Capture of an Intruder by Mobile Agents

Lali Barrière* Paola Flocchini† Pierre Fraigniaud‡ Nicola Santoro§

ABSTRACT

Consider a team of mobile software agents deployed to capture a (possibly hostile) *intruder* in a network. All agents, including the intruder move along the network links; the intruder could be arbitrarily fast, and aware of the positions of all the agents. The problem is to design the agents' strategy for capturing the intruder. The main efficiency parameter is the size of the team. This is an instance of the well known *graph-searching* problem whose many variants have been extensively studied in the literature. In all existing solutions, and in all the variants of the problem, it is assumed that agents can be removed from their current location and placed in another network site arbitrarily and at any time. As a consequence, the existing optimal strategies cannot be employed in situations for which agents cannot access the network at any point, or cannot "jump" across the network, or cannot reach an arbitrary point of the network via an internal travel through insecure zones. This motivates the *contiguous search problem* in which agents cannot be removed from the network, and clear links must form a connected sub-network at any time, providing safety of movements. This new problem is NP-complete in general. We study it for tree networks, and we consider its more general version, the weighted case, which arises naturally when considering networks whose nodes and links are of different nature and thus require a different number of agents to be explored. We give a linear-time algorithm that computes, for any tree T, the minimum number of agents to capture the intruder, and the corresponding search strategy. Beside its optimality in time, our algorithm is naturally distributed: if T is a processor-network, then the minimal search strategy for T can be computed by T in a decentralized manner, using a linear number of messages.

*Departament de Matemàtica Aplicada IV, *Univ. Politècnica de Catalunya*, Spain. *lali@mat.upc.es.*

†School of Information Technology and Engineering, *University of Ottawa*, Canada. *flocchin@site.uottawa.ca.*

‡CNRS, Laboratoire de Recherche en Informatique, *Université Paris-Sud*, France. *pierre@lri.fr.*

§School of Computer Science, *Carleton University*, Canada. *santoro@scs.carleton.ca.*

SPAA'02, August 10-13, 2002, Winnipeg, Manitoba, Canada.

Keywords: Graph-searching, mobile agent, network intruder.

1. INTRODUCTION

Networked environments which support mobile agents can be penetrated by possibly harmful agents, called *intruders*. Concern for the severe damage intruders can cause has motivated a large amount of research, especially on detection, whose focus is on solutions by teams of mobile agents (e.g., see [1, 13, 15, 16, 31, 37]). Once the presence of an intruder is detected, a team of mobile system agents is deployed to capture it. Both the intruder and the agents move along the network links, but the intruder could be arbitrarily fast, and aware of the positions of all the agents. The agents, starting from their homebase, search the network looking for the intruder, according to a given strategy (or protocol). The intruder is captured as soon as it is surrounded by the agents and has no escape link (i.e., no link free from any agent, by which the intruder could move). The problem is to design the agents' strategy for capturing the intruder. The efficiency parameter is the *size* of the team.

From a theoretical point of view, this is an instance of a well known and extensively studied problem known as *graph-searching.*

1.1 Graph Search

In the graph-searching problem, first discussed by Breisch [5] and by Parson [26, 27], we are given a "contaminated" network, i.e., whose links are all contaminated. Via a sequence of operations using "searchers", we would like to obtain a state of the network in which all links are simultaneously clear. The goal is to use as few searchers as possible to decontaminate the network. There are several variants of the problem. For the *edge-search* problem, a *search step* is one of the following operations: (1) *place* a searcher on a node, (2) *remove* a searcher from a node, (3) *move* a searcher along a link. There are two ways in which a contaminated link can become clear. In both cases, a searcher traverses the link from one extremity u to the other extremity v. The two cases are depending on the way the link is preserved from recontamination: either another searcher remains in u, or all other links incident to u are clear. A *search strategy* is a sequence of search steps that results in all links being simultaneously clear. The *search number* $s(G)$ of a network

G is the smallest number of searchers for which a search strategy exists. A search strategy using $s(G)$ searchers in G is called *minimal*.

Megiddo, Hakimi, Garey, Johnson and Papadimitriou [24] proved that determining whether $s(G) \leq k$ is NP-complete. They gave an $O(n)$-time algorithm to determine the search number of n-node trees, and an $O(n \log n)$-time algorithm to determine a minimal search strategy in n-node trees. Ellis, Sudborough and Turner [9] linked $s(G)$ with the vertex separation $\mathrm{vs}(G)$ of G (known to be equal to the pathwidth of G [17]). Ellis *et al.* showed that $\mathrm{vs}(G) \leq s(G) \leq \mathrm{vs}(G) + 2$, and that $s(G) = \mathrm{vs}(G')$ where G' is the 2-augmentation of G, i.e., the network obtained from G by replacing every link $\{x, y\}$ by a path $\{x, a, b, y\}$ of length 3 between x and y. They also showed that the vertex separation of trees can be computed in linear time, and they gave an $O(n \log n)$-time algorithm for computing the corresponding layout. It yields another $O(n)$-time algorithm returning the search number of trees, and an $O(n \log n)$-time algorithm returning a minimal search strategy in trees. In [29], Skodinis recently showed that an optimal layout can also be computed in linear time, and hence a minimal search strategy can be computed in linear time.

Beside network security [14], the graph-searching problem has many other applications, including pursuit-evasion problems in a labyrinth [26], decontamination problems in a system of tunnels, and mobile computing problems in which agents or robots [11] are looking for a hostile intruder [33]. Moreover, the graph-searching problem also arises in VLSI design through its equivalence with the gate matrix layout problem [10, 22]. It is hence not surprising that it gave rise to numerous papers. Another reason for this success is that the problem and its several variants (node-search, mixed-search, t-search, etc.), is closely related to standard graph parameters and concepts, including tree-width, cut-width, path-width, and, last but not least, graph minors [4]. For instance, Makedon and Sudborough [23] showed that $s(G)$ is equal to the cutwidth of G for all networks of maximum degree 3. Similarly, Kirousis and Papadimitriou showed that the node-search number of a network is equal to its interval-width [18], and to its vertex separator plus one [19]. Seymour and Thomas [28] showed that the t-search number is equal to the tree-width plus one. Takahashi, Ueno and Kajitani [35] showed that the mixed-search number is equal to the proper path-width. In [3], Bienstock and Seymour simplified the proof of Lapaugh's result [21] stating that there is a minimal search strategy that does not recontaminate any link (see also [2]). Thilikos [36] used graph minors to derive a linear-time algorithm that checks whether a network has a search number at most 2. For other results on graph-searching, the reader is referred to [7, 8, 12, 30, 32]. Contributions to related search problems can be found in [6, 25, 33, 34, 38, 39] and the references therein.

1.2 Limit of Existing Solutions

In all existing solutions for the standard version of the problem (i.e., edge-search), as well as for any of its variants known to the authors (e.g., node-search, or mixed-search), it is assumed that agents can be removed from their current location and placed in another network site arbitrarily and at any time. This assumption makes the existing optimal solutions difficult if not impossible to implement in some applications, including the one considered in this paper. In particular, the removal of a searcher from a node x and the placement of this searcher in another node y might be difficult in practice, either because it forces a searcher to travel between the two nodes x and y in an unsafe environment, or it forces a searcher to go "out of the system" and reenter the system elsewhere. Both solutions can be unrealistic or dangerous for the searchers.

For applications we are interested in, a feasible solution would consist to let all searchers enter the systems once at the same point (the homebase in our application) and to let the searchers move in the system while insuring mutual protection and/or permanent easy and secure access to the entering point. This motivates the study of edge-search where the *removal of an agent is not allowed.* Even just this assumption considerably changes the nature of the problem. For instance, it is mentioned in [24] that there are trees for which minimal search strategies (i.e., search strategies using the minimum number of searchers) without removal require $\Omega(n \log n)$ moves (i.e., link traversal), whereas, if the removal of agents is allowed, then, for any network G, there exists a minimal search strategy that requires at most $O(n)$ moves in G [21].

But the key point is that minimal search strategies often require the set of clear links to be non connected as different non contiguous parts of the network must be cleared separately to minimize the total number of searchers involved. We hence introduce the new notion of *contiguous search.* A contiguous search strategy is an edge-search strategy in which (1) the removal of searchers is not allowed, and (2) at any time of the search strategy, the set of clear links forms a connected subnetwork. Of course, no contiguous search can exist in a non-connected network. Therefore, in the remaining of this paper, the input network is always assumed to be connected.

1.3 Summary of Results

The problem of determining the contiguous search number (i.e., the minimum team size) and the optimal search strategy (to be followed by the agents) is NP-complete for arbitrary networks. We study and solve it for tree networks, in its more general weighted version.

We first prove that there is always an optimal strategy which is *monotone*, establishing for contiguous search in trees an analogous of Lapaugh's result.

We then construct a linear-time, and thus optimal algorithm for computing the contiguous search number and the corresponding minimal contiguous search strategy of any n-node tree. Note that even if there are trees that require $\Omega(n \log n)$ moves to be cleared, the corresponding sequences of moves can each be computed in $O(n)$ time and coded in $O(n)$ space in the standard $\lceil \log n \rceil$-bits word-RAM model.

Beside its time optimality, our algorithm offers many important features.

Firstly, one can easily envision situations for which the number of searchers required to clear a link might not be the same for all links. For example this occurs quite naturally in the case of wide tunnels or complex corridors, and of course in networks whose links can be of very different nature. Similarly, the number of searchers required to maintain a node clear from unsafe neighbors or links could depend on the node in question (e.g., routers of a large network can vary significantly). Our algorithm applies to the link- and node-weighted version of the problem. Still, its time-complexity remains linear.

Secondly, our algorithm is naturally distributed. If T is a processor-network, then the minimal search number as well as the search strategy for T can be computed in a decentralized manner. In particular, processing elements connected as a tree network can distributively determine the minimal strategy of software agents aiming to surround an intrusive agent. The distributed solution uses $O(n)$ messages and is thus optimal.

Finally, we study extremal trees, i.e., trees requiring the maximum number of searchers to be cleared. We show that n-node extremal trees require roughly $\log_2 n$ searchers to be cleared. This contrasts with the standard edge-search problem for which n-node extremal trees require only $\log_3 n$ searchers to be cleared [9, 24].

2. DEFINITIONS

As for the edge-search problem, we are given a contaminated network $G = (V, E)$, and the problem consists to obtain, via a sequence of operations using searchers, a state of the network in which all links are simultaneously clear. (Again, we assume that G is connected.) A search step of a contiguous search is one of the following two operations (1) place one searcher on a node, and (2) move one searcher along a link, both operations being performed under the constraint that the set X_i of all clear links after step i must be connected, for any i.

Every step lasts one unit of time, step i starting at time $i-1$, and finishing at time i. During each step, a searcher s is either at a node or traversing a link.

In all the literature on the graph-searching problem, there has been an implicit assumption that the number of searchers required to clear a link or a node is unitary. (In all existing strategies, more than one searcher might be sent across a link, or might reside at a node at a given time. However this fact reflects solely the overall strategic plan.) As mentioned in Section 1.3, this assumption is rather restrictive. We shall hence consider the contiguous search problem in its more general formulation of searching a weighted graph, that is, the problem of clearing a network with clearing *requirements* (or weights) associated to each node and each link. Observe that, in this formulation of the problem, the presence of searchers on a link or at a node might not prevent recontamination if their number is less than the corresponding requirement.

In what follows, we consider networks G weighted by $\omega : V \cup E \to \mathbb{Z}^+$, a mapping describing the clearing requirement of links and nodes. In other words, clearing a link e requires at least $\omega(e)$ searchers to be moved along e. A clear link e is preserved from recontamination if, for each extremity x, either at least $\omega(x)$ searchers occupy x, or all other links incident to x are clear (which is alway satisfied if $\deg(x) = 1$). We will assume that a node is at least as hard to guard as the maximum number of searchers required to clear any of its incident link. I.e., for every node x, we have:

$$\omega(x) \geq \omega(e) \text{ for any link } e \text{ incident to } x. \tag{1}$$

This is a natural assumption which includes specific cases such as $\omega(x) = \sum_e \omega(e)$ and $\omega(x) = \max_e \omega(e)$ where the summation and the maximum are taken over all links e incident to x.

For $X \subseteq E$, we denote by $\delta(X)$ the set of vertices having an incident link in X and another in $E \setminus X$. Note that since an edge $e \in E$ is a pair of vertices, $\delta(X) \cap e$ is the endpoint(s) of e which belong(s) to $\delta(X)$. Clearly, if X_i denotes the set of clear links at time i, i.e., just after step i, then there are $\omega(x)$ searchers occupying every node x in $\delta(X_i)$, and the total number of searchers in X_i is at least

$$\omega(X_i) = \sum_{x \in \delta(X_i)} \omega(x). \tag{2}$$

To summarize:

Definition 1. *A search step of a contiguous search in a weighted network is one of the following two operations (1) place a team of $q \geq 1$ searchers on a node, and (2) move a team of $q \geq 1$ searchers along a link, both operations being performed under the constraint that the set X_i of all clear links after step i must be connected, for any i.*

As for the standard (unweighted) edge-search, a search strategy is a sequence of contiguous search steps that results in all links being simultaneously clear. Let $cs(G)$, called the *contiguous search number* of G, denote the minimum number of searchers needed to solve the contiguous search problem in G. A contiguous search strategy is *minimal* if it involves $cs(G)$ searchers.

Remark 1. Note that Definition 1 does not assume that all searchers are simultaneously placed at the same point. However, we will see in Theorem 1 hereafter that enforcing this property does not increase the contiguous search number in trees.

Remark 2. We note that, given any (weighted) network G, and any integer k, determining whether $cs(G) \leq k$ is NP-complete, even if all weights are equal to 1. Indeed, the reduction of the NP-complete problem `min-cut into equal-sized subsets` to the (standard) edge-search problem given in [24] involves search strategies that are actually contiguous.

From the previous remark, the remaining part of the paper will focus on the class of weighted trees for which we will show that linear-time algorithms returning minimal contiguous search strategies exist.

3. MONOTONICITY OF CONTIGUOUS SEARCH IN TREES

A strategy is *monotone* if the following condition holds: once a link becomes clear, it cannot be contaminated ever again. The importance of monotone strategy derives not only from the desirability, within the application context, of avoiding recontamination, but also and mainly from the following property which is the analogy of Lapaugh's result [21] for edge-search.

Theorem 1. *For any tree T there is a monotone contiguous search strategy using $cs(T)$ searchers. Moreover, this strategy satisfies that all searchers are initially placed at the same node x_0, and the first step consists to clear a link incident to x_0.*

The proof is inspired from the short and elegant proof of Bienstock and Seymour [3] for the monotonicity theorems of Lapaugh [21] and Kirousis and Papadimitriou [18, 19]. The structure of the proof as well as the notions of crusades and progressive crusades are directly borrowed from [3]. The proof is nevertheless reported below for its non obvious adaptation to the weighted version of the problem, to point out where the fact that T is a tree is used, and to prove the additional condition (i.e., all searchers are first placed on a single node before starting clearing any link).

Let G be any network, and let $\widehat{G}$ be the network obtained from G by replacing every link e of G by two consecutive links e' and e'' of same weight $\omega(e)$, and whose common extremity has weight $\omega(e)$.

LEMMA 1. $cs(\widehat{G}) \leq cs(G)$.

PROOF. Directly by transforming any move along e in a contiguous search strategy for G in two moves in $\widehat{G}$ along e' and e''. If the move along e clears e, then $q \geq \omega(e)$ searchers have traversed e. From the setting of $\omega(e')$, $\omega(e'')$, and the weight of their common extremity, q searchers are enough to clear the path $\{e', e''\}$. □

Given a network $G = (V, E)$, a sequence $(X_0, X_1, \ldots, X_r)$ of subsets of links is a *crusade* if $X_0 = \emptyset$, $X_r = E$, and $|X_i \setminus X_{i-1}| \leq 1$ for any $1 \leq i \leq r$. The *frontier* of a crusade $(X_0, X_1, \ldots, X_r)$ is $\max_{1\leq i\leq r} \omega(X_i)$ where $\omega(X_i)$ is defined in Eq. 2. A crusade is *progressive* if $X_0 \subseteq X_1 \subseteq \cdots \subseteq X_r$ and $|X_i \setminus X_{i-1}| = 1$ for $1 \leq i \leq r$. Finally, a crusade is *connected* if X_i is connected for any $1 \leq i \leq r$.

LEMMA 2. *If $cs(G) \leq k$ then there exists a connected crusade of frontier $\leq k$ in G.*

PROOF. Given a contiguous search strategy S in a network G, let $C = (X_0, X_1, \ldots, X_r)$ be the sequence of subsets of links such that $X_0 = \emptyset$, and X_i is the set of clear links after step i of S. At most one link is cleared at every step of S, and hence $|X_i \setminus X_{i-1}| \leq 1$, i.e., C is a crusade. If S is a contiguous search strategy in G using at most k searchers, then obviously the frontier of C is at most k. Clearly, all X_i's are connected for $1 \leq i \leq r$ by definition of a contiguous search. □

LEMMA 3. *If there exists a connected crusade of frontier $\leq k$ in a tree T, then there exists a progressive connected crusade of frontier $\leq k$ in T.*

PROOF. The first part of the proof is adapted from (2.2) in [3]. Among all connected crusades of frontier at most k, choose a connected crusade $C = (X_0, X_1, \ldots, X_r)$ which satisfies:

(C1) $\sum_{i=0}^{r}(\omega(X_i) + 1)$ is minimum, and

(C2) $\sum_{i=0}^{r}|X_i|$ is minimum subject to (C1).

Let us show that this crusade is progressive.

First, we show that $|X_i \setminus X_{i-1}| = 1$ for every $i \geq 1$. Indeed, if $|X_i \setminus X_{i-1}| = 0$, i.e., $X_i \subseteq X_{i-1}$, then

$$C' = (X_0, X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_r)$$

is a connected crusade, contradicting (C1).

Next, we show that $X_{i-1} \subseteq X_i$ for every $i \geq 1$. We have

$$\omega(X_{i-1} \cup X_i) \geq \omega(X_i) \tag{3}$$

for every $i \geq 1$. Indeed, otherwise

$$C' = (X_0, X_1, \ldots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \ldots, X_r)$$

is a crusade of frontier at most k. $X_{i-1} \cup X_i$ is connected because X_{i-1} is connected, and $X_i \setminus X_{i-1}$ is incident to X_{i-1} since X_i is connected. Therefore C' is a connected crusade of frontier at most k, in contradiction with (C1).

For any two link-sets A and B, $\omega(A \cap B) + \omega(A \cup B) \leq \omega(A) + \omega(B)$ because every node appearing on the left hand side contributes at least as many times on the right hand side. Thence, we get from Eq. 3 that $\omega(X_{i-1} \cap X_i) \leq \omega(X_{i-1})$ for any $i \geq 1$. Therefore

$$C'' = (X_0, X_1, \ldots, X_{i-2}, X_{i-1} \cap X_i, X_i, \ldots, X_r)$$

is a crusade of frontier at most k. Assume that $X_{i-1} \cap X_i$ is not connected. Then let $W = X_{i-1} \setminus X_i$, $\{e\} = X_i \setminus X_{i-1}$, and $Z = X_{i-1} \cap X_i$. Since $X_{i-1} \cap X_i$ is not connected, $Z = Z' \cup Z''$ where Z' and Z'' do not share any extremity. Hence $X_{i-1} = Z' \cup Z'' \cup W$ and $X_i = Z' \cup Z'' \cup \{e\}$. Since X_{i-1} and X_i are both connected, and $e \notin W$, there is a cycle in $Z' \cup Z'' \cup W \cup \{e\}$, and hence in T, contradicting the fact that T is a tree. Therefore $X_{i-1} \cap X_i$ is connected, and thus C'' is a connected crusade of frontier at most k. From (C2), $|X_{i-1} \cap X_i| \geq |X_{i-1}|$ and hence $X_{i-1} \subseteq X_i$, i.e., C is a progressive connected crusade, which completes the proof. □

LEMMA 4. *Let G be a network such that every link has one of its extremities incident to exactly one other link. If there is a progressive connected crusade of frontier $\leq k$ in G, then there is a monotone contiguous search strategy using $\leq k$ searchers in G. Moreover, this strategy satisfies that the k searchers are initially placed at the same node x_0, and the first step consists to clear a link incident to x_0.*

PROOF. Let $(X_0, X_1, \ldots, X_r)$ be a progressive connected crusade of frontier $\leq k$ in G, and let $e_i = X_i \setminus X_{i-1} = \{x_i, y_i\}$. We construct a search strategy that successively clears the links $e_1, e_2, \ldots, e_r$.

Initially, we place k searchers in x_1. We have $\omega(x_1) + \omega(y_1) = \omega(X_1) \leq k$. Moreover, $\omega(e_1) \leq \omega(y_1)$ from Eq. 1. Hence, we clear link e_1 at step 1 by moving $\omega(y_1)$ searchers from x_1 to y_1 along e_1.

Assume now that we have cleared all links $e_1, \ldots, e_{i-1}$ without recontamination by a contiguous search. The link $e_i = \{x_i, y_i\}$ is incident to X_{i-1}, i.e., $\delta(X_{i-1}) \cap e_i \neq \emptyset$, say $x_i \in \delta(X_{i-1})$.

If $\omega(X_{i-1}) + \omega(y_1) \leq k$, then $\omega(y_1)$ searchers are moved to x_i, and traverse e_i to y_i. This clears link e_i since $\omega(e_i) \leq \omega(y_i)$ from Eq. 1.

If $\omega(X_{i-1}) + \omega(y_i) > k$, then at least one extremity of e_i is not in $\delta(X_i)$. Indeed, since $x_i \in \delta(X_{i-1})$: $x_i \in \delta(X_i) \Rightarrow \deg(x_i) > 2 \Rightarrow \deg(y_i) = 2$. Therefore, $y_i \in \delta(X_i)$ implies that the other link $f_i \neq e_i$ incident to y_i is contaminated, and hence $y_i \notin \delta(X_{i-1})$. Therefore $\omega(X_i) = \omega(X_{i-1}) + \omega(y_i) > k$, which contradicts the fact that the frontier of the crusade is $\leq k$. We hence consider three sub-cases:

1. $x_i \in \delta(X_i)$ and $y_i \notin \delta(X_i)$. Then, as before, $\deg(y_i) = 2$, and the other link $f_i \neq e_i$ incident to y_i belongs to X_{i-1}. Hence there are at least $\omega(y_i) \geq \omega(e_i)$ searchers occupying y_i, and which can be moved along e_i to clear it.

2. $x_i \notin \delta(X_i)$ and $y_i \in \delta(X_i)$. Hence $\omega(X_i) = \omega(X_{i-1}) - \omega(x_i) + \omega(y_i)$. On the other hand, since $x_i \in \delta(X_{i-1})$, it is occupied by at least $\omega(x_i) \geq \omega(e_i)$ searchers. Move the at most $k - \omega(X_{i-1})$ spare searchers to x_i. There are now $\omega(x_i) + k - \omega(X_{i-1}) \geq \omega(x_i) + \omega(X_i) - \omega(X_{i-1}) = \omega(y_i)$ searchers in x_i. By moving $\omega(y_i)$ of them from x_i to y_i, the link e_i is cleared.

3. $x_i \notin \delta(X_i)$ and $y_i \notin \delta(X_i)$. Hence e_i is the only contaminated link adjacent to y_i in $E(G) \setminus X_{i-1}$. Therefore, $\omega(x_i)$ searchers can move from x_i to y_i along e_i to clear it. (Note that possibly $\omega(y_i) > \omega(x_i)$ but that does not create any problem since all links incident to y_i are now clear.)

Clearly, the search strategy obtained by clearing all links as explained above satisfies the statement of the lemma. □

Note that the node x_0 where the k agents are initially placed in the search strategy constructed in the proof of Lemma 4 is one of the two extremities of the unique link in X_1. The choice of this extremity is arbitrary.

Let $\widehat{G}$ be the network obtained from a network G as described in Lemma 1. In particular, the set of nodes of $V(\widehat{G})$ of $\widehat{G}$ contains "original" nodes (i.e., nodes of $V(G)$), and "added" nodes (i.e., nodes resulting of the expansion of edges of $E(G)$).

LEMMA 5. *Any monotone contiguous search strategy for $\widehat{G}$ where searchers are placed only in original nodes can be transformed into a monotone contiguous search strategy for G using the same number k of searchers. Moreover, if (i) the search strategy in $\widehat{G}$ satisfies that the k searchers are initially placed at the same original node x_0, and (ii) step 1 consists to clear a link incident to x_0, then the corresponding search strategy in G satisfies the same two properties.*

PROOF. A monotone contiguous search strategy for G is obtained from a monotone contiguous search strategy for $\widehat{G}$ as follows. Let e' and e'' be two adjacent links of $\widehat{G}$ resulting from the expansion of a link e of G. Assume $e' = \{x, y\}$ and $e'' = \{y, z\}$. If q searchers move from x to y, or from z to y in $\widehat{G}$ then these searchers remain in place in G. If q searchers move from y to x (resp., from y to z) in $\widehat{G}$ then q searchers move from z to x (resp., from x to z) in G. □

PROOF OF THEOREM 1. Let T be a tree. From Lemma 1, $cs(\widehat{T}) \leq cs(T)$. From Lemma 2, we get a connected crusade of frontier $\leq cs(T)$ in $\widehat{T}$. From Lemma 3, there exists a progressive connected crusade of frontier $\leq cs(T)$ in $\widehat{T}$. From Lemma 4, we get that there is a monotone contiguous search strategy using $\leq cs(T)$ searchers in $\widehat{T}$. Moreover, this strategy satisfies that the k searchers are initially placed at the same node x_0 of $\widehat{T}$, and the first step consists to clear a link incident to x_0. Actually, as noted after the proof of Lemma 4, the original position of the searchers can be chosen to be a node in T since every link of $\widehat{T}$ has exactly one extremity in T. From Lemma 5, we hence get that there is a monotone contiguous search strategy using $\leq cs(T)$ searchers in T, and satisfying that all searchers are initially placed at the same node x_0, and the first step consists to clear a link incident to x_0. □

Note that only Lemma 3 uses the fact that T is a tree. We don't know whether Theorem 1 holds for arbitrary networks.

Notation. According to Theorem 1, contiguous search strategies will hence be described by an ordered sequence $\alpha = \alpha_0, \alpha_1, \alpha_2, \ldots$ where α_i is a triple (x_i, y_i, q_i), meaning the transfer of q_i agents from node x_i to a neighboring node y_i. All agents are initially placed at x_0.

4. OPTIMAL-TIME ALGORITHM FOR CONTIGUOUS SEARCH IN TREES

This section is devoted to the description of a linear-time algorithm returning an optimal contiguous search strategy for weighted trees. Let $T = (V, E)$ be a tree with n nodes.

According to Theorem 1, the quest for a minimal strategy can be restricted to monotone strategies satisfying the "single entry point" constraint, i.e., initially, at time $t = 0$, all searchers are in the same node x_0 called *homebase*, and the first step consists to clear a link incident to the homebase. Of course, the choice of the homebase affects the number of searcher of monotone strategies. Hence let $cs_x(T)$, called search number of T *from* node x, denote the minimum number of searchers of monotone strategies starting from $x \in V$. From Theorem 1, $cs(T) = \min_{x \in V}\{cs_x(T)\}$.

Any monotone strategy will induce an ordering on the neighbors (and thus on the incident links) of each node, where the ordering depends on whether one neighbor becomes occupied (and thus safe) before another. This ordering depends solely on the choice of the starting node x, and corresponds to the parent/child relationship in T_x where T_x denotes the tree T when rooted in node x. Hence, in the following, we will refer to a strategy for T starting from x also as the strategy *for* T_x, and alternatively denote $cs_x(T)$ by $cs(T_x)$.

Given a rooted tree T_x and one of its nodes y, let $T_x[y]$ denote the subtree of T_x rooted in y, and let $cs(T_x[y])$ denote the contiguous search number of $T_x[y]$ from y. The most important property of monotone strategies is given by the following lemma.

LEMMA 6. *Let $y_1, y_2, \ldots, y_k$ be the $k \geq 2$ children of y in T_x, and assume, w.l.o.g., that $cs(T_x[y_i]) \geq cs(T[y_{i+1}])$ for all $i < k$. Then*

$$cs(T_x[y]) = \max\{cs(T_x[y_1], cs(T_x[y_2]) + \omega(y)\}.$$

PROOF. Clearly $cs(T_x[y]) \geq cs(T_x[y_1])$, otherwise $T_x[y_1]$ cannot be cleared. If $cs(T_x[y_1]) > cs(T_x[y_2]) + \omega(y)$, then $cs(T_x[y_1])$ searchers suffice to clear $T_x[y]$ by visiting y_1 last among the children of x, and by letting $\omega(y)$ searchers occupying node y while the other subtrees are visited. Indeed, from Eq. 1, $\omega(\{y, y_i\}) \leq \omega(y_i) \leq cs(T_x[y_i])$ for every i.

Hence let $cs(T_x[y_1]) < cs(T_x[y_2]) + \omega(y)$. Let β be a contiguous search strategy which uses $b = cs(T_x[y_2]) + \omega(y) - 1$ searchers to clear $T_x[y]$. If $T_x[y_2]$ is cleared before $T_x[y_1]$, while the $cs(T_x[y_2])$ searchers are clearing $T_x[y_2]$, y will be occupied by at most $\omega(y) - 1$ searchers, and incident to $\{y, y_1\}$ which is contaminated. Thus β does not satisfy the condition of Theorem 1. Similarly, if $T_x[y_1]$ is cleared before $T_x[y_2]$, then, while the $cs(T_x[y_1])$ searchers are clearing $T_x[y_1]$, at most $\omega(y) - 1$ searchers will be at y since by definition $cs(T_x[y_1]) \geq cs(T_x[y_2])$, and $b = cs(T_x[y_2]) + \omega(y) - 1$.

Since y is incident to $\{y, y_2\}$ which is contaminated, y becomes unsafe, and β is hence not monotone. Therefore, strictly more than $cs(T_x[y_2])+\omega(y)-1$ searchers are required for a monotone strategy if $cs(T_x[y_1]) < cs(T_x[y_2])+\omega(y)$. On the other hand, $cs(T_x[y_2]) + \omega(y)$ searchers clearly suffice to clear $T_x[y]$ by visiting y_1 last among the children of y, since at least $\omega(y)$ will stay at y making it safe while the other subtrees are visited. Links incident to y are cleared since, again from Eq. 1, $\omega(\{y, y_i\}) \leq \omega(y_i) \leq cs(T_x[y_i])$ for every i. □

A straightforward application of Lemma 6 allows to compute $cs(T_x)$ in $O(n)$ time, resulting in an $O(n^2)$-time algorithm for computing $cs(T)$ for any tree T. We show that the complexity can be reduced to $O(n)$, and more importantly that a minimal search strategy can also be computed in linear time.

4.1 Labeling and Contiguous Search Numbers

Given a node $x \in V$, consider the following local labeling λ_x of the links incident on x. Let $e = \{x,y\}$ be a link incident to x. If y is a leaf, then $\lambda_x(e) = \omega(y)$. Otherwise, let $z_1, \dots, z_{\deg(y)-1}$ be the neighbors of y distinct from x. Let $\ell_i = \lambda_y(\{y, z_i\})$ where, w.l.o.g., $\ell_i \geq \ell_{i+1}$. Then

$$\lambda_x(e) = \max\{\ell_1, \ell_2 + \omega(y)\}.$$

Notice that every link $e = \{x, y\} \in E$ will be assigned two labels $\lambda_x(e)$ and $\lambda_y(e)$. We first prove that the labels assigned by the λ_x's can be computed in $O(n)$ work.

LEMMA 7. *All links can be labeled according to the λ_x's in sequential time $O(n)$, and distributively with $O(n)$ messages.*

PROOF. Distributively, it can be easily done with $O(n)$ messages using the *saturation* technique for trees (e.g., see [20]). Each message consists of a label ℓ. Initially, all nodes are *ready*. Each leaf x independently sends a message $\omega(x)$ to its only neighbor, and becomes *active*. A ready node x, upon receiving a message q from a neighbor y, will set $\lambda_x(\{x,y\}) = q$. When it has labelled all its incident links except one, say $\{x,z\}$, node x will then compute $\lambda_z(\{x,z\})$ based on the existing labels $\lambda_x(\{x,y\})$ for $y \neq z$. Then x sends $\lambda_z(\{x,z\})$ to z and becomes *active*. An active node x, upon receiving a message q from the neighbor z, sets $\lambda_x(\{x,z\}) = q$. For each neighbor $y \neq z$, x then computes $\lambda_y(\{x,y\})$ based on the existing labels and sends $\lambda_y(\{x,y\})$ to y. Then x becomes *done*. To prove the correctness of this algorithm, it is sufficient to notice that there is no starvation. Indeed, the converge-cast structure of the ready-to-active process insures that a ready node x will eventually receive $\deg(x) - 1$ messages and becomes active. The diverge-cast structure of the active-to-done process insures that an active node x will eventually complete and become done. To complete the proof, a serial implementation of the operations of this algorithm can be trivially done in $O(n)$ time. □

We now show the relationship between the λ_x's and the number of searchers of monotone contiguous search strategy.

LEMMA 8. *For each $e = \{x,y\} \in E$, $cs(T_x[y]) = \lambda_x(e)$.*

PROOF. By induction on the hight $h(y)$ of $T_x[y]$. The lemma trivially holds for $h(y) = 0$ (i.e., y is a leaf) since $\lambda_x(e) = \omega(y) = cs(T_x[y])$. Assume that the lemma holds whenever $0 \leq h(y) < k$, and consider now the case when $h(y) = k$. Let $y_1, y_2, \dots, y_d$ be the children of y in $T_x[y]$, where, w.l.o.g., $\lambda_y(\{y, y_i\}) \geq \lambda_y(\{y, y_{i+1}\})$, and $d = \deg(y) - 1$ if $y \neq x$ and $d = \deg(y)$ otherwise. By definition of λ_x,

$$\lambda_x(\{x,y\}) = \max\{\lambda_y(\{y,y_1\}), \lambda_y(\{y,y_2\}) + \omega(y)\}.$$

On the other hand, by Lemma 6,

$$cs(T_x[y]) = \max\{cs(T_x[y_1], cs(T_x[y_2]) + \omega(y)\}.$$

Since the hight of $T_y[y_i]$ is $h(y_i) < k$, then, by induction hypothesis,

$$\lambda_y(\{y, y_i\}) = cs(T_y[y_i]) = cs(T_x[y_i]).$$

Thus $\lambda_x(\{x,y\}) = cs(T_x[y])$. □

Consider now the following labeling of the nodes of T which will assign to each node $x \in V$ a label $\mu(x)$. Let $e_1, e_2, \dots, e_{\deg(x)}$ be the $\deg(x)$ links incident to x. Assume, w.l.o.g., that $\lambda_x(e_i) \geq \lambda_x(e_{i+1})$ for all $i < \deg(x)$. Set

$$\mu(x) = \max\{\lambda_x(e_1), \lambda_x(e_2) + \omega(x)\}.$$

By definition of μ and by Lemma 6, for all $x \in V$,

$$\mu(x) = cs_x(T). \tag{4}$$

In other words, $\mu(x)$ is exactly the minimum number of searchers required by any monotone strategy using x as the homebase. Since $cs(T) = \min_{x \in V} \mu(x)$, and since a direct consequence of Lemma 7 is that all $\mu(x)$'s can be computed in $O(n)$ work serially or distributively, we get that the contiguous search number of a tree can be compute serially in $\Theta(n)$ time, and distributively with $\Theta(n)$ messages. Interestingly, our algorithm not only computes the contiguous search number, but also provides a way to set up a minimal search strategy, still in linear time, as shown in the following section.

4.2 Optimal Determination of a Minimal Monotone Strategy

We now show that the labelings λ and μ contain all the information needed to construct a minimal monotone strategy α at optimal cost. Given a weighted tree T and the labelings λ and μ, consider the contiguous search strategy α using $x \in V$ as the homebase constructed as follows. Consider T_x, and at each node $y \in V$, order its children in T_x according to the increasing values assigned by λ_y to the corresponding links. A contiguous search of T_x can be performed by starting with $\mu(x)$ searchers in x and simply traversing T_x in pre-order, moving the searchers according to two simple rules: (1) when moving in the traversal from a node y to one of its child z, transfer $\lambda_y(\{y,z\})$ searchers from y to z; (2) when returning from z to y, transfer the $\lambda_y(\{y,z\})$ searchers from z back to y. (Recall that we have seen that $\mu(x) = cs(T_x)$ and $\lambda_y(\{y,z\}) = cs(T_x[z]) = cs(T_y[z])$.) A more detailed description is given in Figure 1, where the homebase is chosen among the nodes with minimum μ, and where a contiguous search strategy α starting from such a node is constructed.

SEARCH /* Returns a strategy α */

(a) Set $\alpha = \emptyset$; /* empty sequence */

(b) Choose a node x such that $cs(T_x)$ is minimum;
/* start with $cs(T)$ searchers at the homebase */

Let $y_1, y_2, \ldots, y_{\deg(x)}$ be the $\deg(x)$ neighbors of x where, w.l.o.g., $cs(T_x[y_i]) \geq cs(T_x[y_{i+1}])$;

(c) For $i = \deg(x)$ down to 1, apply
MOVE$(x, y_i, cs(T_x[y_i]))$;

MOVE (u, v, q) /* searching a subtree rooted at v */

(1) $\alpha = \alpha|(u, v, q)$ where "|" denotes the concatenation operation; /* transfer q searchers from u to v */

Let $w_1, \ldots, w_{\deg(v)-1}$ be the $\deg(v) - 1$ children of v in T_x where, w.l.o.g., $cs(T_x[w_i]) \geq cs(T_x[w_{i+1}])$;

(2) For $i = \deg(v) - 1$ down to 1, apply
MOVE$(v, w_i, cs(T_x[w_i]))$;

(3) $\alpha = \alpha|(v, u, q)$; /* return q searchers from v to u */

Figure 1: Optimal determination of a minimal contiguous search strategy

LEMMA 9. *The search strategy returned by the procedure* SEARCH *of Figure 1 is a minimal monotone contiguous search strategy.*

PROOF. Let x be the chosen homebase, and let

$$\alpha = \alpha_0, \alpha_1, \alpha_2, \ldots$$

be the search strategy returned by the procedure SEARCH. We will first prove that, after each move all traversed nodes are safe.

Assume that all traversed nodes are safe before move $\alpha_i = (u, v, q)$, and the number of searchers at u is $cs(T_x[u])$; observe that this statement clearly holds for $i = 1$ since initially $u = x$. We will show that, after the move, all traversed nodes are safe, and the number of searchers at v is $cs(T_x[v])$. We shall distinguish between two types of moves: "forward" and "return" moves, depending on whether v is visited by this move for the first time or not, respectively.

If α_i is a "return" move, v is the parent of u, it has already been traversed and it is thus safe by hypothesis. By construction, a "return" move is performed only after $T_x[u]$ has been traversed; hence, by hypothesis, also all the children of u are safe. Therefore α_i will keep all safe nodes safe. By construction, all the $q = cs(T_x[u])$ searchers originally moved from v to u will return to v in this move; since the traversal is sequential, all the other searchers (if any) originally moved to v are still there. Hence, after this move, at v there will be $cs(T_x[v])$ searchers.

Let α_i be a "forward" move. In this case, v is a child of u and, by construction, the number of searchers moved to v is $q = cs(T_x[v])$. Let us assume that u has children $z_1, z_2, \ldots, z_d$ where, w.l.o.g., $cs(T_x[z_i]) \geq cs(T_x[z_{i+1}])$, and where $d = \deg(u) - 1$ if $u \neq x$ and $d = \deg(u)$ otherwise. Let $v = z_k$. Since, by the rules of procedures SEARCH and MOVE, traversal is in pre-order, according to the increasing values associated to the children, this implies that all the other children z_l, $l > k$, have already been traversed. According to the strategy α, exactly $cs(T_x[v]) = cs(T_x[z_k])$ searchers are transferred from u to z_k in this move, which will transform z_k into occupied, and thus safe since $\omega(z_k)$ is certainly at most $cs(T_x[z_k])$. Moreover, the $cs(T_x[z_k])$ searchers clear link $\{u, z_k\}$ since, from Eq. 1, $\omega(\{u, z_k\}) \leq \omega(z_k)$, and hence $\omega(\{u, z_k\}) \leq cs(T_x[z_k])$.

If $k = 1$, then all other children (if any) are already traversed and, by assumption, safe. Thus, the move from u to $v = z_1$ will not make any currently safe node unsafe. Consider now the case $k > 1$. Since $cs(T_x[u]) \geq cs(T_x[z_2]) + \omega(u)$ then at least $\omega(u)$ searchers stay at u during this move (and until the return of those searchers). Hence, every traversed node will be safe after this move. Therefore, after each move, all safe nodes remain safe. Since all nodes will be traversed, α is a monotone strategy. Since it uses $cs(T_x) = cs(T)$ searchers, it is minimal. $\square$

To summarize, we have proven the following.

THEOREM 2. *The contiguous search number and a minimal monotone contiguous search strategy for trees can be found serially in $\Theta(n)$ time, and distributively with $\Theta(n)$ messages.*

PROOF. Both the identification of a node with minimal contiguous search number and the pre-order traversal of the labeled tree can be performed serially in $O(n)$ time, and distributively with $O(n)$ messages. Thus, by Lemma 7, the result follows. $\square$

As a final remark, observe that the labeling λ actually contains implicitly *every* minimal monotone strategy. It is possible to reconstruct each of them from (T, λ, μ), and the task is surprisingly not computationally vexing. In particular, we can easily compute the number of minimal monotone strategies, and efficiently generate all of them.

5. EXTREMAL CONTIGUOUS SEARCH NUMBERS

In this section we establish bounds on the contiguous search number in the worst case trees. For the sake of comparison with edge-search, we consider the unweighted case only. Let $\mathcal{T}_n$ be the set of all trees on n nodes, and let $cs(n) = \max_{T \in \mathcal{T}_n} cs(T)$. We first prove that there are trees $T \in \mathcal{T}_n$ with $cs(T) \simeq \log_2 n$ searchers. In contrast, note that it is known [9, 24] that $\log_3 n$ searchers suffice for edge-search in every $T \in \mathcal{T}_n$.

LEMMA 10. *For every $n \geq 1$, $cs(n) \geq \lfloor \log_2(\frac{2}{3}(n+1)) \rfloor$.*

PROOF. For any rooted tree, let us call the *level* of a node u its distance to the root. First, notice that if B is a complete binary tree of $2^k - 1$ nodes then for every pair of nodes $u, v \in B$, u of level $i > 0$ and v parent of u, $\lambda_v(\{v, u\}) = k - i$ and $\mu(u) = \lambda_u(\{u, v\}) = k - 1$. Therefore $\mu(x) = k$ where x is the root of B, and $\mu(u) = k - 1$ for every $u \neq x$. Therefore $cs(B) = k - 1 = \lfloor \log_2(\frac{2}{3}(n+1)) \rfloor$.

Let $n = \sum_{i=1}^{r} 2^{\alpha_i}$, with $\alpha_1 > \alpha_2 > \cdots > \alpha_r$. We construct a tree T of order n with the desired contiguous search number as follows. Let $P = x_1, x_2, \ldots, x_r$ be a path of length r. Add to every vertex x_i a link to a node y_i, which is the root of a complete binary tree of $2^{\alpha_i} - 1$ nodes, denoted by B_i. (If $\alpha_r = 0$, there is no vertex y_r.) Observe that this construction yields a complete binary tree if $n = 2^k - 1$.

Assume that T is not a complete binary tree. From the link-labeling λ of a complete binary tree, for every $u, v \in B_i$, u of level $\ell > 0$ and v parent of u, $\lambda_v(\{v, u\}) = \alpha_i - \ell$. In particular, $\lambda_{y_i}(\{y_i, u\}) = \alpha_i - 1$ for any u child of y_i in B_i, and thus $\lambda_{x_i}(\{x_i, y_i\}) = \alpha_i$. The links in the path P have the following labels:

- From the definition of λ and the fact that $\alpha_i > \alpha_{i+1}$, we have $\lambda_{x_{i+1}}(\{x_i, x_{i+1}\}) = \alpha_1$, for $1 \leq i \leq r - 1$.
- If $\alpha_r \neq 0$ then let $j = r$. Otherwise let j be the largest index i, $1 \leq i \leq r - 1$, such that $\alpha_i > \alpha_{i+1} + 1$. (Observe that j is well defined as T is not a binary tree.) If $j + 1 \leq i \leq r - 1$, then
$$\lambda_{x_i}(\{x_i, x_{i+1}\}) = \alpha_{i+1} + 1 = \alpha_i.$$
For $i = j$, we have
$$\lambda_{x_j}(\{x_j, x_{j+1}\}) = \alpha_{j+1} + 1 < \alpha_j.$$
Finally, if $1 \leq i \leq j - 1$ then
$$\lambda_{x_i}(\{x_i, x_{i+1}\}) = \alpha_{i+1}.$$
Note that:
$$\lambda_{x_1}(\{x_1, x_2\}) = \alpha_2 \text{ if } j \geq 2,$$
and
$$\lambda_{x_1}(x_1, x_2) = \alpha_2 + 1 \text{ if } j = 1.$$

It remains to compute the labels of the links going upward the binary trees B_i's. For $2 \leq i \leq r$, $\lambda_{y_i}(\{y_i, x_i\}) = \alpha_1$, and $\lambda_u(\{v, u\}) = \alpha_1$, with $u, v \in B_i$, v parent of u. Since $\deg(x_1) = 2$, then $\lambda_{y_1}(\{y_1, x_1\}) = \lambda_{x_1}(\{x_1, x_2\})$. Finally, if $\lambda_{x_1}(\{x_1, x_2\}) < \alpha_1 - 1$, then $\lambda_u(\{v, u\}) = \alpha_1 - 1$. Otherwise (i.e., if $\lambda_{x_1}(\{x_1, x_2\}) = \alpha_1 - 1$), we have $\lambda_u(\{v, u\}) = \alpha_1$.

As a consequence: for every node $u \notin B_1$, $\mu(u) = \alpha_1$; $\mu(y_1) = \alpha_1$; and, for every node $u \in B_1$, $u \neq y_1$, if
$$\lambda_{x_1}(\{x_1, x_2\}) < \alpha_1 - 1$$
then $\mu(u) = \alpha_1 - 1$, otherwise $\mu(u) = \alpha_1$. Therefore, $cs(T) = \alpha_1 - 1$, if $\lambda_{x_1}(\{x_1, x_2\}) < \alpha_1 - 1$, and $cs(T) = \alpha_1$ otherwise. From the value of $\lambda_{x_1}(x_1, x_2)$, we get: (1) if $\alpha_2 = \alpha_1 - 1$ then $cs(T) = \alpha_1$; (2) if $\alpha_2 = \alpha_1 - 2$ and $j = 1$ then $cs(T) = \alpha_1$; and (3) $cs(T) = \alpha_1 - 1$ otherwise. Easy calculations then give $cs(T) = \lfloor \log_2 \frac{2}{3}(n + 1) \rfloor$. $\square$

We now prove that $\lfloor \log_2 n \rfloor$ searchers are sufficient for any n-node tree. To do so, we show that binary trees are the worst case trees with respect to the contiguous search number. Given a tree $T \in \mathcal{T}_n$, choose a node x with $\mu(x) = cs(T)$ as root, and "simplify" T_x according to the following rules.

Simplification rules:

Rule 1. Starting from the root x:

If a node y has children $z_1, \ldots, z_k$, $k > 2$, ordered so that $cs(T_x[z_i]) \geq cs(T_x[z_{i+1}])$, then remove all $T_x[z_i]$ for $i > 2$.

Rule 2. Repeatedly, until no longer applicable:

2(a) If a node y has two children z_1 and z_2, with $cs(T_x[z_1]) > cs(T_x[z_2])$, then remove $T_x[z_2]$.

2(b) if a node $y \neq x$ has only one child z, then remove y and connect y's parent to z.

Rule 3. If there are more than two nodes left, and x has only one child y, then remove y and connect x to its grand-children.

We denote by B the tree resulting from the simplification of T_x.

We first note that B is a complete binary tree rooted in x. Indeed Rule 1 returns a tree whose nodes have at most two children. Rule 2(b) then returns a tree whose internal nodes have at least two children, excepted maybe for the root. Rule 3 then makes the root of degree 0 or 2. Hence B is a binary tree whose internal nodes have exactly two children. Assume for the purpose of contradiction that the tree is not complete. Then let y be a node such that the subtree rooted at y is not complete, and there is no node z in the subtree rooted at y such that the subtree rooted at z is not complete. Let T_1 and T_2 be the two subtrees rooted at the two children of y. By the choice of y, T_1 and T_2 are complete binary trees, but of different size. Hence, they have different contiguous search number, in contradiction with Rule 2(a).

We also note that
$$cs_x(B) = cs(T). \tag{5}$$
Indeed, only subtrees requiring strictly less than $cs(T)$ searchers are removed, and the removal of degree 1 node does not change the contiguous search number.

The two observations above allow to show the following:

LEMMA 11. *For every $n > 1$, $cs(n) \leq \lfloor \log_2 n \rfloor$.*

PROOF. For any $T \in \mathcal{T}_n$ and any node x with $\mu(x) = cs(T)$, we have that the order of B is $2^k - 1 < n$. By Equation 5, $cs(T) = cs_x(B) = k \leq \lfloor \log_2 n \rfloor$. $\square$

By Lemmas 10 and 11, we have:

THEOREM 3. *For every $n > 1$, the largest contiguous search number of n-node trees satisfies $\lfloor \log_2 n \rfloor - 1 \leq cs(n) \leq \lfloor \log_2 n \rfloor$.*

Actually we believe that the construction given in the proof of Lemma 10 is the worst possible with respect to the contiguous search number of trees.

We conclude this section by studying the worst case node in a given tree, i.e., the node x such that $cs_x(T) = \max_y cs_y(T)$. Surprisingly, all nodes behave roughly the same in regards to this criteria. More precisely:

THEOREM 4. *For every tree $T = (V, E)$, and every $x \in V$, $cs_x(T) - cs(T) \leq 1$.*

PROOF. Let $v \in T$ such that $\mu(v) = cs(T) = k$. We prove that for every $u \in T$, $\mu(u) = k$ or $\mu(u) = k + 1$. The result then follows from Equation 4. Let $v_1, \ldots, v_{\deg(x)}$ be the neighbors of v, and let $\ell_i = \lambda_v(\{v, v_i\})$ and $\ell'_i = \lambda_{v_i}(\{v, v_i\})$.

Assume, w.l.o.g., that $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_{\deg(v)}$. We consider two cases depending on whether $\ell_1 = \ell_2$ or not.

Case 1. Assume first that $\ell_1 = \ell_2 = k-1$. Since $\ell_i \leq k-1$ for $1 \leq i \leq \deg(v)$, we get $\lambda_{v_i}(\{v_i, u\}) \leq k-1$ for every neighbor $u \neq v$ of v_i. We have also that, for $3 \leq i \leq \deg(v)$, $\ell'_i = k$. Moreover, for $i = 1, 2$, $\ell'_1 = \ell'_2 = \max\{k-1, \ell_3+1\}$, i.e., $\ell'_1 = \ell'_2 = k-1$ or k. By removing the vertex v from T we obtain a tree T_i rooted in v_i, for every i. Let $u \in T_i$. Since $\ell_i \leq k-1$, $\lambda_u(\{u, w\}) \leq k-1$ for every w child of u in T_i. Also, $\lambda_u(\{w, u\}) \leq k$ for w parent of u in T_i. Combined with $cs(T) = k$, we get $\mu(u) = k$ for every $u \in T_i$. Therefore, $\mu(u) = k$ for all $u \in T$.

Case 2. Assume now that $\ell_1 = k$ and $\ell_2 < k$. By removing the link $\{v, v_1\}$ from T we obtain two trees T_v and T_{v_1} rooted in v and v_1 respectively. From the ℓ_i's, $1 \leq i \leq \deg(v)$, we get that $\ell'_2 = \cdots = \ell'_{\deg(v)} = k$ and thus, for all $u \in T_v$, $\mu(u) = k$. On the other hand, since $\ell_1 = k$, for every neighbor u of v_1, $\lambda_{v_1}(\{v_1, u\}) \leq k$. Moreover, either there is exactly one neighbor $u_1 \neq v$ of v_1 such that $\lambda_{v_1}(\{v_1, u_1\}) = k$, or there are at least two neighbors u_1, u_2 of v_1, both different from v, such that $\lambda_{v_1}(\{v_1, u_1\}) = \lambda_{v_1}(\{v_1, u_2\}) = k-1$.

– If $\ell'_1 < k$, then, since $\ell_1 = k$, there are at least two neighbors u_1, u_2 of v_1 such that $\lambda_{v_1}(\{v_1, u_1\}) = \lambda_{v_1}(\{v_1, u_2\}) = k-1$. By exchanging the roles of v_1 and v, we fall into the previous case $\ell_1 = \ell_2 = k-1$, implying $\mu(u) = k$ for all $u \in T$.

– If $\ell'_1 = k$ and $\lambda_{v_1}(\{v_1, u\}) < k$ for every neighbor $u \neq v$ of v_1, then, for all $u \in T_{v_1}$, $\mu(u) = k$, and hence $\mu(u) = k$ for all $u \in T$.

– It remains to consider the case where $\ell'_1 = k$ and there is exactly one neighbor $u_1 \neq v$ of v_1 such that $\lambda_{v_1}(\{v_1, u_1\}) = k$. By removing the link $\{v_1, u_1)\}$ from T_{v_1} we obtain two trees T_1 and T'_{v_1} rooted in u_1 and v_1, respectively. One can easily check that, for all $u \in T'_{v_1}$, $\mu(u) = k+1$, and $\lambda_{u_1}(u_1, v_1) = k$. The analysis for T_1 is analogous to the analysis for T_{v_1}. Now, either for every node $u \in T_1$, $\mu(u) = k$, or there is exactly one neighbor $u_2 \neq v_1$ of u_1 such that $\lambda_{u_1}(\{u_1, u_2\}) = k$. In the former case we are done. In the latter case, by removing the link $\{u_1, u_2\}$ from T_1, we get the trees T_2 and T'_{u_1} rooted in u_2 and u_1, respectively. We repeat the same arguments on T_2, and so on, constructing a sequence $v_1, u_1, u_2, \ldots$. Since T is finite, we will eventually find a node u_i such that, for every node $u \in T_i$, $\mu(u) = k$, which completes the proof. $\square$

6. CONCLUSION

This paper completely solves the contiguous search problem in trees, including its weighted version. This variant of the search problem is closer to practical applications than all variants known to the authors, in which searchers are allowed to be removed and placed from and to arbitrary nodes in the network, which is unrealistic in many practical cases. We have also studied extremal contiguous search, i.e., we have established bounds on the value of the contiguous search number in the worst case tree. In particular we have proved that the contiguous search number of any tree is at most $\lfloor \log_2 n \rfloor$, and the bound is tight up to an additive factor of 1. This bound has to be compared with the worst case edge-search number, which is known to be roughly $\log_3 n$ in trees (cf. [9, 24]), and suggests that contiguity induces a rather limited performance degradation in term of number of searchers.

Still, many problems remain open. In particular, very little is known on contiguous search in arbitrary networks. For instance, is it true that, for any weighted network, there is a minimal contiguous search strategy which is monotone? Also, can we establish a correspondence between the contiguous search number and standard graph parameters such as path-width, vertex-separation, etc.? We recently established a connection between the monotone contiguous search number and the *connected* vertex-separation.

An other interesting problem is to compare the edge-search number $s(G)$ with the contiguous search number $cs(G)$. For instance, it can be shown that $cs(G)/s(G) \leq 2$ for any G. Can we improve this bound?

Returning to trees, the weighted version of the problem does not include cases in which the safety requirements of nodes dynamically vary. For instance, an interesting case is when the weight of node x at time t is defined as $\omega(x,t) = \sum_e \omega(e)$ or $\omega(x,t) = \max_e \omega(e)$ where the summation and the maximum are taken over all links e incident to x that are contaminated at time t.

Acknowledgement. This work was partially done while the first and third authors were visiting Carleton University at Ottawa, and when the third author was visiting Univ. Politècnica de Catalunya (UPC). Additional support from Ottawa U., Carleton, NATO, and the French-Spanish research program PICASSO.

7. REFERENCES

[1] M. Asaka, S. Okazawa, A. Taguchi, and S. Goto. A method of tracing intruders by use of mobile agents. In 9th *Annual Conference of the Internet Society* (INET'99), 1999. (*http://www.isoc.org/*)

[2] D. Bienstock. Graph searching, path-width, tree-width and related problems. *DIMACS Series in Disc. Maths. and Theo. Comp. Sc.*, Vol. 5, 33–49, 1991.

[3] D. Bienstock and P. Seymour. Monotonicity in graph searching. *Journal of Algorithms* 12, 239–245, 1991.

[4] D. Bienstock and M. Langston. Algorithmic implications of the graph minor theorem. *Handbooks in OR & MS*, Vol. 7, Chapter 8, 481–502, Elsevier Science, 1995.

[5] R. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers* VI(5), 72–78, 1967.

[6] H. Buhrman, M. Franklin, J. Garay, J.-H. Hoepman, J. Tromp, and P. Vitányi. Mutual search. *Journal of the ACM* 46(4), 517–536, 1999.

[7] R. Chang. Single step graph search problem. *Information Processing Letters*, 40(2):107–111, 1991.

[8] N. Dendris, L. Kirousis, and D. Thilikos. Fugitive-search games on graphs and related parameters. *Theoretical Computer Science*, 172(1–2):233–254, 1997.

[9] J. Ellis, H. Sudborough, and J. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

[10] M. Fellows and M. Langton. On search, decision and the efficiency of polynomial time algorithm. In 21st *ACM Symp. on Theory of Computing* (STOC '89), pp. 501-512, 1989.

[11] J. Fernandez and J. Gonzalez. Hierarchical graph search for mobile robot path planning. In *Int. IEEE Conference on Robotics and Automation* (ICRA '98), 656–661, 1998.

[12] F. Fomin and P. Golovach. Graph searching and interval completion. *SIAM Journal on Discrete Mathematics* 13(4), 454–464, 2000.

[13] N. Foukia,J. G. Hulaas and J. Harms. Intrusion Detection with Mobile Agents. In 11th *Annual Conference of the Internet Society* (INET '01), 2001. (*http://www.isoc.org/*)

[14] S. Hansen and M. Eldredge. Intruder isolation and monitoring. In 1st *USENIX Security Workshop*, 63–64, 1988.

[15] G. G. Helmer, J. S. K. Wong, V. Honavar, and L. Miller. Intelligent agents for intrusion detection. In *IEEE Information Technology Conference*, 121–124, 1998.

[16] W. Jansen, P. Mell, T. Karygiannis and D. Marks, Mobile agents in intrusion detection and response. In 12th *Annual Canadian Information Technology Security Symposium*, 2000.

[17] N. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992.

[18] L. Kirousis and C. Papadimitriou. Interval graphs and searching. *Discrete Mathematics* 55, 181–184, 1985.

[19] L. Kirousis and C. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, 1986.

[20] E. Korach, D. Rotem and N. Santoro. Distributed algorithms for finding centers and medians in networks. *ACM Trans. on Programming Languages and Systems*, 6(3):380–401, 1984.

[21] A. Lapaugh. Recontamination does not help to search a graph. *Journal of the ACM* 40(2):224–245, 1993.

[22] T. Lengauer. Black-white pebbles and graph separation. *Acta Informatica*, 16(4):465–475, 1981.

[23] F. Makedon and H. Sudborough. Minimizing width in linear layout. In 10th *Int. Colloquium on Automata, Languages, and Programming* (ICALP '83), LNCS 154, Springer-Verlag, 478–490, 1983.

[24] N. Megiddo, S. Hakimi, M. Garey, D. Johnson and C. Papadimitriou. The complexity of searching a graph. *Journal of the ACM* 35(1):18–44, 1988.

[25] S. Neufeld. A pursuit-evasion problem on a grid. *Information Processing Letters*, 58(1):5–9, 1996.

[26] T. Parson. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, Lecture Notes in Mathematics, Springer-Verlag, 426–441, 1976.

[27] T. Parson. The search number of a connected graph. In the 9th *Southeastern Conference on Combinatorics, Graph Theory and Computing*, Utilitas Mathematica, 549–554, 1978.

[28] P. Seymour and R. Thomas. Graph searching, and a min-max theorem for treewidth. *Jour. Combin. Theory, Ser. B*, to appear.

[29] K. Skodinis. Computing optimal linear layouts of trees in linear time. In 8th *European Symp. on Algorithms* (ESA '00), Springer, LNCS 1879, pages 403-414, 2000.

[30] J. Smith. Minimal trees of given search number. *Discrete Mathematics* 66, 191–202, 1987.

[31] E. H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570, October 2000.

[32] Y. Stamatiou and D. Thilikos. Monotonicity and inert fugitive search games. In 6th *Twente Workshop on Graphs and Comb. Opt.*, Elsevier, 1999.

[33] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992.

[34] I. Suzuki, M. Yamashita, H. Umemoto, and T. Kameda. Bushiness and a tight worst-case upper bound on the search number of a simple polygon. *Information Processing Letters*, 66(1):49–52, 1998.

[35] A. Takahashi, S. Ueno, and Y. Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995.

[36] D. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics* 105, 239–271, 2000.

[37] G. Vigna, R.A. Kemmerer. NetSTAT: A network based intrusion detection system. *J. of Computer Security*, 7(1):37–71,1999.

[38] B. von Stengel and R. Werchner. Complexity of searching an immobile hider in a graph. *Discrete Applied Mathematics*, 1996.

[39] M. Yamamoto, K. Takahashi, M. Hagiya, and S.-Y. Nishizaki. Formalization of graph search algorithms and its applications. In *Theorem Proving in Higher Order Logics* (TPHOLs '98), Springer-Verlag, LNCS 1479, 479–496, 1998.