Graph searching with advice

Nicolas Nisse * and David Soguet

LRI, Université Paris-Sud, Orsay, France
{nisse,soguet}@lri.fr

Abstract. Fraigniaud *et al.* (2006) introduced a new measure of difficulty for a distributed task in a network. The smallest *number of bits of advice* of a distributed problem is the smallest number of bits of information that has to be available to nodes in order to accomplish the task efficiently. Our paper deals with the number of bits of advice required to perform efficiently the graph searching problem in a distributed setting. In this variant of the problem, all searchers are initially placed at a particular node of the network. The aim of the team of searchers is to capture an invisible and arbitrarily fast fugitive in a monotone connected way, i.e., the cleared part of the graph is permanently connected, and never decreases while the search strategy is executed. We show that the minimum number of bits of advice permitting the monotone connected clearing of a network in a distributed setting is $O(n \log n)$, where n is the number of nodes of the network, and this bound is tight. More precisely, we first provide a labelling of the vertices of any graph G, using a total of $O(n \log n)$ bits, and a protocol using this labelling that enables clearing G in a monotone connected distributed way. Then, we show that this number of bits of advice is almost optimal: no protocol using an oracle providing $o(n \log n)$ bits of advice permits the monotone connected clearing of a network using the smallest number of searchers.

Keywords: Graph searching, Monotonicity, Bits of advice.

1 Introduction

The *search problem* has been widely used in the design of distributed protocols for clearing graphs in a decentralized manner [1, 6, 10, 11]. In the search problem, the graph is regarded as a "contaminated" network that a team of *searchers* is aiming at clearing. Initially, the whole graph is *contaminated*. The searchers stand at some vertices of the graph and they are allowed to move along edges. An edge is *cleared* when it is traversed by a searcher. A clear edge e is preserved from recontamination if, for any path between e and a contaminated edge, a searcher is occupying a vertex of this path. The search problem deals with a sequence of moves of searchers, that satisfies: (1) initially all searchers stand at a particular vertex of the graph, *the homebase*, and (2) a searcher is allowed to move along an edge if it does not imply any recontamination. Such a sequence

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

of moves, or *steps*, is called a *search strategy*. Given a connected graph G and a vertex $v_0 \in V(G)$, the search problem consists in computing, in a distributed setting, a search strategy of G, with v_0 as the homebase, and using the fewest searchers as possible that results in all edges being simultaneously clear. The strategy is computed online by the searchers themselves. Note that, during the execution of a search strategy, the contaminated part of the graph never grows. The strategy is said *monotone* [5, 15]. Moreover, the cleared part of the graph remains connected at any step. The strategy is said *connected* [1, 2].

The main difference between the existing distributed protocols for clearing a graph is the amount of knowledge about the topology of the graph that searchers have *a priori*. In [1, 10, 11], the searchers know in advance the topology of the network in which they are launched, and clear the network in a polynomial time. Conversely, the protocol provided in [6] enables to clear any network without having any *a priori* information about its topology. However, the clearing of the network is connected but not monotone and it is performed in an exponential time. Thus, not surprisingly, it appears that there is a tradeoff between the amount of knowledge provided to the searchers and the performance of the search strategy.

In [12], Fraigniaud *et al.* propose a new framework for measuring the difficulty of a distributed task: the number of *bits of advice*. Given a distributed task, the minimum number of bits of advice for this problem represents the minimum total number of bits of information that has to be given to nodes or mobile agents to efficiently perform the task. This approach is quantitative, i.e. it considers the amount of knowledge without regarding what kind of knowledge is supplied. This paper addresses the problem of the minimum number of bits of advice permitting to solve the search problem.

1.1 Our model

The searchers are modeled by synchronous autonomous mobile computing entities with distinct IDs. Otherwise searchers are all identical, run the same program and use at most $O(\log n)$ bits of memory, where n is the number of nodes of the network. A network is modeled by a synchronous undirected connected graph. *A priori*, 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, *whiteboard*, of size $O(\log n)$ bits in which searchers can read, erase, and write symbols. It is moreover assumed that searchers can access these whiteboards in fair mutual exclusion. An instance of the problem consists of a couple (G, v_0), where $G = (V, E)$ is a graph and $v_0 \in V$ is the homebase. An *oracle* [12, 13] is a function $\mathcal{O}$ that maps any instance (G, v_0) to a function $f : V \to \{0, 1\}^*$ assigning a binary string, called *advice*, to any node of the network. The size of the advice, i.e. the number of bits of advice, on a given instance, is the sum of the lengths of all the strings assigned to the nodes. Intuitively, the oracle provides additional knowledge to the nodes of the network.

The *search problem* consists in designing an oracle $\mathcal{O}$ and a protocol $\mathcal{P}$ using $\mathcal{O}$, with the following characteristics. For any instance $(G = (V, E), v_0)$, any vertex $v \in V$ is provided with the string $f(v)$, $f = \mathcal{O}(G, v_0)$. Protocol $\mathcal{P}$ must enable the optimal number of searchers to clear G starting from v_0. Moreover, the search strategy performed by searchers is computed locally. That is, the decision of the searcher at a vertex v (moving via some specific port number, switching its state, writing on the whiteboard) only depends on (1) the current state of the searcher, (2) the label $f(v)$ of the current vertex (3) the content of the current node's whiteboard (plus possibly the incoming port number if the searcher just entered the node). In particular, the searchers do not know in advance in which graph they are launched. The only information about the graph is the bit strings available locally at each node.

1.2 Our results

We show that the minimum number of bits of advice permitting the clearing of any n-node graph, in a distributed setting, is $O(n \log n)$, and this bound is tight. More precisely, on one hand, we define an oracle $\mathcal{O}$ and a distributed protocol `Cleaner` that allow to solve the search problem for any connected n-node graph G starting from any vertex $v_0 \in V(G)$. Moreover, the clearing of G is performed in time $O(n^3)$. The searchers are modeled by automata with $O(\log n)$ bits of memory. The nodes' whiteboards have size $O(\log n)$. Actually, our protocol ensures that the whiteboard will only be used in order to allow two searchers present at the same node to exchange their states and IDs. Finally, the number of bits of advice provided by $\mathcal{O}$ is $O(n \log n)$ for any n-node graph. On the other hand, we show that this number of bits of advice is almost optimal: no protocol using an oracle providing $o(n \log n)$ bits of advice permits to solve the search problem.

1.3 Related work

In many areas of distributed computing, the quality of algorithmic solutions for a given network problem often depends on the amount of knowledge given to the nodes of the network (see [9] for a survey). The comparison of two algorithms with knowledge appears however to be not obvious when they are provided with qualitatively different informations: upper bound on the size of the network [3], the entire topology of the network [8], etc. In [12], Fraigniaud *et al.* introduce the notion of bits of advice as a way to quantitatively measure the difficulty of a distributed task. As an example, Fraigniaud *et al.* [12] study the amount of knowledge that must be distributed on the vertices of the graph in order to perform broadcast and wake-up efficiently (i.e., using a minimum number of messages). They prove that the minimum number of bits of advice permitting to perform wake-up (resp., broadcast) with a linear number of messages is $\Theta(n \log n)$ (resp., $O(n)$) bits. This quantitatively differentiate the difficulty of broadcast and wake-up. Fraigniaud *et al.* [13] also study the minimum number of bits of advice that allows to efficiently explore a tree, i.e., with a better competitive ratio than a Depth First Search.

Introduced by Parson [18], *graph searching* looks for the smallest number of searchers required to clear a graph. However, in graph searching, the strategies are not constrained to be connected nor monotone (see [4] for a survey). The *search number* of the graph G, denoted $\mathbf{s}(G)$, is the minimum k such that there is a search strategy for G (not necessarily monotone nor connected) using at most k searchers that results in all edges being simultaneously clear. The graph searching problem has been extensively studied for its practical applications and for the close relationship between its several variants (edge-search, node-search, mixed-search [4]) and standard graph parameters like treewidth [19] and pathwidth [4]. The problem of finding the search number of a graph has been proved to be NP-hard [16]. According to the important Lapaugh's result [5, 15], "recontamination does not help". That is, for any graph G, there is a monotone search strategy for G using at most $\mathbf{s}(G)$ searchers. Monotonicity plays a crucial role in graph searching, since a monotone search strategy ensures a clearing of the graph in a polynomial number of steps. It implies that the graph searching problem is in NP. This result is not valid anymore, as soon as the search strategy is constrained to be connected [20]. Several practical applications (decontamination of polluted pipes [18], speleological rescue [7], network security...) require the search strategy to be connected to ensure safe communications between searchers. Barrière *et al.* [2] prove that, clearing a tree T in a connected way requires at most $2\,\mathbf{s}(T)-2$ searchers and that this bound is tight. The better bound known in the case of an arbitrary n-node graph G is $\mathbf{s}(G)(\log n + 1)$ [14].

Several protocols for clearing a network in distributed setting have been proposed in the literature. It has been proved that any distributed protocol clearing an asynchronous network in a monotone connected way requires at most one searcher more than in the synchronous case [11]. Moreover, this result remains valid even if the topology of the network is known in advance. In [6], Blin *et al.* proposed a distributed protocol that enables the optimal number of searchers to clear any network G in a fully decentralized manner. The strategy is computed online by the searchers themselves. The distributed computation must not require knowing the topology of the network in advance. Roughly speaking, their protocol ensures that searchers try every possible connected monotone partial search strategy. Thus, whilst the search strategy eventually computed by the searchers is monotone, failing search strategies investigated before lead to withdrawals, and therefore to recontamination. Flocchini *et al.* proposed protocols that address the graph searching problem in specific topologies (trees [1], hypercubes [11], tori and chordal rings [10], etc.). For each of these classes of graphs, the authors propose a protocol using the optimal number of searchers for clearing G in a monotone connected way with $O(\log n)$ bits of memory and whiteboards of $O(\log n)$ bits, that clears the graph in a polynomial time. Note that, encoding the entire topology requires $\Omega(n^2)$ bits.

2 Distributed search strategy using little information

This section is devoted to prove the following theorem.

Theorem 1. *The search problem can be solved using $O(n \log n)$ bits of advice.*

To prove it, we describe an oracle $\mathcal{O}$ which provides an advice of size $O(n \log n)$, and a distributed protocol `Cleaner` that solve the search problem in a synchronous decentralized manner. Protocol `Cleaner` is divided in n phases, each one being divided in two stages of $O(n^2)$ rounds. Thus, the clearing of G is performed in a time $O(n^3)$.

2.1 The Oracle

In this section, we describe the oracle $\mathcal{O}$. For any instance $(G = (V, E), v_0)$ of the search problem, we consider a strategy S that is solution of the problem. The function $f = \mathcal{O}(G, v_0)$ is defined from S. Roughly speaking, the bits of advice supplied by $\mathcal{O}$ enable searchers using protocol `Cleaner`, to clear the vertex-set in the same order as S. Moreover, they allow the searchers to circulate in the cleared part of the graph avoiding recontamination. Let us define some notations.

Let $n = |V|$ and $m = |E|$. The strategy S can be defined by the order in which S clears the edges. Let $(e_1, \cdots, e_m)$ be this order. An edge e_i is smaller than an edge e_j, denoted by $e_i \preceq e_j$, if $i \leq j$. S also induces an order on the vertices of G. For any $v, w \in V$, we say that v is smaller than w, denoted $v \preceq w$, if the first cleared edge incident to v is smaller than the first cleared edge incident to w. Let $(v_0, \cdots, v_{n-1})$ be this order, i.e., $v_i \preceq v_j$ if and only if $i \leq j$.

For any $0 \leq i \leq n-1$, let $f_i \in E$ be the first cleared edge incident to v_i. By definition, $f_0 = f_1 \prec f_2 \cdots \prec f_{n-1}$. For any $1 \leq i \leq n-1$, the *parent* of v_i, denoted by $parent(v_i)$, is defined as the neighbour v of v_i such that, $\{v, v_i\} = f_i$. Note that $parent(v_i) \prec v_i$, and for any neighbour w of v_i, $f_i = \{parent(v_i), v_i\} \preceq \{w, v_i\}$. Intuitively, for any $1 \leq i \leq n-1$, $f_i = \{parent(v_i), v_i\}$ is the edge by which a searcher has arrived to clear v_i. Conversely, the children of $v \in V$ are the vertices w such that $v = parent(w)$. For any $0 \leq i \leq n-1$, let T_i be the subgraph of G whose vertex-set is $\{v_0, \cdots, v_i\}$, and the edge-set is $\{f_1, \cdots, f_i\}$. For any $0 \leq i \leq n-1$, T_i is a spanning tree of $G[v_0, \cdots, v_i]$, which denotes the subgraph of G induced by $\{v_0, \cdots, v_i\}$. Intuitively, at the phase i of the execution of Protocol `Cleaner`, T_{i-1} is a spanning tree of the clear part of the graph. It is used to allow the searchers to move in the clear part, performing a DFS of T_{i-1}.

We now define a local labelling $\mathcal{L}(S)$ of the vertices of G. Again, this labelling depends on the strategy S that is considered. Let $v \in V(G)$. The label of a vertex v consists of the following local variables: a boolean $\texttt{TYPE}_v$, four integers $\texttt{TCU}_v$, $\texttt{TTL}_v$, $\texttt{LASTPORT}_v$, $\texttt{PARENT}_v$ and a list $\texttt{CHILD}_v$ of ordered pairs of integers. The index will be omitted whenever this omission does not cause any confusion. Intuitively, $\texttt{PARENT}_v$ and $\texttt{CHILD}_v$ enable the searchers to perform a DFS of a subtree spanning the cleared part. To avoid recontamination, the searchers must know which ports they can take or not, and the moment when such a move is possible, i.e. the phase of the protocol when a searcher can take some port. The informations about the ports are carried by $\texttt{PARENT}_v$, $\texttt{CHILD}_v$, and $\texttt{LASTPORT}_v$. $\texttt{CHILD}_v$, $\texttt{TCU}_v$ and $\texttt{TTL}_v$ carry information about phases. Moreover, if a searcher preserves a node from recontamination, we say that this searcher *guards* the node, otherwise the searcher is said *free*. A searcher which guards a node v will leave v by its largest edge. Such a move will not induce any recontamination because any other edges incident to v will have been previously cleared by free searchers. For this task, we need to distinguish two types of node with $\texttt{TYPE}_v$.

In the following we will say that a port number p of a vertex v (resp., the edge incident to v, corresponding to p) is *labelled* if either there exists $\ell \leq n-1$ such that $(p,\ell) \in \texttt{CHILD}_v$, or $p = \texttt{LASTPORT}_v$, or $p = \texttt{PARENT}_v$. Note that an edge may have two different labels, or may be unlabelled at one of its ends, and labelled at the other, or unlabelled at both ends. Let $0 \leq i \leq n-1$ be the integer such that $v = v_i$. Let e be the largest edge incident to v that is not in $E(T_{n-1})$, and let f be the largest edge incident to v that is not in $(T_{n-1}) \cup \{e\}$.

- $\texttt{PARENT}_v$ is the port number of v leading to *parent*(v) through an edge of $E(T_{n-1})$ (we set $\texttt{PARENT}_{v_0} = -1$).
- $\texttt{CHILD}_v$ is a list of ordered pairs of integers. Let $1 \leq p \leq \deg(v)$ and $0 < j \leq n-1$. $(p,j) \in \texttt{CHILD}_v$ if and only if $v = parent(v_j)$ and p is the port number of v leading to v_j. In the following, $\texttt{CHILD}_v(j)$ denotes the port number p of v such that $(p,j) \in \texttt{CHILD}_v$.
- $\texttt{TYPE}_v$ is a boolean variable. It equals 0 if the largest edge incident to v belongs to T_{n-1}. Otherwise, the variable $\texttt{TYPE}_v$ equals 1. In the following we will say that a vertex is of type 0 (resp., type 1) if $\texttt{TYPE}_v = 0$ (resp., $\texttt{TYPE}_v = 1$). Roughly, a vertex is of type 0 if, in S, the searcher cleared the last uncleared incident edge to v, in order to reach a new vertex which was still uncleared.
  the last incident edge to reach a new vertex that was not occupied yet.
- $\texttt{LASTPORT}_v = -1$ if $\texttt{TYPE}_v = 0$, else $\texttt{LASTPORT}_v$ is the port number corresponding to e.
- $\texttt{TCU}_v$ (*Time to Clean Unlabelled port*), represents the phase when the free searchers must clear all the unlabelled ports of v. Case $\texttt{TYPE}_v = 0$: if e does not exist, then $\texttt{TCU}_v = -1$, else $\texttt{TCU}_v$ is the largest k such that $f_{k-1} \preceq e$. Case $\texttt{TYPE}_v = 1$: if f does not exist, then $\texttt{TCU}_v = -1$, else $\texttt{TCU}_v$ is the largest k such that $f_{k-1} \preceq f$.
- $\texttt{TTL}_v$ (*Time To Leave*), represents the phase when, a searcher that guards v will leave v. Case $\texttt{TYPE}_v = 0$: $\texttt{TTL}_v = j$ such that v_j is the largest child of v. Case $\texttt{TYPE}_v = 1$: $\texttt{TTL}_v$ is the largest k such that $f_{k-1} \leq e$.

We now define the bits of advice $\mathcal{O}(G,v_0)$ provided by oracle $\mathcal{O}$ to G, using the labelling $\mathcal{L}(S)$. For any $0 \leq i \leq n-1$,
$\mathcal{O}(G,v_0)(v_i) = (i, n, \texttt{TYPE}_{v_i}, \texttt{PARENT}_{v_i}, \texttt{LASTPORT}_{v_i}, \texttt{TCU}_{v_i}, \texttt{TTL}_{v_i}, \texttt{CHILD}_{v_i})$.

The following lemma follows obviously from the definition of the oracle.

Lemma 1. *For any n-node graph, $\mathcal{O}$ provides $O(n \log n)$ bits of advice.*

2.2 The Protocol `Cleaner`

In this section, we define a distributed protocol `Cleaner` using the oracle $\mathcal{O}$, that enables to clear any n-node synchronous network G starting from the homebase v_0. Protocol `Cleaner` is formally described in Figures 1 and 2.

Let us roughly describe our protocol. Our searchers can be in seven different states: DFS_TEST, DFS_BACK, CLEAR_UNLABELLED, CLEAR_UNLABELLED_BACK, CLEAR, WAIT, GUARD. Initially, all searchers stand at v_0. Each of them reads n on the label $\mathcal{O}(G, v_0)(v_0)$ of v_0 to initialize their counters. Then the searcher with the largest Id is elected to guard v_0 and switches to state GUARD, the other searchers become free and switch to state DFS_TEST. After the phase $1 \leq i \leq n-1$, our protocol ensures the following. (1) A subgraph G' of $G[v_0, \cdots, v_i]$ containing T_i as a subgraph is cleared. (2) For any vertex v of the border of G', i.e. v is incident to an edge in $E(G')$ and an edge of $E(G) \setminus E(G')$, one searcher is guarding v (in state GUARD). (3) Any other searcher is free and stand at a vertex of G'.

During the first stage of the phase $i+1$, the free searchers are aiming at clearing the unlabelled edges of those vertices v of $V(G[v_0, \cdots, v_i])$ such that the largest unlabelled edge e incident to v satisfies $f_i \prec e \prec f_{i+1}$. Note that such an edge e belongs to $E(G[v_0, \cdots, v_i])$. For this purpose, any free searcher performs a DFS of T_i thanks to the labels `PARENT` and `CHILD`. The searcher is in state DFS_TEST if it goes down in the tree, in state DFS_BACK otherwise.

During this DFS, if the searcher meets a vertex v_j $(j \leq i)$ labelled in such a way that $\texttt{TCU}_{v_j} = i+1$ (recall that `TCU` means *Time to Clear Unlabelled edges*), then the searcher clears all unlabelled edges of v_j and then it carries on the DFS. To clear the unlabelled edges of v_j, the searcher take successively, in the order of the port numbers, all the unlabelled ports. It takes each unlabelled port back and forth, in state CLEAR_UNLABELLED for the first direction, and CLEAR_UNLABELLED_BACK for the second direction.

Moreover, during this stage, any searcher that is guarding a vertex labelled in such a way that ($\texttt{TYPE} = 1$ and $\texttt{TCU} < \texttt{TTL} = i+1$) is aiming at clearing, in state CLEAR, the edge corresponding to the port number `LASTPORT` of the considered vertex (recall that `TTL` means *Time To Leave*). Protocol `Cleaner` ensures that the corresponding port number corresponds to the single contaminated edge incident to the considered vertex at this stage.

Before the first round of the second stage of phase $i+1$, the two following properties are satisfied: (1) if there exists a vertex v such that v is labelled with ($\texttt{TYPE}_v = 0$ and $\texttt{TTL}_v = i+1$), f_{i+1} is the only contaminated edge incident to $v = parent(v_{i+1})$, and (2) for any vertex v labelled in such a way that ($\texttt{TYPE}_v = 1$ and $\texttt{TCU}_v = \texttt{TTL}_v = i+1$), the edge corresponding to $\texttt{LASTPORT}_v$ is the only contaminated edge incident to v.

During the second stage of the phase $i+1$, Protocol `Cleaner` performs the clearing of f_{i+1} (incident to $parent(v_{i+1}) \in V(G')$) and the clearing of any edge corresponding to port number $\texttt{LASTPORT}_{v_j}$ of a vertex v_j $(j \leq i)$ labelled in such a way that ($\texttt{TYPE}_{v_j} = 1$ and $\texttt{TCU}_{v_j} = \texttt{TTL}_{v_j} = i+1$). For this purpose, any free searcher performs a DFS of T_i.

Program of searcher A.

Initialisation: /* all searchers start at v_0 */
Read n on $\mathcal{O}(G, v_0)$ to initialize the counter;
if A is the searcher with the largest ID at v_0 **then**
Switch to the state GUARD;
else
At the first round on the second stage of phase 1,
Switch to the state DFS_TEST;
endif

Program of searcher A at any round of stage $s \in \{0, 1\}$ of phase $1 \leq i \leq n$.

/* Searcher A arrives at node v_j, coming by port number p_ℓ of v_j */
(corresponding to the edge $\{v_\ell, v_j\}$).

Let p_{first} be the smallest unlabelled port number of v_j.
$p_{first} = -1$ if there are no such edges.
Let p_{next} be the smallest unlabelled port number p of v_j, such that $p > p_\ell$.
$p_{next} = -1$ if there are no such edges.

Let $p_{firstChild}$ be the port number p of v_j such that it exists $1 \leq k \leq n-1$ with p being labelled CHILD(k), and for any $1 \leq k' < k$, no port numbers of v_j are labelled CHILD(k'). $p_{firstChild} = -1$ if there are no such edges.
If $p_{firstChild} \neq -1$, let $firstChild$ denote the corresponding neighbour of v_j.

Let $p_{nextChild}$ be the port number p of v_j such that it exists $\ell < k \leq n-1$ with p being labelled CHILD(k), and for any $\ell \leq k' < k$, no port numbers of v_j are labelled CHILD(k'). If $p_{nextChild} \neq -1$, $nextChild$ denotes the corresponding neighbour of v_j.

```
Case:
    state = DFS_TEST
        if s = 1 and there is a port p labelled CHILD(i) then
            Take port p in state CLEAR;
        else if s = 0 and TCU = i then
            Take port p_first in state CLEAR_UNLABELLED;
        else if p_firstChild ≠ −1 and firstChild ⪯ v_{i−1} then
            Take port p_firstChild in state DFS_TEST;
        else Take port labelled PARENT in state DFS_BACK;
        endif

    state = CLEAR_UNLABELLED_BACK
        if p_next ≠ −1 then
            Take port p_next in state CLEAR_UNLABELLED;
        else if p_firstChild ≠ −1 and firstChild ⪯ v_{i−1} then
            Take port p_firstChild in state DFS_TEST;
        else Take port labelled PARENT in state DFS_BACK;
        endif
```

Fig. 1. Protocol Cleaner (1/2)

```
    state = CLEAR_UNLABELLED
        Take port p_ℓ in state CLEAR_UNLABELLED_BACK;

    state = DFS_BACK
        if s = 1 and there is a port p labelled CHILD(i) then
            Take port p in state CLEAR;
        else if p_nextChild ≠ −1 and nextChild ⪯ v_{i−1} then
            Take port p_nextChild in state DFS_TEST;
        else if PARENT ≠ −1 then
            Take port labelled PARENT in state DFS_BACK;
        else Take port CHILD(1) in state DFS_TEST;
        endif

    state = CLEAR
        if v_j ≺ v_i or deg(v_j) = 1 then
            if j > 0 then
                Take port labelled PARENT in state DFS_BACK;
            else Take port labelled CHILD(1) in state DFS_TEST;
            endif
        else Switch to the state WAIT;
        endif

/* Searcher that stands at node v_j */

    state = GUARD
        if TYPE = 1 then
            if TCU = TTL then
                At the first round of the second stage of phase TTL,
                Take port labelled LASTPORT in state CLEAR;
            else At the first round of the first stage of phase TTL,
              take port labelled LASTPORT in state CLEAR;
            endif
        else At the first round of the second stage of phase TTL
            take port CHILD(TTL) in state CLEAR;
        endif

    state = WAIT
        At the last round of this phase:
        if A is the searcher with the greatest ID at v_j then
            Switch to the state GUARD;
        else Take port labelled PARENT in state DFS_BACK;
        endif

end
```

Fig. 2. Protocol `Cleaner` (2/2)

When the searcher meets the vertex $parent(v_{i+1})$ whose a port number is labelled $\texttt{CHILD}(i+1)$, it takes the corresponding edge in state CLEAR. Moreover, any searcher that is guarding the vertex $parent(v_{i+1})$ also takes the edge corresponding to $\texttt{CHILD}(i+1)$ in state CLEAR if ($\texttt{TTL} = i+1$ and $\texttt{TYPE} = 0$). Finally, any searcher that is guarding a vertex labelled in such a way that ($\texttt{TYPE} = 1$ and $\texttt{TCU} = \texttt{TTC} = i+1$), takes the edge corresponding to port number LASTPORT in state CLEAR. During this stage, any searcher arriving at v_{i+1} waits (in state WAIT) the last round of the stage if $\deg(v_{i+1}) > 1$, else it becomes free. During this last round, if $\deg(v_{i+1}) > 1$, the searcher with largest Id that stands at v_{i+1} is elected to guard v_{i+1} while other searchers are free and take the port labelled PARENT in state DFS_BACK.

2.3 Sketch of Proof of Cleaner

In order to prove the correctness of our protocol we need the following notations. A searcher is called free if it is not in state GUARD nor WAIT. For any $0 \leq i \leq n-1$, let $M_i = \{v \in V(G) \mid \textit{for any edge } e \textit{ incident to } v,\ e \preceq f_i\}$. $M_i \subseteq V(T_i)$ is the set of the vertices whose all incident edge, but f_i, have been cleared by S before the step corresponding to the clearing of f_i. Moreover, we set $M_n = V$. Thus, after the step corresponding to the clearing of f_i, no vertices in M_i need to be guarded in the strategy S. Note that, for any $0 \leq j \leq n-1$, the set $M_j \setminus M_{j-1}$ is exactly the set of vertices v such that $\texttt{TTL} = j$. The proof of theorem 1 easily follows from the following lemma.

Lemma 2. *Let G be a connected graph and $v_0 \in V(G)$. Let S be a strategy that clears the graph G, starting from v_0, and using the smallest number of searchers. Let $\mathcal{O}(G, v_0)$ be the labelling of the vertices of G, using $\mathcal{L}(S)$. After the last round of the phase $i \geq 1$ of the execution of Protocol* Cleaner*, the cleared part of the graph G satisfies the following:*

1. *any edge in $\{f_0, \cdots, f_i\}$ is clear,*
2. *any edge incident to vertex in M_i is clear,*
3. *there is exactly one searcher in state* GUARD *at any vertex of $V(T_i) \setminus M_i$,*
4. *any other searcher is free and stands at a vertex of T_i,*
5. *for any vertex v with $\texttt{TCU} \leq i$, any unlabelled edge of v is clear.*

Roughly speaking, this lemma implies that, after the phase i, those vertices that have been all their incident edges cleared by S (the strategy from which the oracle is defined) before the clearing of f_{i+1} are cleared by Protocol Cleaner as well. Moreover, there is a searcher in state GUARD at any vertex of the border of the clear part of the graph, which avoids any recontamination. The proof of the lemma is by induction on $1 \leq i \leq n$. One can easily check that the case $i = 1$ holds. Let us assume that the result holds for $1 \leq i \leq n-1$. We prove that it still holds after the last round of the phase $i+1$. We consider two cases according whether there is a free searcher or not. Due to lack of space, the proof of this lemma is omitted, and can be found in [17].

3 Lower Bound

In this section, we show that the upper bound proved in the previous section is almost optimal. More precisely, we prove that:

Theorem 2. *The search problem cannot be solved using only $o(n \log n)$ bits of advice.*

To prove the theorem, we build a $4n + 4$-node graph $\mathcal{G}_n$. Then, we prove that any distributed protocol requires $\Omega(n \log n)$ bits of advice to clear $\mathcal{G}_n$ in a monotone connected way starting from $v_0 \in V(\mathcal{G}_n)$, and using the fewest number of searchers.

Let $n \geq 4$. Let $t = 2n+7$. Let $P = \{v_1, \cdots, v_t\}$ be a path and let K_{n-2}, resp. K_n, be a $(n-2)$-clique, resp. a n-clique. We obtain the graph $\mathcal{G}_n$ by adding all edges between v_i and the vertices of K_{n-2}, for any $1 \leq i \leq t$. Then, let the node v_t coincide with a vertex of K_n. Finally, let us choose one vertex of K_{n-2} and denote it by v_0.

We now enumerate some technical lemmas that describe how any search strategy clears $\mathcal{G}_n$ using the fewest number of searchers.

Lemma 3. *The smallest number of searchers sufficient to clear $\mathcal{G}_n$ is n.*

Proof. Since $\mathcal{G}_n$ admits K_n as a minor, we get that the smallest number of searcher required to clear $\mathcal{G}_n$ is at least n. We now describe a strategy that clears $\mathcal{G}_n$ using n searchers. Starting from v_0, move one searcher to guard any vertex of K_{n-2}. Use the two remaining searchers to clear any edge of $E(K_{n-2})$. Then, move one remaining searcher to v_1. The second remaining searcher clears any edge between v_1 and K_{n-2}. Then, the searcher at v_1 move to v_2 and the second remaining searcher clears any edge between v_2 and K_{n-2}. And so on, until any vertex of P has been cleared. At this step, there are one searcher at any vertex of K_{n-2} and one searcher at v_t. Finally, let us use all the searchers to clear K_n. $\square$

Lemma 4. *For any optimal search strategy that clears $\mathcal{G}_n$, the last vertex of $\mathcal{G}_n$ to have all its incident edges clear belongs to $V(K_n)$.*

Proof. During the clearing of K_n, the n searchers must stand at vertices of K_n. Thus, v_0 is not occupied by a searcher anymore. To avoid recontamination, any vertex of P and K_{n-2} must have all its incident edges clear. $\square$

Due to lack of space, the proof of the following lemma is omitted, and can be found in [17].

Lemma 5. *For any optimal search strategy that clears $\mathcal{G}_n$, the first vertex of $\mathcal{G}_n$ to have all its incident edges clear is v_1 or v_2. Moreover, at this step, any vertex of K_{n-2} is occupied by a searcher, and no vertices of $\{v_4, \cdots, v_t\}$ have been occupied.*

The following lemma aims at proving that any strategy clearing $\mathcal{G}_n$ using n searchers and starting from v_0 is strongly constrained.

Lemma 6. *Let S be an optimal connected search strategy that clears $\mathcal{G}_n$ starting from v_0. For any $5 \leq i \leq t-2$, at the first step of S when a searcher reaches v_i, the following is satisfied:*

- *any vertex in $V(K_n) \cup \{v_{i+1}, \cdots, v_t\}$ has all its incident edges contaminated;*
- *there is one searcher at any vertex of K_{n-2};*
- *any vertex in $\{v_1, \cdots, v_{i-2}\}$ has all its incident edges clear;*
- *either v_{i-1} has all its incident edges clear, or there is a searcher at v_{i-1} and v_{i-1} has only one incident edge that is still contaminated. In the latter case, the next move consists in moving a searcher along the last contaminated edge incident to v_{i-1}.*

Proof. Let s be the first step of the strategy such that, after this step, a searcher is occupying v_i. Let us consider the situation just before this step. Since $i \geq 5$, by Lemma 5, just before step s, v_1 or v_2 has all its incident edges clear, and there are one searcher at any vertex from K_{n-2} to preserve them from recontamination. Moreover, there is a vertex on the path between v_1 and v_i in P, that is occupied by a searcher for preserving v_1 or v_2 from recontamination. Let j, $1 < j < i$, be the minimum index such that a searcher is standing at v_j. Note that, for any k, $1 \leq k < j$, v_k has all its incident edge clear.

First, let us show that for any $\ell > i$, v_ℓ is not occupied before step s. For purpose of contradiction, let us assume v_ℓ is occupied. Since v_i has all its incident edges contaminated, for any k, $j < k < \ell$, v_k has all its incident edges contaminated. By Lemma 4 a vertex of K_n has at least one contaminated incident edge. Thus, for any k, $\ell < k \leq t$, v_k has all its incident edges contaminated, since there are no searchers on the path between v_k and K_n. Thus, there exits $k \neq i$ such that v_k has all its incident edges contaminated. Thus, the searchers at K_{n-2} cannot move, because they preserve recontamination from v_i and v_k. The searcher at v_ℓ cannot move because it preserves recontamination from v_i and K_n. The searcher at v_j may move at v_{j+1}, but then could not move anymore. Then the strategy fails, a contradiction. This proves the first item of the lemma.

Thus, before step s, there are one searcher at any vertex of K_{n-2}. These searchers preserve recontamination from v_i and v_t. Therefore, they cannot move. This proves the second item of the lemma.

According to the first item of the lemma, v_{i-1} has been reached before v_i. Since the strategy is monotone, just before the step s, a searcher is occupying v_{i-1}. Two cases must be considered:

- If s consists in moving a searcher occupying v_{i-1} along the edge $\{v_{i-1}, v_i\}$, the monotonicity of the strategy implies that either all edges incident to v_{i-1} are clear, or just before step s two searchers were occupying v_{i-1}. In the first case, the lemma is valid. Thus, let us assume that at least one edge incident to v_{i-1} is still contaminated after step s. Since $i \leq t-2$, any vertex in $V(K_{n-2}) \cup \{v_i\}$ is occupied by a searcher, and incident to at least two contaminated edges: all edges incident to v_{i+1} and v_{i+2} are contaminated. If more than one edge incident to v_{i-1} is contaminated, the strategy fails. Therefore, at most one edge incident to v_{i-1} is contaminated, and the single possible move consists in moving the searcher at v_{i-1} along this edge.

– Else, the step s consists in moving a searcher along an edge between a vertex u of K_{n-2} and v_i. Since $i \leq t-2$, there must be two searchers at u just before step s. Again, just after step s, any vertex in $V(K_{n-2}) \cup \{v_i\}$ is occupied by a searcher, and incident to at least two contaminated edges: all edges incident to v_{i+1} and v_{i+2} are contaminated. Moreover, a searcher is occupying v_{i-1} and $\{v_{i-1}, v_i\}$ is contaminated. If another edge incident to v_{i-1} is contaminated, the strategy fails. Hence, at most one edge incident to v_{i-1} is contaminated, and the single possible move consists in moving the searcher at v_{i-1} along this edge.

This concludes the proof of the lemma. □

A *local orientation* of a graph is a mapping from the incidence of the graph (between a vertex and an edge) into the port number of the graph. An instance of the problem consists of a graph, a vertex of this graph (the homebase) and a local orientation for this graph. Let $\mathcal{C}$ be the set of the following instances $\{(\mathcal{G}, v_0, \ell o) \mid \ell o$ is a local orientation of $\mathcal{G}\}$. Let $\mathcal{I} = |\mathcal{C}|$. The following lemma proves that any distributed protocol, using an arbitrary string of bits of advice, can clear only some amount of the instances of $\mathcal{C}$.

Lemma 7. *Let $\mathcal{P}$ be a distributed protocol for solving the search problem. Let f be a binary string of bits of advice provided by an oracle. Using f, $\mathcal{P}$ can clear at most $\mathcal{I} * (\frac{1}{n-2})^n$ instances of $\mathcal{C}$.*

Proof. Let $\mathcal{I}_{k,j}$ be the number of instances such that $(\mathcal{P}, f)$ allows to a searcher to clear j edges between v_k and K_{n-2}. We prove that, for $5 \leq k \leq n+5$ and any $1 \leq j \leq n-3$, $\mathcal{I}_{k,j} \leq \mathcal{I}_{k,j-1}\frac{n-j-1}{n-j}$.

Let us consider the last step such that exactly $0 \leq j \leq n-3$ edges between v_k and K_{n-2} are clear. By the lemma above, at this step, there is a searcher at v_k and a searcher at any vertex of K_{n-2}. Moreover, the remaining searcher cannot move to a vertex of $\{v_{k+1}, \cdots, v_t\}$. Let v be the vertex where this searcher stands. Using f, protocol $\mathcal{P}$ chooses a port number p that the remaining searcher must take. There are two cases according whether the remaining searcher stands at v_k or at a vertex of K_{n-2}.

– If the remaining searcher stands at v_k, it remains $n-j-1$ contaminated edges incident to this vertex and the strategy fails if p leads to v_{k+1}. Thus, the strategy fails in at least $\mathcal{I}_{k,j}\frac{1}{n-j-1}$ instances. Therefore, $\mathcal{I}_{k,j+1} \leq \mathcal{I}_{k,j}\frac{n-j-2}{n-j-1}$.
– If the remaining searcher stands at a vertex of K_{n-2}, it remains at most $n-3+t-k+1$ contaminated edges incident to this vertex and the strategy fails if p leads to one vertex in $\{v_{k+1}, \cdots, v_t\}$. Thus, the strategy fails in at least $\mathcal{I}_{k,j-1}(\frac{t-k}{n-3+t-k+1})$ instances. Hence, $\mathcal{I}_{k,j} \leq \mathcal{I}_{k,j-1}\frac{n-2}{t+n-2-k}$. To conclude, it is sufficient to remark that, since $n \geq 4$, $t = 2n+7$, $1 \leq j \leq n-3$

and $5 \leq k \leq n-5$, we have $\frac{n-2}{t+n-2-k} \leq \frac{n-2}{2n}$ and $\frac{n-j-2}{n-j-1} \geq \frac{n-3}{2}$. Thus, $\frac{n-2}{t+n-2-k} \leq \frac{n-j-2}{n-j-1}$.

Hence, $\mathcal{I}_{k,n-2} \leq \mathcal{I}_{k-1,n-2} \prod_{j=1..n-3}(\frac{n-j-2}{n-j-1}) = \mathcal{I}_{k-1,n-2}(\frac{1}{n-2})$. Using f, $\mathcal{P}$ can clear at most $\mathcal{I}_{n-5,n-2} \leq \mathcal{I}_{5,n-2}(\frac{1}{n-2})^n$. Since, $\mathcal{I}_{5,n-2} \leq \mathcal{I}$, the lemma holds. □

Proof. of the Theorem 2. Let $N = |V(\mathcal{G})| = 4n+4$. To prove the theorem, it is sufficient to prove that for any $\alpha < 1/4$, and for any oracle that provides less than $q = \alpha N \log N$ bits of advice, no distributed protocol using $\mathcal{O}$ permit to clear all instances of $\mathcal{C}$. Let $\mathcal{O}$ be such an oracle. The number of functions f that the oracle $\mathcal{O}$ can output for $\mathcal{G}_n$ is at most $(q+1)2^q\binom{N+q}{N}$ [12]. Thus, there exists a set $\mathcal{S} \subseteq \mathcal{C}$ of at least $B = \frac{\mathcal{I}}{(q+1)2^q\binom{N+q}{N}}$ instances of $\mathcal{C}$ for which $\mathcal{O}$ returns the same string of bits of advice.

Let $\mathcal{P}$ be a distributed protocol that uses the oracle $\mathcal{O}$ for solving the search problem. By Lemma 7, $\mathcal{P}$ cannot clear more than $\mathcal{I} * (\frac{1}{n-2})^n$ instances of $\mathcal{C}$ using the same string of bits of advice.

To conclude, it remains to prove that $B > \mathcal{I} * (\frac{1}{n-2})^n$. Indeed,

$$B * (\frac{(n-2)^n}{\mathcal{I}}) = \frac{(n-2)^n}{(q+1)2^q\binom{N+q}{N}}$$

Using the Stirling formula we get that for n large enough,

$$B * (\frac{(n-2)^n}{\mathcal{I}}) \sim \frac{(n-2)^n}{2^{\alpha N \log N}(1+\alpha \log N)^N} * (\frac{\alpha \log N}{1+\alpha \log N})^{\alpha N \log N}$$

Since $N = 4n+4$, we obtain:

$$\log[B * (\frac{(n-2)^n}{\mathcal{I}})] \sim (1-4\alpha) n \log n$$

Since $\alpha < 1/4$, we get that $B > \mathcal{I} * (\frac{1}{n-2})^n$. Thus, the result holds. □

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. M.A. Bender, A. Fernandez, D. Ron, A. Sahai and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs. Information and Computation 176, pages 1–21, 2002.
4. 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.

5. D. Bienstock and P. Seymour. Monotonicity in graph searching. Journal of Algorithms 12, pages 239-245, 1991.
6. 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.
7. R. Breisch. An intuitive approach to speleotopology. Southwestern Cavers VI(5), pages 72-78, 1967
8. A.E.F. Clementi, A. Monti and R. Silvestri. Selective families, superimposed codes, and broadcasting on unknown radio networks. In 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA), pages 709-718, 2001.
9. F. Fich and E. Ruppert Hundreds of impossibility results for distributed computing. Distributed Computing 16, pages 121–163, 2003.
10. 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.
11. 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.
12. P. Fraigniaud, D. Ilcinkas and A. Pelc. Oracle Size: a New Measure of Difficulty for Communication Tasks. In 25th Annual ACM Symp. on Principles of Distributed Computing (PODC), pages 179-187, 2006.
13. P. Fraigniaud, D. Ilcinkas and A. Pelc. Tree Exploration with an Oracle. In 31st International Symposium on Mathematical Foundations of Computer Science (MFCS), LNCS 4162, pages 24-37, 2006.
14. P. Fraigniaud and N. Nisse. Connected Treewidth and Connected Graph Searching. In 7th Latin American Theoretical Informatics Symp. (LATIN 2006), LNCS 3887, pages 470-490, 2006.
15. A. LaPaugh. Recontamination does not help to search a graph. Journal of the ACM 40(2), pages 224-245, 1993.
16. 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.
17. N. Nisse and D. Soguet. Graph searching with advice. Technical Report LRI-1469, University Paris-Sud, France, Mar. 2007.
18. T. Parson. Pursuit-evasion in a graph. Theory and Applications of Graphs, Lecture Notes in Mathematics, Springer-Verlag, pages 426-441, 1976.
19. P. Seymour and R. Thomas. Graph searching and a min-max theorem for treewidth. J. Combin. Theory Ser. B, 58, pages 22–33, 1993.
20. 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.