The cost of monotonicity in distributed graph searching

David Ilcinkas[1], Nicolas Nisse[2,⋆], and David Soguet[2]

[1] Université du Québec en Outaouais, Canada,
[2] LRI, Université Paris-Sud, France
{ilcinkas,nisse,soguet}@lri.fr

Abstract. Blin *et al.* (2006) proposed a distributed protocol that enables the smallest number of searchers to clear any unknown asynchronous graph in a decentralized manner. *Unknown* means that the searchers are provided no *a priori* information about the graph. However, the strategy that is actually performed lacks of an important property, namely the monotonicity. That is, the clear part of the graph may decrease at some steps of the execution of the protocol. Actually, the protocol of Blin *et al.* is executed in exponential time. Nisse and Soguet (2007) proved that, in order to ensure the smallest number of searchers to clear any n-node graph in a monotone way, it is necessary and sufficient to provide $\Theta(n \log n)$ bits of information to the searchers by putting short labels on the nodes of the graph. This paper deals with the smallest number of searchers that are necessary and sufficient to monotoneously clear any graph in a decentralized manner, when the searchers have no a priori information about the graph.

The distributed graph searching problem considers a team of searchers that is aiming at clearing any connected contaminated graph. 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. The *search number* $\mathbf{mcs}(G)$ of a graph G is the smallest number of searchers necessary to clear G in a monotone connected way in centralized settings. We prove that any distributed protocol aiming at clearing any unknown n-node graph in a monotone connected way, in decentralized settings, has *competitive ratio* $\Theta(\frac{n}{\log n})$. That is, we prove that, for any distributed protocol $\mathcal{P}$, there exists a constant c such that for any sufficiently large n, there exists a n-node graph G such that $\mathcal{P}$ requires at least $c\frac{n}{\log n}\,\mathbf{mcs}(G)$ searchers to clear G. Moreover, we propose a distributed protocol that allows $O(\frac{n}{\log n})\,\mathbf{mcs}(G)$ searchers to clear any unknown asynchronous n-node graph G in a monotone connected way.

Key words: Graph searching, Monotonicity, Competitive ratio.

⋆ Additional supports from the project Fragile of the ACI Sécurité Informatique, and from the project Grand Large of INRIA.

1 Introduction

In graph searching [6, 17], a team of *searchers* is aiming at capturing an invisible arbitrarily fast fugitive hidden in a graph (see [3] for a survey). Equivalently, an undirected connected graph is thought as a system of tunnels contaminated by a toxic gas. In this setting, the searchers are aiming at clearing the graph. The *search problem* has been widely studied in the design of distributed protocols for clearing a network in a decentralized manner [5, 7–9, 16]. Initially, all edges are contaminated. The searchers stand at the 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, initially, all the searchers are placed at a particular vertex of the graph, called the *homebase*. Moreover, this sequence of moves must satisfy 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 number of searchers to clear G starting from v_0. The search strategy must be computed online by the searchers themselves.

Note that, by definition, a search strategy satisfies two important properties. First, 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 in polynomial time. 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 proposed 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 $v_0 \in V(G)$, 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 be performed in exponential time, 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 $\mathbf{mcs}(G, v_0) + 1$ searchers to monotoneously clear a graph G, starting from a homebase v_0, such that the searchers are given some *a priori* information about it. In this paper, we consider the problem from another point of view. More precisely, we address the problem of the minimum number of searchers permitting 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

The searchers are modeled by synchronous autonomous mobile computing entities with distinct IDs. A network is modeled by a synchronous undirected connected simple graph. The network is anonymous, that is, the nodes are not labelled. The $\deg(u)$ edges incident to any node u are labelled 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. It is moreover assumed that searchers can access these whiteboards in fair mutual exclusion.

A *search protocol* $\mathcal{P}$ is a distributed protocol that solves the search problem, i.e., for any connected graph G and any homebase $v_0 \in V(G)$, a team of searchers executing $\mathcal{P}$ can clear G in a connected monotone way, starting from v_0. In these settings, the searchers do not know in advance in which graph they are launched. The number of searchers used by $\mathcal{P}$ to clear G is the maximum number of searchers that stand at the vertices of G over all steps of the execution of $\mathcal{P}$. The quality of a search protocol $\mathcal{P}$ is measured by comparing the number of searchers it used to clear a graph G to 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* $r(\mathcal{P})$ of the protocol $\mathcal{P}$.

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 a search protocol that has competitive ratio $O(\frac{n}{\log n})$. More precisely, we prove that for any distributed protocol $\mathcal{P}$, 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 $\mathcal{P}$ requires at least $c\frac{n}{\log n}\,\mathbf{mcs}(G, v_0)$ searchers to clear G, starting from v_0. On the other hand, we propose a 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 connected monotone 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.

1.3 Related work

In connected graph searching [1, 2, 10], 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 the communications between the searchers to be secured. Contrary to the classical, i.e., non-connected, graph searching [4, 13, 17], the monotonicity has a cost in terms of number of searchers. Indeed, Alspash *et al.* proved that *recontamination does help* in the case of connected graph searching [18] (see also [11]). That is, they describe a class of graphs for which the smallest number of searchers required to 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 in decentralized settings since, first, they perform in polynomial time, 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 tori [7] and meshes [8], Flocchini, Huang and Luccio considered hypercubes [9], and Luccio dealt with Sierpinski's graphs [14]. Assuming the searchers know the topology of the graph 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, compared to the centralized case, is necessary and due to the asynchrony of the network [8]. In [5], Blin *et al.* proposed a distributed protocol that allows $\mathbf{mcs}(G, v_0) + 1$ 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 that is actually performed is not monotone and may be performed in exponential time. 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, using $\mathbf{mcs}(G, v_0) + 1$ searchers and starting from a homebase v_0.

2 Lower Bound

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, 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 number of agents 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 concerning 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). 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 whose internal vertices have 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 that results in clearing the whole graph. $\mathbf{s}(G)$ denotes the smallest number of searchers that are necessary to clear a graph G in such a way. The class of trees has particularly been studied regarding graph searching. In particular, the following results have been proved.

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])**
For any $v_0 \in V(T)$, $\mathbf{mcs}(T, v_0) \leq 2\mathbf{s}(T) - 1$ **(Barrière** *et al.* **[2])**

The remaining part of this section is devoted to the proof of Theorem 2.

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

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

Let $n \geq 5$. We consider an unknown ternary tree T, that $\mathcal{P}$ has to clear starting from $v_0 \in V(T)$. Let us describe the game executed turn by turn by $\mathcal{P}$ and the adversary $\mathcal{A}$. 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, $\mathcal{P}$ and $\mathcal{A}$ play alternatively, starting with $\mathcal{P}$. At each round, $T_p = (V, H, F)$ corresponds to the part of T that $\mathcal{P}$ currently knows. $\mathcal{P}$ chooses a searcher and it moves this searcher along an edge e of T_p if it does not imply recontamination. Such a move is always possible since $\mathcal{P}$ is a search protocol, and thus, it eventually clears T. Note that e may be a half-edge or a full-edge. If e is a full-edge, then $\mathcal{A}$ 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, $\mathcal{A}$ 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 latter case, $\mathcal{A}$ adds a new end v to e such that v is incident to only one new half-edge f. Again, this is possible since $\mathcal{P}$ does not know the graph in advance. The game ends when $|V(T_p^+)| = n$. At such a round, $\mathcal{A}$ decides that the graph T is actually T_p^+.

Let us consider the last round, that is when $|V(T_p^+)| = n$. We show that at this round the number k of vertices of T_p^+ occupied by searchers is at least $k \geq n/4$. 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 $(n+2)/2$ leaves (this can be easily prove by induction on the number of rounds). Moreover, T_p^- is exactly the clear part of T at this step of the execution of $\mathcal{P}$. 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 $\mathcal{P}$ ensures that the strategy performed 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. From the previous remarks, it follows that T_p^+ is a ternary tree with at least $(n+2)/4$ vertices occupied by a searcher. Indeed, every parent of a leaf in T_p^+ must be occupied by a searcher, and every node is parent of at most two leaves. Thus, $\mathcal{P}$ uses at least $k \geq n/4$ searchers. By Theorem 1, $\mathbf{mcs}(T, v_0) \leq 2(1 + \log_3(n-1))$. Therefore,

$$k \geq \mathbf{mcs}(T, v_0) \times \frac{n}{8(1 + \log_3(n-1))} .$$

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

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

which concludes the proof the theorem. □

3 Upper Bound

In this section, we propose a search protocol `mc_search` (for monotone connected search) with competitive ratio $O(\frac{n}{\log n})$ for any n-node graph. Combining with the lower bound proved in section 2, it shows that $\Theta(\frac{n}{\log n} \, \mathbf{mcs}(G, v_0))$ searchers are necessary and sufficient to clear any unknown n-node graph G in a monotone connected way, starting from any homebase v_0 and in decentralized settings.

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 $v \in V(T)$ that is not the root. Let u be the parent of v, then 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.

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

3.1 Idea of protocol `mc_search`

Let us roughly describe the search protocol `mc_search`. Let G be a connected n-node graph and $v_0 \in V(G)$. The main issue of `mc_search` is to maintain two

* The *contraction* of the edge e with endpoints u, v is the replacement of u and v with a single vertex whose incident edges are the edges other than e that were incident to u or v.

dynamic rooted trees T and S. At each step, T is a subtree of the clear part of G, and S is a minor of T with same root. Intuitively, S represents the current positions of the searchers in G, and T enables the searchers to move in the clear part of the graph by performing a DFS of T. Initially, $S = T = \{v_0\}$ and all searchers are at v_0.

Roughly speaking, at each step, Protocol `mc_search` tries to clear an edge of G that is chosen such that S becomes as close as possible to a complete ternary tree. If the chosen edge e reaches a new vertex, i.e., a vertex that is not occupied by a searcher yet, e is added to S and labelled $Minor$. Otherwise, e is labelled *Removed*, meaning that e has been cleared but it does not belong to S nor T.

At some step of the execution of Protocol `mc_search`, it might happen that some vertices of S are not "useful" to let S be the densest possible ternary tree. Such vertices are those vertices of S with degree two or less in S, and whose all incident edges (in G) have been cleared. Let v be such a vertex and e its parent-edge. Protocol `mc_search` is aiming at "contracting" e. There are two cases according whether v is a leaf of S or not. In the first case, e is labelled *Removed*. In the latter case, e will be used by the searchers to circulate between the different components of S in G. For this purpose, e is labelled $Tree$. As a consequence, edges labelled $Minor$ and $Tree$ induce a tree T that enables the searchers to circulate in the clear part of G, by performing a DFS. Especially, T enables the searchers to reach all vertices of S.

We will show in the next sections that Protocol `mc_search` eventually clears G in a monotone connected way, starting from v_0, and using $N > 0$ searchers. Moreover, `mc_search` organizes the moves of the searchers in such a way that the following three properties are satisfied at any step. These three properties enable to show that $N = O(\frac{n}{\log n} \times \mathbf{mcs}(G, v_0))$.

1. T and S have maximum degree three,
2. the vertex-set of S is the set of vertices of G occupied by a searcher at this step, and
3. S has depth $k \geq 1$ only if there exists a previous step when S was the complete ternary tree T_{k-1}.

Let us consider k to be the maximum depth of S during the clearing of G. By properties **1,2** and **3**,

$$N \leq |V(T_k)| = \tfrac{|V(T_k)|}{\log |V(T_k)|} \times \log |V(T_k)|.$$

Moreover, by property **3**, T_{k-1} is minor of G, thus $\mathbf{s}(T_{k-1}) \leq \mathbf{s}(G) \leq \mathbf{mcs}(G, v_0)$ and $|V(T_{k-1})| \leq 2|V(G)|$. By Theorems 1 and 3, $\log |V(T_k)| = O(k) = O(\mathbf{mcs}(T_{k-1})) \leq O(\mathbf{s}(T_{k-1})) \leq O(\mathbf{s}(G)) \leq O(\mathbf{mcs}(G, v_0))$. Finally, since the function $\frac{x}{\log x}$ is strictly increasing, and $|V(T_k)| = 3\,|V(T_{k-1})| + 1 \leq 3\,|V(G)| + 1 = 3\,n + 1$, we obtain:

$$N = O(\tfrac{n}{\log n} \times \mathbf{mcs}(G, v_0)).$$

3.2 Protocol `mc_search`

In this section, we describe the main features of protocol `mc_search` that is described in Figure 1. For the purpose of simplifying the presentation, we assume in this figure that searchers are able to communicate by exchanging messages of size $O(\log n)$ bits. This assumption can be implemented by an additional searcher. This extra searcher will be 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. First, we describe the data structure used by `mc_search`.

Every searcher has a state variable $\ell evel \in \{0, \cdots, n\}$. Roughly, this variable indicates the distance between the vertex currently occupied by the searcher and the root, in the tree S. Initially, any searcher has $\ell evel = 0$.

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 vertex v and any edge e, $status_v[e] =$ *Contaminated*. To simplify the presentation, we assume that each edge $e = \{u, v\} \in E(G)$ has only one label $\ell(e) = status_v[e] = status_u[e] \in L$. This also may be implemented by the extra searcher. Moreover the whiteboard of every vertex v contains a boolean $root_v$ which is either true if v is the current root of S or false.

The protocol is divided in $O(|E(G)|)$ phases. At each phase, at least an edge is relabelled. Note that any edge labelled *Contaminated* (resp., $Minor$, resp., $Tree$) can be labelled $Minor$ or *Removed* (resp., $Tree$ or *Removed*, resp., *Removed*). The edges labelled *Removed* are not relabelled, which proves that Protocol `mc_search` terminates.

Let us define some notations. At any step, T is the subgraph of G induced by the edges labelled $Minor$ 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 labelled $Tree$. Initially, T is rooted at v_0. 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 labelled $Minor$, $Tree$, *Removed*, *Contaminated*.

Let us describe a phase of the execution of Protocol `mc_search`. A phase starts by the election of the searcher that will perform the move or the labelling of an edge. The elected searcher is an arbitrary searcher with minimum $\ell evel$ and that occupies a vertex $v \in V(G)$ satisfying one of the following four conditions, that we detail below. **Case a:** $t_v + m_v \leq 2$ and $c_v \geq 1$, **Case b:** $m_v = 1$, $t_v = 0$ and $c_v = 0$, **Case c:** $m_v + t_v = 2$, $m_v > 0$, $c_v = 0$ and v is not the root, **Case d:** $m_v + t_v = 2$, $c_v = 0$ and v is the root. We prove below that, while the graph is not clear, at least one vertex occupied by a searcher satisfies one of these conditions.

We will prove that, at any phase, any searcher actually occupies a vertex of S. Therefore, this election can easily be implemented by the extra searcher performing a DFS of T. Moreover, that can be done with $O(\log n)$ bit of memory, since the extra searcher only needs to remember the minimum $\ell evel$ 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, let k be the minimum $\ell evel$ satisfying one of the conditions, it has met. Then, the extra searcher performs a new DFS to reach a searcher A with $\ell evel = k$ at a vertex $v \in V(G)$ satisfying one of the conditions. We consider the four cases.

Case a. $t_v + m_v \leq 2$ and $c_v \geq 1$. That is, v has degree at most two in T and it is incident to a contaminated edge e. This case is aiming at adding an edge to T and S for letting S to be as close as possible to a complete ternary tree. In this case, the extra searcher has led another searcher B from the root to v during its second DFS. The searcher B, followed by the extra searcher, clears e and reaches its other end $u \in V(G)$. Either there is an other searcher at u, i.e., u belongs to S, or not, i.e., $u \notin V(T)$. In the first case, the extra searcher labels e with *Removed*, i.e. e is clear but it does not belong to T. Then, B and the extra searcher goes back to the root. In the second case, the extra searcher labels e with $Minor$, i.e. e is added to S and T. Then, B remains at u to guard it, and B takes $\ell evel = k + 1$.

Case b. $m_v = 1$, $t_v = 0$ and $c_v = 0$. That is, v has degree one in T and S, and it is incident to no contaminated edge. This case is aiming at removing a leaf from S and T, because no other edge incident to this vertex might be added to T. This corresponds to relabelling *Removed* the edge e incident to v in S that was labelled $Minor$. Moreover, let P be the maximal-inclusion path in T, such that v is an end of P, all edges of P are labelled $Tree$ and all internal vertices in P have degree two in T, then all these edges are relabelled *Removed*, which corresponds to removing all the vertices of P from T.
In this case, searcher A traverses the edge e labelled $Minor$, labelling it *Removed*. Let u be the other end of e. Once e has been removed from T, if u has degree one in T and its incident edge f in T has label $Tree$, f is removed in a similar way. This process is done recursively while it is possible. Note that u cannot be incident to a contaminated edge, otherwise, the protocol ensures that another searcher with $\ell evel < k$ would have stand at u. To conclude this case, the extra searcher and searcher A go back to the root and takes $\ell evel = 0$. Again, it is possible thanks to a DFS of T.

Case c. $m_v + t_v = 2$, $m_v > 0$, $c_v = 0$ and v is not the root. That is, v has degree two in T and at least one in S and it is incident to no contaminated edge. This case is aiming at contracting an edge e in S. That corresponds to relabelling $Tree$ an edge incident to v in S that was labelled $Minor$. We prove that the parent-edge of such vertex is actually labelled $Minor$.
In this case, searcher A traverses the edge e labelled $Minor$, labelling it $Tree$. Then, searcher A goes back to the root and takes $\ell evel = 0$. Since, this case correspond 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 in v. Finally, the extra searcher goes back to the root.

Initially all searchers stand at v_0 with $\ell evel = 0$. $T = (v_0, \emptyset)$ with v_0 as root.
During the execution of `mc_search`, T is the tree that consists of edges labelled $Tree$ or $Minor$.

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

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

1. Election of a searcher A occupying a vertex v, with minimum $\ell evel$, say L, such that one of the four following cases is satisfied.
   (Case a) $t_v + m_v \leq 2$, $c_v \geq 1$
   (Case b) $m_v = 1$, $t_v = 0$, $c_v = 0$
   (Case c) $m_v + t_v = 2$, $m_v > 0$, $c_v = 0$ and v is not the root
   (Case d) $m_v + t_v = 2$, $c_v = 0$ and v is the root
2. **(Case a)**
   A searcher B standing at the root is called and goes to v.
   Let e be an edge incident to v and labelled *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 to the root;
   else Label e $Minor$; Searcher B takes $\ell evel = L + 1$; **endif**

   (Case b)
   Let e be the edge incident to v labelled $Minor$.
   Label e *Removed* and let u its other end;
   if v is the root **then** u becomes the new root;
   all searchers standing at v go to u; **endif**
   While $m_u = 0$, $t_u = 1$, $c_u = 0$ **do**
   Let f be the edge incident to u labelled $Tree$.
   Label f *Removed*; Let u' the other end of f and 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**
   $u \leftarrow u'$;
   EndWhile
   Searcher A goes to the root;

   (Case c)
   Let e be the parent-edge of v and u 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 $\ell evel$ by one;
   Searcher A goes to the root;

   (Case d)
   Let e be an edge that is closest to v in T such that e is labelled $Minor$;
   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 that does not contain v;
   Any searcher occupying a vertex of T' decreases its $\ell evel$ by one;
   u becomes the new root;
   All searchers that were standing at v go to u;

endWhile

Fig. 1. Protocol `mc_search`

Case d. $m_v + t_v = 2$, $c_v = 0$ and v is the root. That is, v has degree two in T and it is incident to no contaminated edge. This case is aiming at contracting an edge in S. There are two cases according whether v is incident to an edge labelled $Minor$, or not. If v is incident to an edge labelled $Minor$, let e be this edge. Otherwise, let w be the vertex that is one of the two vertices closest to v in T and such that $m_w > 0$, let e be the edge labelled $Minor$ incident to w, and let u be the other end of e. Note that we will prove that such a vertex w has degree two in T and is incident to exactly one edge labelled $Minor$. This case is aiming at contracting the edge e in S. That corresponds to relabelling the edge e with $Tree$. This case also modifies the position of the root.
In this case, all searchers standing at v (the root) are aiming at traversing the edge e and at labelling it $Tree$. If e is incident to v, it can easily be done. Otherwise, the searchers choose one of the two edges incident to v and traverse all edges labelled $Tree$ that they meet until reaching a vertex incident to an edge labelled $Minor$, i.e., the vertex w. Then, they traverse $e = \{w, u\}$ and relabelled it $Tree$. In both cases, the searchers reach the vertex u that 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 standing at a descendant of v in the subtree containing u. This can be done by the extra searcher as in the previous case. Finally, the extra searcher goes back to the new root.

3.3 Correctness of Protocol `mc_search`

This section is devoted to prove the following theorem.

Theorem 4. *Let G be a connected n-node graph and $v_0 \in V(G)$. 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. The difficult part of the proof consists in showing that not too many searchers are used. Thus, let us first prove that Protocol `mc_search` clears G in a monotone connected way. Initially, all edges are labelled *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). Therefore, the strategy is monotone and 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 labelled, and any edge is relabelled at most three times: $Minor, Tree$, and *Removed* in this order. Thus, no loop can occur. Moreover, we prove below that T is a tree. Therefore, at any step, at least the searchers occupying its leaves satisfy the conditions 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.

The remaining part of the section is devoted to prove that Protocol `mc_search` uses at most $O(\frac{n}{\log n}\,\mathbf{mcs}(G,v_0))$ searchers. For this purpose, it is sufficient to prove the three properties described in section 3.1. More precisely, we prove the following lemma.

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

1. *T and S are rooted trees with maximum degree at most three,*
2. *the vertex-set of S is the set of vertices of G occupied by a searcher at this phase, and*
3. *S has depth $k \geq 1$ only if there exists a previous step when S was the complete ternary tree T_{k-1}.*

The proof is by induction on the number of phases of the execution of Protocol `mc_search`. Initially, the result is obviously valid. Let $p > 0$ be a phase of the execution of `mc_search` and let us assume that the result is valid for any previous phase. Let T' be the subgraph of G induced by the edges labelled $Minor$ or $Tree$ after phase $p-1$, and S' the minor corresponding to the contraction of edges labelled $Tree$.

First we prove that S and T are acyclic. 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 v be a vertex incident to at least one edge labelled *Contaminated* and that is not occupied by a searcher. By monotonicity of the strategy, all edges incident to v are labelled *Contaminated*. Thus, such a vertex does not belong to T'. Let us show that after 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., $T = T'$. Since, $v \in V(T')$ and $u \notin V(T')$, T is a tree in both cases. Moreover, $m_v + t_v \leq 2$, thus v has degree at most two in T'. Thus T has maximum degree at most three.

Case b. $m_v + t_v = 1$, thus v is a leaf of T. Let $u' \in V(T')$ be the neighbor of v and $e = \{u', v\}$ that is labelled $Minor$. First e is relabelled *Removed*, thus v is removed from T'. Then, if u' is of degree one in $T' \setminus \{v\}$ and its incident edge f in $T' \setminus \{v\}$ is labelled $Tree$, f is relabelled *Removed*, i.e. u' is removed from $T' \setminus \{v\}$. This process is repeated recursively. Thus, T is a tree obtained from T' by recursively removing leaves of T'. Hence, the maximum degree of T is at most three.

Cases c and d. At most one edge of T' is relabelled $Tree$, thus $T' = T$. In the proof of the Claim (see above) we prove that exactly one edge of T' is relabelled $Tree$.

It follows that T is a tree with maximum degree at most three. Since S is a minor of T, S is a tree.

Before proving that the maximum degree of S is three, we prove the second property. We prove by induction on p that the vertices occupied by a searcher are exactly: the root, and those vertices the parent-edge of which is labelled $Minor$.

Initially, the result is obviously valid. Let $p > 0$ be a phase of the execution of `mc_search` and let us assume that the result is valid for any previous phase. We consider the four cases a, b, c and d. Let V'_M be the set of vertices such that their parent-edge are labelled $Minor$ after the phase $p-1$.

Case a. An edge $e = \{v, u\}$ labelled *Contaminated* is the only edge to be relabelled. It is relabelled either *Removed* or $Minor$. In the first case, $S = S'$ and the searchers occupy exactly the same vertices than after the phase $p-1$, thus the property holds. In the second case, u is a leaf of T, and e is the parent edge of u. Thus $S = (V(S') \cup \{u\}, E(S') \cup \{e\})$. Moreover the vertices occupied by a searcher are exactly $V(S') \cup \{u\}$. Thus the property holds.

Case b. Let $e = \{v, u\}$ be the edge adjacent to v labelled $Minor$. e is the only edge relabelled from $Minor$ to *Removed*. All the other relabelled edges are labelled from $Tree$ to *Removed*. Thus $V_M = V'_M \setminus \{v\}$. Indeed note that if the root changes, the parent-edge of each vertex in $V'_M \setminus \{v\}$ does not change. If the root does not change, then $S = (V(S') \setminus \{u\}, E(S') \setminus \{e\})$. Moreover the vertices occupied by a searcher are exactly $V(S)$ and the property holds. If the root changes to w, $S = (V_M \cup \{w\}, E(S') \setminus \{e\})$, the vertices occupied by a searcher are exactly $V(S)$ and the property holds.

Case c. The parent-edge e of the vertex v is the only edge relabelled, and according to induction hypothesis it is relabelled from $Minor$ to $Tree$. Thus $S = (V(S') \setminus \{v\}, E(S') \setminus \{e\})$. Moreover the vertices occupied by a searcher are exactly $V(S)$, thus the property holds.

Case d. Let e be an edge that is closest to v in T' such that e is labelled $Minor$. We will prove in the next proof that such an edge always exists. If this edge does not exist nothing happens and the property holds.

Let u be the vertex such that e is its parent-edge. The edge e is the only edge relabelled, it is relabelled from $Minor$ to $Tree$. Thus $V_M = V'_M \setminus \{u\}$. Indeed the root changes such that the parent-edge of each vertex in V_M does not change and u is the new root. The root changes to u, thus $S = (V_M \cup \{u\}, E(S') \setminus \{e\})$. Moreover the vertices occupied by a searcher are exactly $V_M \cup \{u\}$ and the property holds.

Thus, at phase p, the vertex-set of S is the set of vertices of G occupied by a searcher at this phase.

In order to prove that S has maximum degree at most three, we need the following claim:

Claim. Let $v \in V(T)$ incident to an edge e labelled $Tree$, and such that e is not its parent-edge. Let T_v be the subtree of T obtained by removing e from T and that does not contain v. There exists an edge $f = \{u, w\}$ labelled $Minor$, such that f is the parent edge of w, u has degree two in T, and the subtree P of T_v obtained by removing f from T_v and that contains u consists of a path of edges labelled $Tree$.

Obviously, T_v contains at least one edge labelled $Minor$ because all leaves of T are labelled $Minor$. Indeed, when a leaf is added to T, its incident edge is labelled $Minor$ (Case a) and, when a leaf and its incident edge e labelled $Minor$ are removed, the whole path of edges labelled $Tree$ at which e is attached are removed (Case b).

We now prove that, for any vertex $u \in V(T)$ that is not the root, such that all its incident edges in T are labelled $Tree$, u has degree two in T. Since we have proved that a leaf can only be incident to an edge labelled $Minor$, u has degree at least two in T. For purpose of contradiction, let us assume that u has degree three in T. Let us consider the phase of the execution of `mc_search` such that the last edge incident to u and labelled *Contaminated* has been relabelled. From this phase, the degree of u in T might only have decreased. It follows that this vertex cannot have satisfied conditions corresponding to Cases b,c, or d. Thus, u has never been the root otherwise it would still be the case. Moreover, the parent-edge of u has never been relabelled contradicting the fact that it is labelled $Tree$. Hence, such a vertex u has degree exactly two in T.

Let f be the edge labelled $Minor$ that is the closest to v in T_v. Let u be the end of f that is closest to v. Obviously, u is not the root and its parent-edge is labelled $Tree$. It only remains to prove that u has degree exactly two in T. Similarly to the previous paragraph, we assume, for purpose of contradiction, that u has degree three in T. Again, this leads to the fact that its parent-edge could not have been relabelled, a contradiction. Thus, u has degree two and it is incident to an edge labelled $Minor$ and another edge labelled $Tree$. Moreover, all internal vertices of the path between u and v have degree two in T and they are incident to edges labelled $Tree$. This concludes the proof of the Claim. $\diamond$

Now, let us prove that S has maximum degree at most three. According to the induction hypothesis, S' has maximum degree at most three. To prove that the maximum degree of S is at most three, the four cases a, b, c and d must be considered by taking into account the previous claim. Indeed using the Claim, we get that the degree in S of a node v is actually equal to $m_v + t_v$, i.e., its degree in T. The induction consists to prove that at the end of the phase p, for all node $v \in S$, $m_v + t_v \leq 3$ according to the case a, b, c and d. The formal proof is omitted due to lack of space and can be found in [12].

To conclude the proof of the lemma, let us prove the third property. First, for any searcher occupying a vertex v of S, its level is the distance between v and the root. 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 $u \in V(S)$ occupied by a searcher with level $k-1$, and $v \in V(G) \setminus V(T)$. Since the move performed at phase p is executed by the searcher with smallest level, it means that no searcher with level less than $k-1$ can move. That is, all internal vertices of S have degree three and S has depth $k-1$, i.e. $S = T_{k-1}$. This concludes the proof of the lemma and of the theorem. $\square$

References

1. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA), pages 200-209, 2002.
2. L. Barrière, P. Fraigniaud, N. Santoro, and D. Thilikos. Connected and Internal Graph Searching. In 29th Workshop on Graph Theoretic Concepts in Computer Science (WG), Springer-Verlag, LNCS 2880, 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, pages 33–49, 1991.
4. D. Bienstock and P. Seymour. Monotonicity in graph searching. Journal of Algorithms 12, pages 239-245, 1991.
5. L. Blin, P. Fraigniaud, N. Nisse and S. Vial. Distributing Chasing of Network Intruders. In 13th Colloquium on Structural Information and Communication Complexity (SIROCCO), Springer-Verlag, LNCS 4056, pages 70-84, 2006.
6. R. Breisch. An intuitive approach to speleotopology. Southwestern Cavers VI(5), pages 72-78, 1967
7. P. Flocchini, F.L. Luccio, and L. Song. Decontamination of chordal rings and tori. Proc. of 8th Workshop on Advances in Parallel and Distributed Computational Models (APDCM), 2006.
8. P. Flocchini, F.L. Luccio, and L. Song. Size Optimal Strategies for Capturing an Intruder in Mesh Networks. Proceedings of the 2005 International Conference on Communications in Computing (CIC), pages 200-206, 2005
9. P. Flocchini, M. J. Huang, F.L. Luccio. Contiguous search in the hypercube for capturing an intruder. Proc. of 18th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS), 2005.
10. P. Fraigniaud and N. Nisse. Connected Treewidth and Connected Graph Searching. In 7th Latin American Theoretical Informatics Symp. (LATIN), LNCS 3887, pages 470-490, 2006.
11. P. Fraigniaud and N. Nisse. Monotony properties of connected visible graph searching. In 32th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), LNCS 4271, pages229-240, 2006.
12. D. Ilcinkas, N. Nisse, and D. Soguet. The cost of monotonicity in distributed graph searching. Technical Report, LRI-1475, University Paris-Sud, France, Sept. 2007.
13. A. LaPaugh. Recontamination does not help to search a graph. Journal of the ACM 40(2), pages 224-245, 1993.
14. F. L. Luccio Intruder capture in Sierpinski graphs. Proceedings of the 4th International Conference on Fun with algorithms (FUN), Springer-Verlag, LNCS 4475, pages 249-261, 2007.
15. N. Megiddo, S. Hakimi, M. Garey, D. Johnson and C. Papadimitriou. The complexity of searching a graph. Journal of the ACM 35(1), pages 18-44, 1988.
16. N. Nisse and D. Soguet. Graph searching with advice. In 14th Colloquium on Structural Information and Communication Complexity (SIROCCO), Springer-Verlag, LNCS 4474, pages 51-67, 2007.
17. T. Parson. Pursuit-evasion in a graph. Theory and Applications of Graphs, Lecture Notes in Mathematics, Springer-Verlag, pages 426-441, 1976.
18. B. Yang, D. Dyer, and B. Alspach. Sweeping Graphs with Large Clique Number. In 15th Annual International Symp. on Algorithms and Computation (ISAAC), pages 908-920, 2004.