Connected and Internal Graph Searching

Lali Barrière[*] Pierre Fraigniaud[†] Nicola Santoro[‡] Dimitrios M. Thilikos[§]

Abstract

This paper is concerned with the *graph searching* game. The *search number* $s(G)$ of a graph G is the smallest number of searchers required to "clear" G. A search strategy is *monotone* (m) if no recontamination ever occurs. It is *connected* (c) if the set of clear edges always forms a connected subgraph. It is *internal* (i) if the removal of searchers is not allowed. The difficulty of the "connected" version and of the "monotone internal" version of the graph searching problem comes from the fact that, as shown in the paper, none of these problems is minor closed for arbitrary graphs, as opposed to all known variants of the graph searching problem. Motivated by the fact that connected graph searching, and monotone internal graph searching are both minor closed *in trees*, we provide a complete characterization of the set of trees that can be cleared by a given number of searchers. In fact, we show that, in trees, there is *only one* obstruction for monotone internal search, as well as for connected search, and this obstruction is the same for the two problems. This allows us to prove that, for any tree T, $mis(T) = cs(T)$. For arbitrary graphs, we prove that there is a unique chain of inequalities linking all the search numbers above. More precisely, for any graph G, $s(G) = is(G) = ms(G) \leq mis(G) \leq cs(G) = ics(G) \leq mcs(G) = mics(G)$. The first two inequalities can be strict. In the case of trees, we have $mics(G) \leq 2\ s(T) - 2$, that is there are exactly 2 different search numbers in trees, and these search numbers differ by a factor of 2 at most.

1 Introduction

Imagine a group of $k + 1$ friends visiting a large bookstore, and assume that one of them gets separated from the k others, who are now looking for him in the store. The search for the lost friend is made difficult by the placement of the shelves, and by the complex topology of the store occupying several buildings, on several floors, connected by many stairs and bridges. It is also made difficult by the behavior of the lost friend who, as opposed to what is recommended in this situation, starts moving sporadically in the store, also looking for his friends. The question that arises among the group of k "searchers" is whether they are enough to eventually find their "fugitive" friend. For instance, if the bookstore would be displayed as a path, then $k = 1$ searchers would be enough. But if the bookstore is displayed as a ring, then 2 searchers are required, otherwise the fugitive could perpetually escape from the searcher. Let G be the graph corresponding to the map of the bookstore. We address the problem of computing the minimum number of searchers required to find the fugitive in G. This problem is known as the *graph searching* problem. However, the

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

[†]CNRS, Laboratoire de Recherche en Informatique, Université Paris-Sud, France. *http://www.lri.fr/~pierre*.

[‡]School of Computer Science, Carleton University, Canada. *santoro@scs.carleton.ca*.

[§]Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain. *http://www.lsi.upc.es/~sedthilk*.

search strategy developed by our group of friends must satisfy additional properties. It must be "internal", in the sense that searchers must follow the corridors of the bookstore, as they cannot jump over the shelves, nor pass through the walls. The strategy must also be "monotone" in the sense that searchers don't want to check several times the same part of the bookstore. It must also be "connected" as searchers certainly prefer not to split in several groups that could lose contact from each other.

The searchers consult the literature available at the bookstore. They find that, according to Lapaugh's theorem [21], monotonicity can be assumed for free, i.e., if k searchers can find the fugitive, then they can find it according to a monotone strategy. However, this monotone strategy is not necessarily internal nor connected. Hence, the group of searchers start doubting whether the classic definition of graph searching is realistic. Subsequently they start wondering how much it costs (in term of number of searchers) to impose internality. In this paper, we show that it does not cost more than imposing connectness, that is if k searchers can find the fugitive according to a connected strategy, then they can find it according to a monotone internal strategy. This paper studies more thoroughly the relationships between monotone, internal, and connected search strategies.

1.1 Statement of the problem

More formally, we are given a graph whose edges are all "contaminated", and a set of "searchers". The goal is to obtain a state of the graph in which all edges are simultaneously "clear". To clear an edge $e = (u, v)$, a searcher must traverse the edge from one end-point u to the other end-point v. A clear edge is preserved from recontamination if either another searcher remains in u, or all other edges incident to u are clear. In other words, a clear edge e is recontaminated if there exists a path between e and a contaminated edge, with no searcher on any node of the path. In the standard setting of the graph searching problem, the basic operations, called *search steps*, can be the following:

(1) place a searcher on a node,

(2) move a searcher along an edge,

(3) remove a searcher from a node.

Graph searching is the problem of developing a *search strategy*, that is a sequence of search steps that results in all edges being simultaneously clear. The smallest number of searchers for which a search strategy exists for a graph G is called the *search number* $s(G)$ of G. As far as practical applications are concerned (e.g., decontaminating a set of tunnels, capturing an intruder in a network, rescuing a speleologist in a maze of caves, etc.), the line of investigation is the determination of efficient search strategies satisfying additional properties, which are desirable or even necessary for some applications. Three properties are of particular interest.

Internal Search. A search strategy is *internal* if, once placed, searchers can only move along the graph edges (i.e., they cannot be removed and placed somewhere else). It is easy to see that this is equivalent to the case where operation (3) is not allowed. The minimum number of searchers for which an internal search strategy exists is denoted by $is(G)$.

Monotone search. A search strategy is *monotone* if no recontamination ever occurs. Hence each edge should be cleared only once. The minimum number of searchers for which a monotone search strategy exists is denoted by $ms(G)$.

Connected search. A search strategy is *connected* if the set of clear edges always induces a connected subgraph. Alternatively, one can define such strategies by not allowing operation (3), and allowing (1) only in the beginning of the search or when applied to vertices incident to an already clean edge. The minimum number of searchers for which a connected search strategy exists is denoted by $cs(G)$.

Obviously, for any G, $is(G) = s(G)$ because the removal of a searcher (operation (3)) from a node u, and its placement (operation (1)) later at node v, can be replaced by a sequence of moves (operation (2)) from u to v. Lapaugh's theorem [21] says that, for any G, $ms(G) = s(G)$, that is recontamination does not help. Interestingly, for internal search, recontamination does help, i.e., there are graphs G for which $is(G) < mis(G)$, for instance the 22-node tree T^* obtained from three copies of the complete binary tree of depth 2, by joining their roots to a new vertex. Similarly, it is easy to check that $s(T^*) < cs(T^*)$, that is non-connectness helps too. Now, it can be desirable to mix the three properties monotonicity, internality, and connectness, resulting in the search numbers mis, mcs, ics, and $mics$. Obviously, for any G, $cs(G) = ics(G)$, and $mcs(G) = mics(G)$ because once a strategy is connected, it is easy to make it internal as well, searchers moving freely inside the connected component. It is known [1] that, for trees, all the four numbers cs, mcs, ics, and $mics$ collapse into one because $mcs(T) = cs(T)$ for any tree T, i.e., recontamination does not help for connected search *in trees*. Surprisingly, unlike the case of monotone strategies for which there exist detailed studies and characterizations, very little is known about connected search strategies and monotone internal strategies. Unfortunately, the existing techniques and results for the many variants of the problem (cf. Section 1.3) not only cannot be employed but do not even provide any direct insight on these two important properties. We claim that the reason for that is that neither monotone internal search, nor connected search, is minor closed for arbitrary graphs, as opposed to all known variants of the graph searching problem.

1.2 Our results

In this paper, we first prove a strong difference between traditional search and both connected and monotone internal searches, that is none of these latter versions of the problem is minor closed. Nevertheless, we show that there is a unique chain of inequalities linking all the search numbers above. More precisely, for any graph G,

$$s(G) = is(G) = ms(G) \leq mis(G) \leq cs(G) = ics(G) \leq mcs(G) = mics(G).$$

The first two inequalities can be strict. In particular, $mis \neq cs$. To obtain these results, we extend the notion of crusades defined by Bienstock and Seymour [3], and use it in a novel way. In fact, we employ it not to prove monotonicity, but to transform a connected strategy into a monotone internal one with the same number of searchers. On the other hand, it is easy to see that for all mentioned search problems, the class of *trees* that can be cleared with up to k searchers is minor closed. Therefore, the figure can be more precisely stated. We prove that, for any tree T,

$$cs(T) \leq 2\, s(T) - 2, \tag{1}$$

that is, for any tree T there exists a *monotone connected internal* search strategy for T using at most $2\, s(T) - 2$ searchers. We show that the upper bound of Eq. (1) is tight. We also show that

$$mis(T) = cs(T)$$

for any tree T. This and the result in [1] imply that there are exactly 2 different search numbers in total for trees. These search numbers differ by a factor of 2 at most. We summarize the situation for trees by the inequalities chain

$$s(T) = is(T) = ms(T) \leq mis(T) = cs(T) = ics(T) = mcs(T) = mics(T) \leq 2\, s(T) - 2.$$

We provide a complete characterization of the set of trees that can be cleared by k searchers. This characterization is given both explicitly, in terms of *k-caterpillars* (related to the notion of caterpillar dimension of [24]), and implicitly in terms of minimal forbidden minors. In fact, we show that, in trees, there is *only one* obstruction for monotone internal search, as well as for connected search, and the obstructions for the two problems are identical. This must be contrasted with the fact that, for traditional search, the number of obstructions in trees is *super-exponential* in the number of searchers [28, 37].

1.3 Previous works

Graph searching refers to a problem that has been throughly and extensively investigated in the literature, and that describes a variety of application scenarios [8, 28, 29]. In particular, it arises in VLSI design, through its equivalence with the gate matrix layout problem (see, e.g., [11, 13, 22]). It is also related to network security for its relation with the capture of an intruder by software agents (see, e.g., [1, 17, 35]), and protection from mobile eavesdroppers [16]. Moreover, the problem and its variants, i.e., *node-search*, *mixed-search*, *inert-search*, etc., are closely related to standard graph parameters and concepts, including treewidth, cutwidth, pathwidth, and linear-width [2, 39]. For instance, $s(G)$ is equal to the cutwidth of G for all graphs of maximum degree 3 (see [25]), and is equal to the vertex separation of the 2-expansion of G for all graphs (see [12]). Similarly, the node-search number of a graph is equal to its pathwidth plus one, and also to its vertex separator plus one [18, 19, 20]. The inert-search number is equal to the treewidth plus one [10, 32], and the mixed-search number is equal to the proper pathwidth [38, 39]. For more on graph searching, we refer the reader to, e.g., [9, 14, 15, 36].

Determining whether $s(G) \leq k$ for arbitrary G and k, is NP-complete [26]. Not surprisingly, the research has focused on restricted classes of graphs (e.g., [19, 25, 27, 33, 34]), and on bounded search numbers (e.g., see [6, 28, 37, 39]). In particular, for any fixed k, the class of graphs that can be cleared with up to k searchers is minor closed. Therefore, there is a finite number of *obstructions* for this class [31]. A consequence if that is the existence of a polynomial-time algorithm for testing whether an arbitrary graph G satisfies $s(G) \leq k$, for a fixed k. Of course, the algorithm requires the knowledge of the whole set of obstructions. Unfortunately, the number of obstructions for search grows super-exponentially with k, even for trees [28, 37]. More precisely, for any k, there are at least $(k!)^2$ obstructions for the class of trees T such that $s(T) \leq k$.

The importance of monotone searching arises in applications where the cost of clearing an edge by far exceeds the cost of traversing an edge. Lapaugh [21] has proved that for every G there is always a monotone search strategy that uses $s(G)$ searchers. A similar positive result exists also for node-search and mixed-search [2, 3]. The necessity for connectness arises, e.g., in applications where communication between the searchers can occur only within completely clear areas of the network. Hence connectivity is required for their coordination. Safety is another motivation for connectness, as it would always ensure the presence of secure routes between all the searchers. The problem of determining minimal search strategies under the connectness and/or the internality constraint is still NP-complete in general (it follows from the reduction in [26], as observed in [1]).

It has been shown in [1] that minimal connected strategies can however be computed in linear time for trees. The removal of a searcher from a node x, and the placement of this searcher in another node y, might be difficult or impossible to implement. In fact, it assumes that a searcher is able to go "out of the system" and to reenter the system elsewhere. This assumption is clearly unrealistic e.g., in the case of software mobile agents. In this case the searchers can only move in the network from site to neighboring site. Actually, it does not hold even in the original setting of a maze of caves [28]. Hence the importance of internal search strategies. There are trees for which minimal internal search strategies require $\Omega(n \log n)$ moves (i.e., edge traversal) [26], whereas, if the removal of searchers (and their arbitrary placement somewhere else) is allowed, then, for any graph G, there exists a minimal search strategy that requires at most $O(n)$ moves in G [21].

2 Connected vs. Monotone Internal Graph Searching

In this section, we show the following.

Theorem 2.1 *For every graph G, $mis(G) \leq cs(G)$.*

To prove this result, we use a generalization of the concept of *crusade* introduced in the short and elegant proof of Bienstock and Seymour [3] of Lapaugh's Theorem. For a set X of edges in a graph G, we denote by $\delta(X)$ the set of nodes in G having at least one incident edge in X, and at least one incident edge not in X.

Definition 2.1 [3] *Given a graph $G = (V, E)$, a sequence $(X_0, X_1, \ldots, X_r)$ of subsets of edges 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 $\max_{1 \leq i \leq r} |\delta(X_i)|$. A crusade is* progressive *if $X_0 \subseteq X_1 \subseteq \ldots \subseteq X_r$ and $|X_i \setminus X_{i-1}| = 1$ for $1 \leq i \leq r$.*

We say that a crusade is *connected* if the subgraph induced by X_i is connected for any $1 \leq i \leq r$.

Lemma 2.1 *If $cs(G) \leq k$ then there exists a connected crusade of frontier at most k in G.*

Proof. Given a search strategy S in a graph G, let $C = (X_0, X_1, \ldots, X_r)$ be the sequence of subsets of edges such that $X_0 = \emptyset$, and X_i is the set of clear edges after step i of S. At most one edge 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 search strategy in G using at most k searchers, then obviously the frontier of C is at most k. All X_i, $1 \leq i \leq r$, are connected by definition of connected search. ■

Given a crusade $C = (X_0, X_1, \ldots, X_r)$, we define the *skeleton* $\mathcal{S}$ of C as the directed graph of $r + 1$ levels L_i, $i = 0, \ldots, r$ such that L_i consists of as many nodes as the number of connected components of X_i. There are edges only between levels of consecutive indices in $\mathcal{S}$ (i.e., each L_i forms a stable). More precisely, there is an edge from the node $a \in L_i$, representing a connected component A of X_i, to the node $b \in L_{i+1}$, representing a connected component B of X_{i+1}, if and only if one of the three following properties holds.

- $X_{i+1} \setminus X_i = \emptyset$, and $B \subseteq A$;
- $X_{i+1} \setminus X_i = e_i \notin B$ and $B \subseteq A$;

- $X_{i+1} \setminus X_i = e_i \in B$ and one of the (at most two) connected component(s) of $B \setminus \{e_i\}$ is included in A.

Note that the out-degree of a node in $\mathcal{S}$ can be greater than 1 because a connected component of X_i can split in several connected components of X_{i+1} due to recontamination. On the other hand, the in-degree of a node is at most 2 because $|X_{i+1} \setminus X_i| \leq 1$ (i.e., there is at most one new clear edge in X_{i+1}). Hence at most two distinct connected components of level i can form a unique component at level $i+1$. More precisely, there is at most one node of in-degree 2 at every level of $\mathcal{S}$, and all the other nodes have in-degree ≤ 1. Note also that all nodes in a skeleton of a progressive crusade have out-degree 1 because $X_i \subseteq X_{i+1}$, and hence a connected component never splits.

We denote by $\Gamma^+(u)$ (resp., $\Gamma^-(u)$) the set of edges in $\mathcal{S}$ out-going from (resp., incoming to) node $u \in \mathcal{S}$. A node $u \in \mathcal{S}$ represents a set of edges X in G. Hence, by extension, we denote by $\delta(u)$ the set of nodes in $\delta(X)$.

We now define the concept of *consistent* crusade.

Definition 2.2 *A crusade C is k-*consistent *if its frontier is at most k, and every node u (resp., edge e) of its skeleton $\mathcal{S}$ can be labeled by a positive integer k_u (resp., k_e) satisfying:*
(1) $k_u \geq |\delta(u)|$ for every $u \in \mathcal{S}$;
(2) $\sum_{u \in L_i} k_u \leq k$ for every level L_i;
(3) $\sum_{e \in \Gamma^+(u)} k_e = k_u \geq \sum_{e \in \Gamma^-(u)} k_e$.

Intuitively, k_u represents the number of searchers in the connected component represented by u. The labels k_e, $e \in \Gamma^+(u)$, represent how the searchers are distributed among the possibly many connected components resulting from a split of the component represented by u. Condition (1) states that the number of searchers in each component is sufficient to protect the component from recontamination. Condition (2) states that the total number of searchers in the graph at any step of a search strategy cannot exceed k. Condition (3) states that: on one hand, a connected component A of X_i shares its searchers among the connected component(s) resulting from the split of A in X_{i+1}. On the other hand, it also states that the number of searchers in a connected component B of X_{i+1} is equal to the sum of searchers coming from component(s) of X_i whose merging results in B.

Observe that the skeleton $\mathcal{S}$ of a connected crusade C is a path. Labeling every node and edge of $\mathcal{S}$ by k makes it k-consistent. Therefore, a connected crusade of frontier at most k is k-consistent. The main reason for introducing consistent crusades is actually the following lemma.

Lemma 2.2 *If there exists a k-consistent crusade in G, then there exists a progressive k-consistent crusade in G.*

Proof. The proof is inspired by (2.2) in [3]. Among all k-consistent crusades, choose a k-consistent crusade $C = (X_0, X_1, \ldots, X_r)$ which satisfies:

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

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

Let us show that this crusade is progressive.

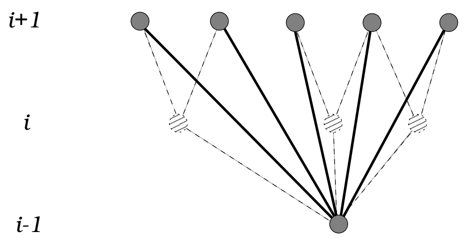


Figure 1: Skeleton $\mathcal{S}'$.

Claim 2.1 *$|X_i \setminus X_{i-1}| = 1$ for every $i \geq 1$.*

For the purpose of contradiction, let i be such that $|X_i \setminus X_{i-1}| = 0$, 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 crusade of frontier $\leq k$. Let us show that C' is k-consistent.

In a skeleton, the out-neighbors of a node u are called the children of u, and the out-neighbors of the children of u are called its grandchildren. We define similarly the notion of parents and grandparents.

The skeleton $\mathcal{S}'$ of C' can be obtained from the skeleton $\mathcal{S}$ of C by removing level i, and connecting every node of L_{i-1} to its grandchildren in $\mathcal{S}$ (see Figure 1).

We show that we can label $\mathcal{S}'$ so that the three conditions of Definition 2.2 are satisfied. The node-labeling of $\mathcal{S}'$ is the node-labeling of $\mathcal{S}$. The edge-labeling of $\mathcal{S}'$ is the edge-labeling of $\mathcal{S}$, but between L_{i-1} and L_{i+1}. Edges from L_{i-1} to L_{i+1} are labeled as follows. Let $v \in L_{i+1}$. Let u be a grandparent of v in $\mathcal{S}$. There can be at most two distinct paths from u to v in $\mathcal{S}$ because the in-degree of v is at most 2. The edge (u, v) of $\mathcal{S}'$ receives the label of (w, v) of $\mathcal{S}$ if there is a unique path (u, w, v) from u to v in $\mathcal{S}$. It receives the sum of the labels of (w, v) and (w', v) if there are two paths (u, w, v) and (u, w', v) from u to v in $\mathcal{S}$. Since $|X_i \setminus X_{i-1}| = 0$, there is no node of in-degree 2 in L_i of $\mathcal{S}$, and thus this labeling gives k-consistency to $\mathcal{S}'$.

Hence C' is a k-consistent crusade contradicting (C1). Therefore Claim 2.1 holds, i.e., $|X_i \setminus X_{i-1}| = 1$ for every $i \geq 1$.

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

Claim 2.2 *$|\delta(X_{i-1} \cup X_i)| \geq |\delta(X_i)|$ for every $i \geq 1$.*

For the purpose of contradiction, assume that there exists i such that $|\delta(X_{i-1} \cup X_i)| < |\delta(X_i)|$, and let

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

C'' is a crusade of frontier $\leq k$. Let us show that it is k-consistent.

The skeleton $\mathcal{S}''$ of C'' can be obtained from the skeleton $\mathcal{S}$ of C by replacing L_i by a copy L'_i of L_{i-1}, and by placing edges from each node in L_{i-1} to its copy L'_i (see Figure 2). If the edge $X_i \setminus X_{i-1}$ merges two components of X_{i-1}, then the corresponding two nodes of L'_i are merged into one node. Finally, there is an edge from node $u' \in L'_i$ to all the grandchildren (in $\mathcal{S}$) of its copy $u \in L_{i-1}$. In Figure 2, there are three nodes represented at level L_{i-1}. There are six nodes in L_i

which are replaced by three copies of the three nodes of L_{i-1}. The two copies of u' and u'' are merged into a single node u because the two components corresponding to u' and u'' are connected by $X_i \setminus X_{i-1}$.

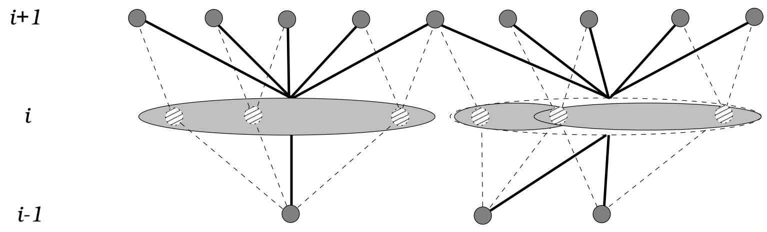


Figure 2: Skeleton $\mathcal{S}''$.

We show that we can label $\mathcal{S}''$ so that the three conditions of Definition 2.2 are satisfied. The node-labeling of $\mathcal{S}''$ is the node-labeling of $\mathcal{S}$ for all nodes of levels $j \neq i$. If a node $u \in L'_i$ does not results from the merging of two nodes u' and u'', then u receives the label of its original in L_{i-1}. Otherwise u receives the sum of the labels of the originals of u' and u'' in L_{i-1}. The edge-labeling of $\mathcal{S}'$ is the edge-labeling of $\mathcal{S}$ except between levels $i-1$, i, and $i+1$. The out-going edge of any node u of L_{i-1} receives label k_u. The setting of the edge-labeling between levels i and $i+1$ is slightly more complex. (Recall that there is at most one node of in-degree 2 at every level.) At levels i and $i+1$, there is a one-to-one correspondence between incoming edges to nodes with in-degree 1 in $\mathcal{S}''$ and incoming edges to nodes with in-degree 1 in $\mathcal{S}$. Thus, the edge incoming to a node at level $i+1$, with in-degree 1 in $\mathcal{S}''$, receives the label of the corresponding edge in $\mathcal{S}$. Let v be a node at level $i+1$ of $\mathcal{S}$, with in-degree 2. If v is still of in-degree 2 in $\mathcal{S}''$ (like in Figure 2), then the two incoming edges of $\mathcal{S}''$ take the same labels as the corresponding edges in $\mathcal{S}$. Otherwise, the unique incoming edge to node v in $\mathcal{S}''$ takes the sum of the labels of the two edges incoming to node v in $\mathcal{S}$. One can easily check that this labeling gives k-consistency to $\mathcal{S}''$.

Hence, C'' is a k-consistent crusade, in contradiction with (`C1`). Therefore Claim 2.2 holds, i.e., for every $i \geq 1$, $|\delta(X_{i-1} \cup X_i)| \geq |\delta(X_i)|$.

Now, for any two edge-sets A and B, $|\delta(A \cap B)| + |\delta(A \cup B)| \leq |\delta(A)| + |\delta(B)|$ because every node appearing on the left hand side contributes at least as many times on the right hand side. Thence, we get from Claim 2.2 that $|\delta(X_{i-1} \cap X_i)| \leq |\delta(X_{i-1})|$ for any $i \geq 1$. Hence, let

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

C''' is a crusade of frontier at most k.

Claim 2.3 *C''' is k-consistent.*

The skeleton $\mathcal{S}'''$ of C''' can be obtained from the skeleton $\mathcal{S}$ of C by replacing L_{i-1} by a copy L'_{i-1} of L_i, and by placing an edge from every copy of a node in L'_{i-1} to its original in L_i, with the following modification (see Figure 3): (1) the node x with in-degree 2 at level i of $\mathcal{S}$ (if any) has two copies in L'_{i-1}; (2) each copy is connected to x by an edge. We describe now the edges between level L_{i-2} and L'_{i-1}. Let w be a node of $\mathcal{S}$ at level L_{i-1}. If w has in-degree 1, then the parent of w is connected in $\mathcal{S}'''$ to all the copies of the children of w. If w has in-degree 2 (as in Figure 3), then let u and v be the two parents of w in $\mathcal{S}$, and let $x_1, \ldots, x_p$, $p \geq 0$, be the children of w in $\mathcal{S}$. If $p > 1$, then the connected component w resulting from the merging of two sub-components u and v is split into pieces in strategy $\mathcal{S}$. The merging is induced by the unique new edge $e_{i-1} = X_{i-1} \setminus X_{i-2}$. Therefore, if w splits into subcomponents, there is at most one subcomponent w' of w that contains e_{i-1}. Any other component is either a subcomponent of u or a subcomponent of v, but not both. Hence, we set an edge from u (resp., v) only to the copies of the x_j's which are subcomponents of u (resp., v). Note that w' may have in-degree 1 or 2.

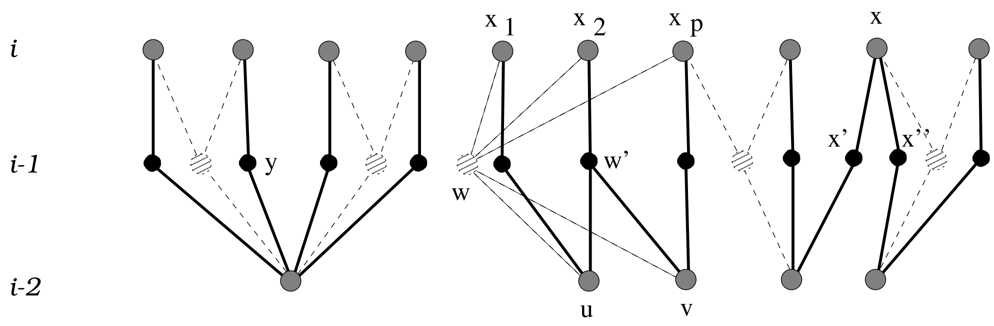


Figure 3: Skeleton $\mathcal{S}'''$.

Let us now show that we can label $\mathcal{S}'''$ so that the three conditions of Definition 2.2 are satisfied. The node-labeling of $\mathcal{S}'''$ is the node-labeling of $\mathcal{S}$, except for level L'_{i-1}. The edge-labeling of $\mathcal{S}'''$ is the edge-labeling of $\mathcal{S}$, except between levels $i-2$, $i-1$, and i. A copy $u' \in L'_{i-1}$ of a node $u \in L_i$ with in-degree 1 receives label $k_{u'} = k_u$. Each of the two copies x' and x'' of node x of L'_i with in-degree 2 receive labels that will be specified later. There is a one-to-one correspondence between the edges incoming to level i in $\mathcal{S}'''$ and in $\mathcal{S}$. An edge incoming to L_i in $\mathcal{S}'''$ receives the label of its corresponding edge in $\mathcal{S}$. We set $k_{x'}$ and $k_{x''}$ as the label of the edges (x', x) and (x'', x), respectively. The edge incoming to a node y of in-degree 1 in L'_{i-1} receives the label k_y. The edge $f = (u, w')$ incoming to the node w' of degree 2 in L'_{i-1} receives label $k_u - \sum_{e \in \Gamma^+(u), e \neq f} k_e$. Similarly, the edge $f' = (v, w')$ receives label $k_v - \sum_{e \in \Gamma^+(v), e \neq f'} k_e$. One can check that this labeling gives k-consistency to $\mathcal{S}'''$. Therefore Claim 2.3 holds.

From (C2), we then get $|X_{i-1} \cap X_i| \geq |X_{i-1}|$, that is $X_{i-1} \subseteq X_i$. Therefore C is a progressive k-consistent crusade, which completes the proof. ■

Lemma 2.3 *Let G be a graph such that every edge has one of its extremities incident to exactly one other edge. If there is a progressive k-consistent crusade in G, then $mis(G) \leq k$.*

Proof. Let $C = (X_0, X_1, \ldots, X_r)$ be a progressive k-consistent crusade in G, with skeleton $\mathcal{S}$ labeled as in Definition 2.2. Let $e_i = X_i \setminus X_{i-1} = \{x_i, y_i\}$. We construct a monotone internal search strategy that successively clears the edges $e_1, e_2, \ldots, e_r$. Note that every node of $\mathcal{S}$ has out-degree 1 because C is progressive.

In $\mathcal{S}$, X_1 consists of a unique node u representing $\{e_1\}$. We use k_u searchers to clear e_1. Since, $k_u \geq |\delta(\{e_1\})|$, this number of searchers is sufficient.

Assume now that we have cleared all edges $e_1, \ldots, e_{i-1}$ with a monotone internal strategy. Assume moreover that the number of searchers in each connected component of X_{i-1} is equal to the label of the node of $\mathcal{S}$ corresponding to that component.

Let u be a component of level i of $\mathcal{S}$. We consider three cases depending on the in-degree $\deg^-(u)$ of node u.

Case 1: $\deg^-(u) = 0$. Then u consists of a unique edge e_i, which is cleared with k_u searchers.

Case 2: $\deg^-(u) = 2$. Then e_i connects two connected components Y_{i-1} and Z_{i-1} of clear edges. One of the two extremities of e_i, say x_i, is of degree 2 by definition of G. One searcher is staying at x_i to avoid recontamination. The edge e_i is cleared by moving this searcher from x_i to y_i. Hence $k_{Y_{i-1}} + k_{Z_{i-1}}$ searchers are sufficient.

Case 3: $\deg^-(u) = 1$. Then e_i is incident to a unique connected component Y_{i-1} of clear edges. Assume, w.l.o.g., that x_i is the end-point of e_i with degree 2. If $x_i \in \delta(Y_{i-1})$, then e_i is cleared by moving one searcher from x_i to y_i. Thus assume now that $x_i \notin \delta(Y_{i-1})$. This implies $y_i \in \delta(Y_{i-1})$. If $y_i \notin \delta(Y_{i-1} \cup \{e_i\})$, then one searcher can clear e_i by moving from y_i to x_i. If $y_i \in \delta(Y_{i-1} \cup \{e_i\})$, then $|\delta(Y_{i-1} \cup \{e_i\})| = |\delta(Y_{i-1})| + 1$. Since $k_u \geq |\delta(Y_{i-1} \cup \{e_i\})|$, there is a free searcher in Y_{i-1} that can move to y_i, and clear e_i by moving from y_i to x_i.

The search strategy obtained by clearing all edges as explained above is internal and monotone. ■

Proof of Theorem 2.1. Let G be any graph with $cs(G) \leq k$. The 1-expansion of G is the graph H obtained from G by replacing every edge e by two consecutive edges e' and e''. We have $cs(H) \leq cs(G)$, by transforming any move along $e \in E(G)$ of a search strategy for G into two moves in H along e' and e''. Therefore, thanks to Lemma 2.1, there exists a connected crusade of frontier $\leq k$ in H. As we noticed before, a connected crusade is k-consistent. Therefore, applying Lemma 2.2, we get that there exists a progressive k-consistent crusade in H. Since H is the 1-expansion of G, each of its edges has one of its extremities incident to exactly one other edge. Therefore, by Lemma 2.3, $mis(H) \leq k$. We complete the proof by observing that $mis(G) \leq mis(H)$. Indeed, a monotone internal strategy S_G for G can be obtained from a monotone internal strategy S_H for H as follows. Let e' and e'' be the two incident edges of H resulting from the expansion of an edge e of G. Assume $e' = \{x, y\}$ and $e'' = \{y, z\}$. If one searcher moves from x to y, or from z to y in S_H then this searcher remains in place in S_G. If one searcher moves from y to x (resp., from y to z) in S_H then this searcher moves from z to x (resp., from x to z) in S_G. ■

3 Connected and Monotone Internal Searches are not Minor Closed

A standard way to tackle graph searching problems is to study the nature of their obstruction sets. Unfortunately, we have the following.

Observation 3.1 *The class of graphs that can be cleared by a connected search strategy using at most k searchers is not minor closed. There is a pair of graphs (G, H) with $H \prec G$ and $cs(H)/cs(G) \simeq 3/2$.*

Observation 3.2 *The class of graphs that can be cleared by a monotone internal search strategy using at most k searchers is not minor closed. There is a pair of graphs (G, H) with $H \prec G$ and $mis(H)/mis(G) \simeq 3/2$.*

Observations 3.1 and 3.2 can be justified as follows. Figure 4 displays a subgraph G_0 of the $p \times q$ mesh, with $q = 2k$, and $p = 6l$, $l \gg k$. Clearly $s(G_0) = 2k + 1$ by clearing G_0 "from left to right": there is one searcher per row, and the extra searcher clears all vertical edges. Similarly, one can check that $mis(G_0) = cs(G) = 2k + 1$. Edges e and f play an important role, as they allow

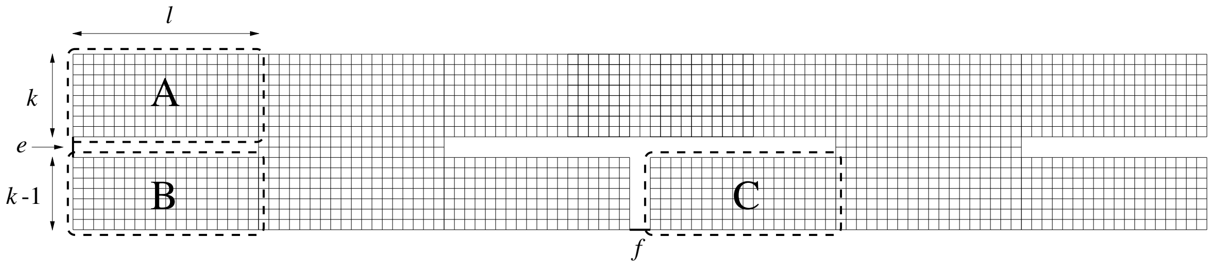


Figure 4: The graph G_0.

the monotone "left to right" search strategy to be internal, and connected. If e is removed, then $s(G_0 - e)$ and $mis(G_0 - e)$ remain unchanged. The beginning of the strategy, i.e., while clearing the l leftmost columns, is however not connected. There are $k+1$ searchers clearing the upper k rows (zone A), and k searchers clearing the lower $k-1$ rows (zone B). For connected search, we have $cs(G_0 - e) = 3k$. Intuitively, up to symmetry, since $l \gg k$, zone B must be cleared from right to left, while $2k$ searchers protect the cleared part (including zone A) from recontamination. Hence Observation 3.1 holds. If f is also removed, then $s(G_0 - e - f) = s(G_0) = 2k+1$, and $cs(G_0 - e - f) = cs(G_0 - e) = 3k$, but $mis(G_0 - e - f) = 3k > mis(G_0 - e)$. Intuitively, up to symmetry, since $l \gg k$, zone C must be cleared with k new searchers. Indeed, k of the $2k$ searchers currently in the graph cannot jump to the leftmost column of zone C, and moving there requires protection of $k + l \gg 3k$ nodes. Hence Observation 3.2 holds.

Notice that the graph $G_0 - e$ of Figure 4 allows us to assert the following.

Observation 3.3 *There is a graph G with $cs(G)/mis(G) \simeq 3/2$.*

The next section tackles the case of trees, in which monotone internal and connected searches are both minor closed.

4 The Case of Trees

In this section, we show that there is a unique obstruction for the class of trees T such that $cs(T) \leq k$ (since, for any tree T, $mcs(T) = cs(T)$ [1], we consider only the monotone case). Our proof is based on the notion of *k-caterpillar* and *spine*. A spine is a path. A 0-caterpillar is also a path, and it is its own spine. For $k > 0$, a tree T is a k-caterpillar with spine P if, for every connected component T' of $T \setminus P$, the two following properties hold: (1) there is a path P' such that T' is a $(k-1)$-caterpillar with spine P', and (2) one of the two extremities of P' is adjacent to P. A 1-caterpillar is hence a subdivision of a caterpillar in the usual sense, i.e., a path $x_1, \ldots, x_k$ with $k_i \geq 0$ paths pending from every x_i.

Notice that any tree is a k-caterpillar for k large enough. The notion of k-caterpillar is related to the notion of *caterpillar dimension* introduced in [24] (see also [23]). One can easily show that the connected search number of a tree, starting from node v, is at most the caterpillar dimension of the tree rooted at v plus 1. However, as far as we know, there is no characterization of connected search number in terms of caterpillar dimension, whereas we establish an equivalence between connected search numbers and k-caterpillars. Indeed, we show that k-caterpillars form the class of trees that can be connectedly cleared with at most $k+1$ searchers.

Given a tree T and two vertices v, w of T, we denotes by T_v the tree T rooted at v, and by $T_v[w]$ the subtree of T_v rooted at w. Recall that the depth of a rooted tree T is the maximum distance from its root to the leaves. We denote by B_k the complete binary tree of depth k, and by D_k the tree obtained by connecting the three roots of three copies of B_{k-1} to a unique new vertex. Finally, we denote by $T_1 \preceq T_2$ the relation "T_1 is a minor of T_2".

We now prove a sequence of preliminary lemmas.

Lemma 4.1 *Any tree T such that $D_k \not\preceq T$ is a $(k-1)$-caterpillar.*

Proof. We start by a preliminary statement. Let T_1 and T_2 be two trees, rooted at x_1 and x_2 respectively. We denote by $T_1 \preceq_{x_2} T_2$ the relation "T_1 is a x_2-rooted minor of T_2", that is node x_1 is either x_2 or the result of contracting a series of edges, some of them containing x_2 as end-point. Now, let T be a tree and v be a vertex of T such that $B_k \not\preceq_v T$, $k \geq 1$. We claim that T is a $(k-1)$-caterpillar and v is an extremity of its spine. The proof of that claim is by induction on k. If $B_1 \not\preceq_v T$ then clearly T is a path with extremity v. If $k > 1$ and there is a vertex v such that $B_k \not\preceq_v T$, then there are two cases. If $B_{k-1} \not\preceq_v T$, then by induction hypothesis, T is a $(k-2)$-caterpillar with v as the first vertex of the spine. If $B_{k-1} \preceq_v T$, then let S be the set of vertices w such that $B_{k-1} \preceq_w T_v[w]$. S is a path starting at v, and all the connected components of $T - S$ are $(k-2)$-caterpillars, in which the corresponding spine starts at the vertex adjacent to one of the vertices of S in T. Indeed, if $z \notin S$ and z is adjacent to $w \in S$, then $T_v[z]$ is one of the connected components of $T - S$ and $B_{k-1} \not\preceq T_v[z]$.

To complete the proof of the lemma, it hence just remains to show that there is a vertex v such that $B_k \not\preceq_v T$. By contradiction, assume that $D_k \not\preceq T$ and for every v vertex of T, $B_k \preceq_v T$. There is a vertex z with two neighbors, z_1 and z_2, such that $B_{k-1} \preceq_{z_1} T_z[z_1]$ and $B_{k-1} \preceq_{z_2} T_z[z_2]$. This implies that, either $B_k \preceq_{z_i} T_z[z_i]$ or $B_k \preceq_z T_{z_i}[z]$. In both cases, we get $D_k \preceq T$, a contradiction. ■

Lemma 4.2 *For any $k \geq 1$, $cs(D_k) \geq k+1$.*

Proof. We prove that, for any connected search strategy in D_k, there is a step in which at least $k+1$ searchers are required to avoid recontamination. Let T_1, T_2, and T_3 be the three sub-trees attached to the root of D_k and isomorphic to B_{k-1}. Consider the first step i_1 during which the root of D_k is reached by a searcher. Assume, w.l.o.g., that T_1 is still completely contaminated at step i_1. Let $i_2 > i_1$ be the first step during which a leaf of T_1 is reached by a searcher. The path P from the root r to this leaf, say f, has length k. Moreover, at step i_2, P is cleared but, for every vertex $x \neq f$ of P, there is a path from x to a contaminated leaf, and thus at least one searcher is needed for every x to avoid recontamination. Moreover, there is one additional searcher used to clear f. Hence, at least $k+1$ searchers are required at step i_2. ■

Lemma 4.3 *If T is a k-caterpillar then $cs(T) \leq k+1$.*

Proof. We show that, if T is a k-caterpillar with spine P, then there is a connected search strategy using $k+1$ searchers starting at one extremity of P. The proof is by induction. For $k = 0$, a 0-caterpillar is a path and hence the result holds trivially. Assume now that every $(k-1)$-caterpillar with spine $P' = \{w_0, \ldots, w_\ell\}$ can be cleared with k searchers, starting at w_0. Let T be a k-caterpillar with spine $P = \{v_0, \ldots, v_m\}$. Let us denote by $w_{i,0} \ldots w_{i,d_i}$ the set of neighbors of v_i not in P. Then, $T_{w_{i,j}}[v_i]$ is a $(k-1)$-caterpillar with path $P_{i,j}$ starting at $w_{i,j}$. The search strategy for T is the following. Start at v_0 with $k+1$ searchers. Every time you reach a new vertex v_i of P, let one searcher at v_i and, for $j = 0, \ldots, d_i$, clear every tree $T_{w_{i,j}}[v_i]$ with the k remaining searchers, using the strategy that starts at w_j (there is one, by induction hypothesis). Then, follow the path to the next contaminated vertex v_{i+1}, with the $k+1$ searchers. ■

Now we are ready to prove the following Theorem.

Theorem 4.1 *For any tree T, the following three properties are equivalent:*

(1) *T is not a $(k-1)$-caterpillar;*

(2) *$D_k \preceq T$;*

(3) *$cs(T) \geq k+1$.*

Proof. The theorem is a direct consequence of the previous lemmas: Lemma 4.1 proves (1)⇒(2), Lemma 4.2 proves (2)⇒(3), and Lemma 4.3 proves(3) ⇒(1) . ■

Rephrasing Theorem 4.1, we get:

Corollary 4.1 *For a tree T, $cs(T) \leq k$ if and only if T is a $(k-1)$-caterpillar. Moreover, the set of obstructions of the class of trees T with $cs(T) \leq k$ contains D_k as unique element.*

Corollary 4.2 *For any tree T, if $s(T) \geq 2$, then $s(T) \leq cs(T) \leq 2s(T) - 2$. Moreover, for $k \geq 1$, $cs(D_{2k-1}) = 2s(D_{2k-1}) - 2$.*

Proof. Let T be a tree, and assume that $s(T) = j$. Let M_k be any tree obtained from a complete ternary tree of depth k after removing one leaf from every set of three sibling leaves (i.e., nodes at distance k from the root). Parsons [28] has proved that M_k is an obstruction of the class of graphs G with search number $\leq k$. Therefore $M_j \not\preceq T$.

Now, M_k is a subgraph of the graph obtained from D_{2k-2} by contracting every edge connecting a vertex of level $2j-1$ to a vertex of level $2j$, for $0 < j < k-1$. Therefore, for any $k \geq 1$, $M_k \preceq D_{2k-2}$. Thus $D_{2j-2} \not\preceq T$, which implies, by Theorem 4.1, that $cs(T) \leq 2j - 2 = 2s(T) - 2$.

To prove that the bound is tight, let us consider D_{2k-1}. We have $s(D_{2k-1}) \leq cs(D_{2k-1}) = 2k$ and $M_k \preceq D_{2k-1}$, which implies that $s(D_{2k-1}) \geq k+1$. On the other hand, we give a search strategy for D_{2k-1} that uses $k+1$ searchers. The strategy starts by placing a searcher in the root r. Next, it proceeds to clear the edges of the three branches which are isomorphic to B_{2k-2}. It is easy to see that this can be done with k searchers, and the edges connecting r to the three branches need no additional searcher. Therefore $cs(D_{2k-1}) = 2s(D_{2k-1}) - 2$, which completes the proof. ■

Theorem 4.2 *For any tree T, $mis(T) = cs(T)$.*

To prove this theorem, we use the following result proved in [1]. Let $cs_x(T)$ be the minimum number of searchers required to clear T by a connected strategy where all searchers are initially placed at node x, and the first move consists of clearing one edge incident to x. We have:

Lemma 4.4 (Barrière *et al.* [1]) *Let r be the root of B_k, $k \geq 1$, then $cs(B_k) = k$ and $cs_r(B_k) = k+1$.*

Proof of Theorem 4.2. Since mis and cs are both minor closed for trees, we only need to prove that $mis(D_k) = k+1$ for all $k \geq 1$. For that purpose, we first show the following.

Claim. For every k, $mis(B_k) = k$. Moreover, in any search strategy for B_k using k searchers, there is a step in which (1) k searchers are involved, (2) none of these searchers occupies the root, and (3) all edges incident to the root of B_k are clear.

The proof is by induction. The result is true for $k = 1$. Assume that it holds for $k-1$, and let us consider B_k. Assume, for the purpose of contradiction that there exists a search strategy S for B_k using $k-1$ searchers. (Note that $mis(B_k) \geq k-1$ since, otherwise, $mis(B_{k-1}) < k-1$.) B_k is composed of two copies of B_{k-1} denoted by A and A', rooted at a and a', respectively. Assume, w.l.o.g., that A is the first copy to be completely clear in S. Let S' be the search strategy for A consisting of all steps of S in which only nodes or edges of A are concerned. S' clears A with $k-1$ searchers. By induction hypothesis, there is a step s_0 of S' in which (1) $k-1$ searchers are involved, (2) none of these searchers occupies node a, and (3) all edges of A incident to a are clear. Since at least one edge of A' is still contaminated at step s_0, all edges incident to the root of A are recontaminated, in contradiction with the monotonicity of S. Therefore $mis(B_k) \geq k$. On the other hand, $cs(B_k) \leq k$ by Lemma 4.4. Thus $mis(B_k) \leq k$ by Theorem 2.1.

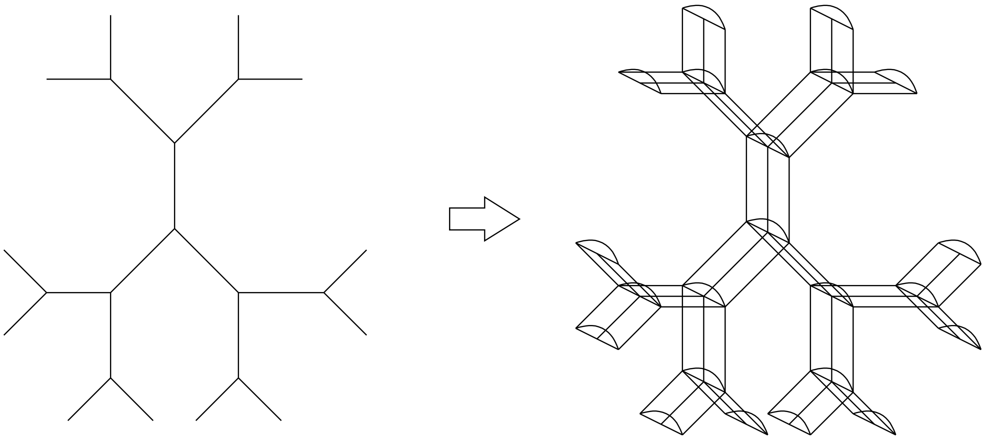
To prove properties (1), (2), and (3), let S be any monotone internal strategy clearing B_k with k searchers. Again, assume, w.l.o.g., that A is the first copy to be completely clear in S, and let S' be the search strategy for A consisting of all steps of S in which only nodes or edges of A are concerned.

– If there is a step s_0 at which k searchers are in A, then all edges of A' are still contaminated at this step. Since S is internal, it clears A' by moving from A to A', and "starting" from a'. By Lemma 4.4, the k searchers are needed for the clearing of A' by S, and thus the three properties (1), (2), and (3), are satisfied.

– If only $k-1$ searchers are used by S', then, by induction hypothesis, there is a step s_0 of S' in which (1) $k-1$ searchers are involved, (2) none of these searchers occupies a, and (3) all edges of A incident to a are clear. Let v be the node occupied by the remaining searcher at step s_0. The path from v to a is clear to protect edges incident to a from recontamination. If $v = r$, then, as in the previous case, S clears A' "starting" from a' with the k searchers, and thus the three properties (1), (2), and (3), are satisfied. If $v \in A'$, then properties (1), (2), and (3), are satisfied at step s_0. This completes the proof of the Claim.

Since $B_k \prec D_k$, $mis(D_k) \geq k$ from the Claim. Assume, for the purpose of contradiction, that there exists a monotone internal strategy S clearing D_k with k searchers. D_k is obtained from three copies A, A' and A'' of B_{k-1}. Assume, w.l.o.g., that A is the first copy to be completely clear, and that A' is the next copy to be completely clear. Let S' be the search strategy for A consisting of all steps of S in which only nodes or edges of A are concerned. If there is a step s_0 at which k searchers are in A, then all edges of A' and A'' are still contaminated at this step. If S' uses only $k-1$ searchers to clear A, then, from the Claim, there is step s_0 satisfying properties (1), (2), and (3), and thus the kth searcher is necessarily at the root of D_k, and all edges of A' and A'' are still contaminated at this step. Therefore, in both cases, since S is internal, it clears A' by "starting" from a'. By Lemma 4.4, k searchers are needed for the clearing of A' by S. This implies recontamination of A from yet contaminated edges of A'', a contradiction. ■

Remark. Distinct values for s and cs can be achieved for graphs of arbitrary connectivity. Figure 5(a) displays the tree D_3. We have $cs(D_3) = 2s(D_3) - 2 = 4$. Figure 5(b) displays the graph



$T = D_3$ $\qquad$ G

Figure 5: Search numbers of graphs with high connectivity.

$D_3 \times K_3$. For a fixed r, the ratio $cs(D_k \times K_r)/s(D_k \times K_r)$ approaches 2 when k goes to infinity.

5 Concluding Remarks and Open Problems

The main open problem is whether recontamination helps for connected search. That is, whether, for any graph G with $cs(G) \leq k$, there exists a monotone connected search strategy using $\leq k$ searchers. All the standard techniques for proving monotonicity fail for connected search mainly because the kernel argument of all the proofs of monotonicity is based on the fact that, in any search variant, the cost of the search can be expressed by a *connectivity function*, that is a nonnegative valued function on a set $S \subseteq \mathcal{P}(M)$ that is invariant over complement and satisfies the submodular property (see [15] for a more detailed discussion). These techniques fail for connected search as the intersection of two connected sets is not necessarily connected.

Another important open problem is whether the ratio $cs(G)/s(G)$ can be bounded. We proved that $cs(T)/s(T) < 2$ for trees (cf. Corollary 4.2). A similar bound for general graphs, say $cs(G)/s(G) \leq b$, would imply that: $H \prec G \Rightarrow cs(H) \leq b \cdot cs(G)$. That way, we could derive approximation algorithms for cs from algorithms for the usual search number s.

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 '02), Winnipeg, August 10-13, 2002.

[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] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21:358–402, 1996.

[6] H. L. Bodlaender and D.M. Thilikos. Computing small search numbers in linear time. Technical Report UU-CS-1998-05, Utrecht University, 1998.

[7] H. L. Bodlaender, and D. M. Thilikos. Graphs with branchwidth at most three. *Journal of Algorithms*, 32:167–194, 1999.

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

[9] 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.

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

[11] J. Díaz, J. Petit, and M. Serna. A survey on graph layout problems. *ACM Computing Surveys*, to appear.

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

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

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

[15] F. Fomin and D.M. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, to appear.

[16] M. Franklin, Z. Galil, and M. Yung. Eavesdropping games: a graph theoretic approach to privacy in distributed systems. *Journal of the ACM*, 47(2):225–243, 2000.

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

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

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

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

[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] N. Linial, A. Magen and M. Saks. Trees and Euclidian metrics. In 30st *ACM Symp. on Theory of Computing* (STOC '98), pages 169–175, 1998.

[24] J. Matousek. On embedding trees into uniformly convex Banach spaces. *Israelian Journal of Mathematics*, 114:221–237, 1999.

[25] 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.

[26] 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.

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

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

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

[30] N. Robertson and P. Seymour. Graph minors XIII. The disjoint path problem. *Journal of Combinatorial Theory, Ser. B*, 63:65–110, 1995.

[31] N. Robertson and P. Seymour. Graph minors — A survey. *Surveys in Combinatorics*, Cambridge University Press, I. Anderson (ed.), pages 153–171, 1985.

[32] P. Seymour and R. Thomas. Graph searching, and a min-max theorem for treewidth. *Jour. Combin. Theory, Ser. B*, 58:239–257, 1993.

[33] 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. (To appear in *SIAM J. Computing*.)

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

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

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

[37] A. Takahashi, S. Ueno, and Y. Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. *Discrete Mathematics*, 127(1-3):293–304, 1994.

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

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