interesting to establish a tradeoff between the optimal competitive ratio of a search protocol and the amount of information provided to the searchers. Another difficult problem is to improve the competitive ratio of a search protocol by allowing the search strategy to be not monotone while it is still performed in a polynomial number of steps. Moreover, since the search problem assumes performing in an hostile environment, it would be interesting to design fault tolerant and/or self stabilizing algorithms for clearing a graph.

Acknowledgements David Ilcinkas received additional support from the ANR project *Aladdin* and the INRIA project *Cepage*. Nicolas Nisse received additional support from the CONICYT via the project *Anillo en Redes*, ACT08 and from the European projects 1ST FET AEOLUS.

References

1. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 200–209, 2002.
2. L. Barrière, P. Fraigniaud, N. Santoro, and D. M. Thilikos. Searching is not jumping. In *Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, number 2880 in LNCS, pages 34–45, 2003.
3. D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science*, 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, P. Fraigniaud, N. Nisse, and S. Vial. Distributed chasing of network intruders. *Theor. Comput. Sci.*, 399(1-2):12–37, 2008.
6. R. L. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, 6:72–78, 1967.
7. P. Flocchini, M. J. Huang, and F. L. Luccio. Decontaminating chordal rings and tori using mobile agents. *Int. J. Found. Comput. Sci.*, 18(3):547–563, 2007.
8. P. Flocchini, M. J. Huang, and F. L. Luccio. Decontamination of hypercubes by mobile agents. *Networks*, 52(3):167–178, 2008.
9. P. Flocchini, F. L. Luccio, and L. X. Song. Size optimal strategies for capturing an intruder in mesh networks. In *Proceedings of the 2005 International Conference on Communications in Computing (CIC)*, pages 200–206, 2005.
10. F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.
11. P. Fraigniaud and N. Nisse. Connected treewidth and connected graph searching. In *Proceedings of the 7th Latin American Symposium (LATIN)*, number 3887 in LNCS, pages 479–490, 2006.
12. P. Fraigniaud and N. Nisse. Monotony properties of connected visible graph searching. *Information and Computation*, 206(12):1383–1393, 2008.
13. A. S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993.
14. F. L. Luccio. Contiguous search problem in sierpinski graphs. *Theory of Computing Systems*, 44:186–204, 2009.
15. 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, 1988.
16. N. Nisse and D. Soguet. Graph searching with advice. *Theor. Comput. Sci.*, 410(14):1307–1318, 2009.
17. T. D. Parsons. The search number of a connected graph. In *Proceedings of the 9th Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1978)*, pages 549–554, 1978.
18. B. Yang, D. Dyer, and B. Alspach. Sweeping graphs with large clique number. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 908–920, 2004.