Connected Graph Searching

Lali Barrière, Paola Flocchini, Fedor V. Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, Dimitrios M. Thilikos

Connected Graph Searching[1] [2]

Lali Barrière[a], Paola Flocchini[b], Fedor V. Fomin[c], Pierre Fraigniaud[d], Nicolas Nisse[e], Nicola Santoro[f], Dimitrios M. Thilikos[g]

[a] *Departament de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya, Spain.*

[b] *School of Information Technology and Engineering, University of Ottawa, Canada.*

[c] *Department of Informatics, University of Bergen, Norway.*

[d] *CNRS and Université Paris Diderot, France.*

[e] *MASCOTTE, INRIA, I3S (CNRS/Université Nice Sophia Antipolis), France.*

[f] *School of Computer Science, Carleton University, Canada.*

[g] *Department of Mathematics, National and Kapodistrian University of Athens, Greece.*

Abstract

In the graph searching game the opponents are a set of searchers and a fugitive in a graph. The searchers try to capture the fugitive by applying some sequence of moves that include placement, removal, or sliding of a searcher along an edge. The fugitive tries to avoid capture by moving along unguarded paths. The search number of a graph is the minimum number of searchers required to guarantee the capture of the fugitive. In this paper, we initiate the study of this game under the natural restriction of connectivity where we demand that in each step of the search the locations of the graph that are clean (i.e. non-accessible to the fugitive) remain connected. We give evidence that many of the standard mathematical tools used so far in classic graph searching fail under the connectivity requirement. We also settle the question on "the price of connectivity", that is, how many searchers more are required for searching a graph when the connectivity demand is imposed. We make estimations of the price of connectivity on general graphs and we provide tight bounds for the case of trees. In particular, for an n-vertex graph the ratio between the connected

[1] Results of this paper have appeared in the proceedings of the 14th ACM Symposium on Parallel Algorithm and Architecture (SPAA 2002), the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG 2003), the 13th SIAM Conference on Discrete Mathematics, (DM 2006), and the 7th Latin American Theoretical Informatics Symposium (LATIN 2006).

[2] Paola Flocchini and Nicola Santoro are supported in part by NSERC Discovery Grant; Pierre Fraigniaud is supported by the ANR projects ALADDIN and PROSE, and by the INRIA project GANG; Nicolas Nisse is supported by the ANR project AGAPE; Dimitrios M. Thilikos is supported by the project "Kapodistrias" (ΑΠ 02839/28.07.2008) of the National and Kapodistrian University of Athens (project code: 70/4/8757).

searching number and the non-connected one is $O(\log n)$ while for trees this ratio is always at most 2. We also conjecture that this constant-ratio upper bound for trees holds also for all graphs. Our combinatorial results imply a complete characterization of connected graph searching on trees. It is based on a forbidden-graph characterization of the connected search number. We prove that the connected search game is monotone for trees, i.e. restricting search strategies to only those where the clean territories increase monotonically does not require more searchers. A consequence of our results is that the connected search number can be computed in polynomial time on trees, moreover, we show how to make this algorithm distributed. Finally, we reveal connections of this parameter to other invariants on trees such as the Horton-Stralher number.
Keywords: Graph searching, cops-and-robbers, network security.

---

1. Introduction

The classical isoperimetric problem (often attributed to Dido, the legendary founder and first queen of Carthage) can be stated as follows: Among all closed curves in the plane of given length, which curve encloses the maximum region? While the circle appears to be an intuitive solution to the problem, it took several thousands of years to develop tools for proving this apparently evident fact. There is a conceptual link between the isoperimteric problem and the principle of least action in physics. The most familiar illustration of this link is the shape of a water drop: given a fixed amount of water, the surface tension forces the drop to take a shape which minimizes the surface area of the drop. The study of discrete versions of the isoperimetric problem in graphs has brought to the notion of expander graphs, a notion that is now extensively used in different areas of mathematics and computer science [1].

A natural extension of discrete isoperimetric problem is the following dynamic version, where the task is to expand a subset of elements S into another set T via a sequence of steps, under the constraint that, at every step, the surface area – or simply its border size – be as small as possible. Let us give two illustrative examples of frameworks in which this constrained expansion process finds application. The first example is from visualization, specifically from the domain of compression techniques design used for streaming triangular meshes over communication channels with limited bandwidth. 3D meshes are used for a variety of applications in entertainment, e-commerce, CAD, and medicine. Quite often, those meshes are acquired using modern 3D scanning technologies, and they easily reach sizes of gigabytes. A common approach for processing large meshes (those that are too large to fit into main memory) is to perform a "conquest" of the mesh by starting from an arbitrary

triangle, and by successively extending the frontier by conquering a new vertex adjacent to one of the current conquered triangles. A lower bound on the memory requirements for processing a streaming mesh is then provided by the number of "border" vertices bounding the conquered part of the mesh [2, 3].

The second example is from agent-based software-system testing [4]. Given a set of inter-related functional units in a distributed system, the objective is to check the correctness of these units, one by one. In order to avoid checked units to be subject to propagation of faults from neighboring unchecked units, the "checker" uses resources to protect checked units against unchecked neighboring ones. Once all neighboring units of a checked unit U have been checked, there is no need to protect U anymore, and only the *frontier* between checked and unchecked units has to be guarded. The objective is to minimize the amount of resources required for the system to be entirely checked.

Both illustrative problems can be modelled as *graph searching*. In a graph searching game, alternatively known as a *pursuit-evasion game*, the one part is a set of "escaping" mobile entities, called *fugitives* (or *evaders*), that hide in a graph representing a network, and the other part is a number of "chasing" entities, called *searchers* (or *pursuers*), that move systematically in the graph and aim at capturing the evaders [5]. The game may vary significantly according to the capabilities of the fugitives and the searchers in terms of relative speed, sensor capabilities, visibility, mobility restrictions, etc. [6]. These variants are mainly application driven. However, their study has inspired, and was linked to foundational issues in computer science. Some of the former and current applications of graph searching are VLSI design [7], computational complexity [8], network security [9], and databases [10].

The first mathematical models for the analysis of graph searching games where introduced in the 70's by Parsons [11, 12] and Petrov [13], while the first variants, along with the corresponding algorithmic and complexity results, appeared during the 80's [14, 8, 5]. Graph searching revealed the need to express several intuitive informal concepts in a formal mathematical way. These concepts include sense of direction/orientation, avoidance, surrounding, hiding, persecution, threatening, etc. This led to the design of various advanced combinatorial tools. Some of most powerful tools for tackling graph searching problems emerged from the Graph Minors theory, developed by Robertson and Seymour towards proving the long-standing Wagner's Conjecture [15]. This theory has offered deep graph-theoretic results and techniques with direct consequences to problems at the core of graph searching games.

In many applications of graph searching, especially those aiming at clearing a network, a crucial issue is to guarantee secure communication channels between the searchers so that

they can safely exchange information. In graph-theoretic terms, the clean part of the network is required to remain always *connected*. Unfortunately, the aforementioned combinatorial and algorithmic tools used for tackling the graph searching problem, i.e., those tools coming from Graph Minors theory, generally fail under such a global connectivity restriction. Hereafter, we provide two open problems whose solutions currently escape the reach of Graph Minors theory. Let us define a search strategy as an ordered sequence $\mathcal{S}$ of *actions* where each action is either *placing* a searcher at a vertex, or *removing* a searcher from a vertex, or *sliding* a searcher along an edge. The invisible fugitive is supposed to move arbitrarily fast from node to node along the edges of the graph, and it can traverse any node that is not occupied by a searcher.

NP-*membership.* A straightforward reduction from classical graph searching shows that the problem of checking whether a graph G can be connectedly cleared using at most k searchers is NP-hard. However, NP-membership remains an insisting open problem. The standard approach for proving NP-membership for searching problems is to prove that the corresponding game is *monotone*, i.e. strategies allowing the fugitive to revisit (or "recontaminate") an already cleaned part of the graph cannot do better than strategies for which recontamination is not allowed. See, e.g., [16, 17]. If a game is monotone then NP-membership follows directly from the fact that the existence of a monotone search strategy yields the existence of a search strategy with a bound on its length, and such a strategy can be used as a succinct certificate. A basic technique for proving monotonicity (emerging from the Graph Minors theory) is to represent each search strategy as an *expansion* of clean edges, i.e., as a sequence $\mathcal{E}$ of edge sets $X_0, X_1, \ldots, X_t$ (representing the clean territories at search step i) where $X_0 = \emptyset$, $X_t = E$, and $|X_{i+1} \setminus X_i| \leq 1$ for every $i = 0, 1, \ldots, t-1$. The objective is to find an expansion that minimizes the maximum frontier size of the X_i's, where the frontier of a set $X \subseteq E$, is defined as the set of vertices $\partial(X)$ that are both incident to an edge in X and to an edge in $E \setminus X$. Let us denote by $\delta(X)$ the number of vertices in $\partial(X)$. It has been observed in [16, 18] that monotonicity essentially follows from the fact that δ is a submodular function, that is, for every $X, Y \subseteq E$,

$$\delta(X \cap Y) + \delta(X \cup Y) \leq \delta(X) + \delta(Y).$$

However, the submodularity does not have a counterpart for connected search games because $X \cap Y$ is not necessarily connected. As first observed in [19], connected search is *not* monotone, which implies than the conjectured NP-membership of the connected variant requires techniques beyond monotonicity to be proved.

Polynomial time algorithms for fixed parameter. Monotonicity also establishes a close relationship between graph searching and various notions of *width*, including *pathwidth* for graph searching, and *treewidth* for "visible" graph searching. In particular, it enables the design of efficient exponential-time exact or polynomial-time approximation algorithms for computing the minimum number of searchers required to clear a graph (see, e.g., [20, 21]). For almost all width parameters related to graphs searching, the problem is polynomial-time solvable for fixed parameter [22]. In contrast, the absence of monotonicity in the case of connected graph searching precludes using these powerful tools in this context. More specifically, let us denote by $\mathbf{s}(G)$ the minimum number of searchers required to search (i.e., to clean) the graph G, and by $\mathbf{cs}(G)$ the analogous graph searching parameter in the case where only connected strategies are permitted. Let us first explain why the problem of checking whether a given graph G satisfies $\mathbf{s}(G) \leq k$ can be solved in polynomial time for every fixed non-negative integer k. The main observation is that the class

$$\mathcal{G}_k = \{G \,|\, \mathbf{s}(G) \leq k\}$$

is *minor-closed*. That is, if H is a *minor* of $G \in \mathcal{G}_k$ (i.e., if H is a subgraph of a graph obtained from G after contracting some edges) then $\mathbf{s}(H) \leq \mathbf{s}(G)$. Combining this observation with known results from Graph Minors theory, it follows that, for each fixed k, there is a *finite* set of minor-minimal graphs (called *obstructions*) not in $\mathcal{G}_k$. Therefore, $G \in \mathcal{G}_k$ if and only if G does not contain any of these graphs as a minor. Since checking whether a graph M is a minor of a graph G can be done in polynomial time for a fixed size graph M [23, 24], we conclude that for each fixed k, $\mathcal{G}_k$ can be recognized in polynomial time. Actually, one can even conclude that the problem is solvable in time $f(k) \cdot n$ where f is a function not depending on the size of the graph. Unfortunately, no such good news exist for the connected counterpart of the search problem because the graph class

$$\mathcal{C}_k = \{G \,|\, \mathbf{cs}(G) \leq k\}$$

is *not* minor-closed. Actually, it is possible to show that parameter $\mathbf{cs}$ can increase twice by removing just a single edge. Therefore it seems that the design of an algorithm that checks $\mathbf{cs}(G) \leq k$ in polynomial time for fixed k requires tools beyond those provided by the yet powerful Graph Minors theory.

1.1. Our results

The results in this paper are directly motivated by the above open questions, and they constitute a first attempt to studying the impact of the connectivity requirement on graph

searching problems. This impact is measured by the ratio

$$\frac{\mathbf{cs}(G)}{\mathbf{s}(G)}$$

between the number $\mathbf{cs}(G)$ of searchers required to clear a graph G under the connectivity constraint, and the number $\mathbf{s}(G)$ of searchers required to clear the same graph in absence of such a constraint. In other words, this paper tackles the *price of connectivity* in graph searching. In this paper we make advances in understanding the price of connectivity by proving the following two results:

1. For each connected n-vertex graph G, $\frac{\mathbf{cs}(G)}{\mathbf{s}(G)} = O(\log n)$;
2. For each tree T, $\frac{\mathbf{cs}(T)}{\mathbf{s}(T)} < 2$.

To derive the first of these two results, we use the concept of a *branch-decomposition* introduced by Robertson and Seymour in [25]. Branch-decompositions provide ways to decompose a graph into smaller parts that are arranged in a tree-like fashion. Our result is based on the fact that these parts can be made connected while maintaining the "separation cost" of this decomposition (cf. Lemma 1). Then, by suitably defining an "orientation" of this connected branch-decomposition, we are able to transform every search strategy into a connected one, with just a logarithmic overhead on its cost in term of number of involved searchers. These results, combined with existing recently derived algorithmic results on approximating the size of minimal separators, and on approximating the value of the treewidth, imply that $\mathbf{cs}(G)$ can be approximated in polynomial time up to a multiplicative factor $O(\log^{3/2} n)$.

Our second result is derived from a complete characterization of the connected search number of trees. We first prove that, as opposed to the case of arbitrary graphs, connected search is monotone on trees (cf. Lemma 3). Next, we identify, for each k, the obstruction set for the class $\mathcal{T}_k = \{T \mid \mathbf{cs}(T) \leq k\}$. That is, we identify the set of contraction-minimal trees that are not in $\mathcal{T}_k$. We stress that this identification provides one of the very few examples of contraction-closed graph classes for which the obstruction set can be entirely characterized. In fact, somewhat surprisingly, the obstruction set for $\mathcal{T}_k$ is reduced to one element, which is in contrast to the fact that the size of the obstruction set of all classic (non-connected) graph searching parameters is growing (at least) exponentialy as a function of k. These combinatorial results are used for the design of a linear-time algorithm computing $\mathbf{cs}(T)$ for trees. This algorithm makes use of the rooted variant, $\mathbf{cs}_v(T)$, of the connected search number, in which we prescribe the vertex v of T from which the connected search strategy should expand. An important byproduct of the design of our algorithm for

trees is that $\mathbf{cs}_v(T)$ is equal to – and thus may serve as an alternative definition for – the Horton-Strahler number [26, 27, 28]. This latter ubiquitous number serves as a measure of the "branching complexity" of a rooted tree, and is known to find applications in many different areas of science for the statistical analysis of hierarchical systems (including hydrology [29], programming languages [30], mathematical biology [31, 32] and, recently, social networks [33]).

All the results summarized above, in conjunction with additional considerations to be detailed further in the text, allow us to conjecture the following:

$$\lim_{n\to\infty} \sup_{|V(G)|=n} \frac{\mathbf{cs}(G)}{\mathbf{s}(G)} = 2. \tag{1}$$

In other words, we conjecture that what we have proved for trees actually holds for all graphs. The validity of this conjecture would imply –and there are indications for this (cf. Section 3)– that the tree structure is critical towards evaluating the price of connectivity. Conceptually, graph searching parameters ask for a "sense of direction" in a graph, or, stated differently, for an arrangement of the vertices along a virtual axis along which the optimal search strategy should be deployed. Informally, the connectivity requirement places another constraint on the direction to be followed by the searchers during the deployment of the search strategy. As indicated in the proof of Theorem 2, the existence of these two possibly conflicting directions is the key motivation for our conjecture. In some sense, there are empirical evidences that the worst-case conflict between these two orientations occurs in trees.

1.2. Related work

Graph searching is a well studied model in Mathematics and Theoretical Computer Science. The first mathematical formulation of graph searching is due to Parsons [11, 12]. The formulation was inspired by an earlier article of Breisch in Southwestern Cavers Journal [34] proposing a "speleotopological" approach for the problem of finding an explorer lost in a system of dark caves. Megiddo et al. [5], proved that the decision version of the problem is NP-complete in general and solvable in linear time on trees. Let us remark that the proof that the problem is in NP is based on the highly non-trivial fact that there are optimal monotone strategies [16, 17]. For more references on graph searching, we refer to [6].

Since the appearance of the conference versions of this paper in [35, 36], the connected graph searching has been studied both from algorithmic and combinatorial points of view. Yang et al. proved that for the connected variant of searching, recontamination can be useful [19]. The "cost of connectivity": the ratio between the number of searchers required in the

connected case and the number of searchers required without the connectivity requirement, was studied in [37, 38, 39].

A related problem on planar triangulations was studied in [40]. Connected searching with visible fugitive is discussed in [41]. Finally, distributed connected search strategies have been designed in [42, 43, 44, 45, 46].

2. Model and definitions

Graph searching refers to a problem that has been thoroughly and extensively investigated in the literature, and that describes a variety of application scenarios ranging from "decontaminating a set of tunnels" to "capturing an intruder in a network".

Using the original metaphor [34, 11], 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. 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 main complexity measure is the number of searchers used by the strategy. The smallest number of searchers for which a search strategy exists for a graph G is called the *search number* $\mathbf{s}(G)$ of G.

An interesting line of investigation is the determination of efficient search strategies satisfying *additional* properties, which are desirable or even necessary for some applications. Two properties are of particular interest: absence of recontamination, and connectivity of the cleared area.

A search strategy is *monotone* if no recontamination ever occurs. The importance of monotone searching arises in applications where the cost of clearing an edge by far exceeds

the cost of traversing an edge. Hence each edge should be cleared only once. Lapaugh [17] has proved that for every G there is always a monotone search strategy that uses $\mathbf{s}(G)$ searchers. A short and elegant proof of this result was found by Bienstock and Seymour [47, 16].

A search strategy is *connected* if the set of clear edges is always connected. 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 cleared edge. The necessity for connectivity 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. Moreover, the same condition should be imposed in cases where the searchers cannot "jump" from one node to a non-adjacent one (e.g., cannot pass through the "walls" that determine the structure of the graph where the search takes place). Safety is another motivation for connectivity, as it would always ensure the presence of secure routes between all the searchers. We denote by $\mathbf{cs}(G)$, the *connected search number* of graph G, the minimum number of searchers required to clear all edges of G by making use of connected strategy. Correspondingly, the *monotone connected search number* of graph G, $\mathbf{mcs}(G)$, is the minimum number of searchers required to clear all edges of G by making use of connected and monotone strategy.

3. Price of connectivity

The main result of this section is the following theorem.

Theorem 1. *For any n-vertex graph G, $\mathbf{cs}(G)/\mathbf{s}(G) = O(\log n)$.*

To prove the theorem, we introduce several auxiliary notions and statements.

Branchwidth. A *branch decomposition* [48] of a graph $G = (V, E)$ is a tree T with all internal vertices of degree 3 and with a one-to-one correspondence between the leaves of T and the edges of G. Given an edge e of T, removing e from T results in two trees $T_1^{(e)}$ and $T_2^{(e)}$, and an *e-cut* is defined as the pair $\{E_1^{(e)}, E_2^{(e)}\}$, where $E_i^{(e)} \subset E$ is the set of leaves of $T_i^{(e)}$ for $i = 1, 2$. Note that $E_1^{(e)} \cap E_2^{(e)} = \emptyset$ and $E_1^{(e)} \cup E_2^{(e)} = E$. The *width* of T is defined as $\omega(T) = \max_e \delta(E_1^{(e)})$ where the maximum is taken over all e-cuts in T, see Figure 1. The *branchwidth* of G is then $\mathrm{bw}(G) = \min_T \omega(T)$, where the minimum is taken over all branch decompositions T of G. For the purpose of our proof, we define the following notion.

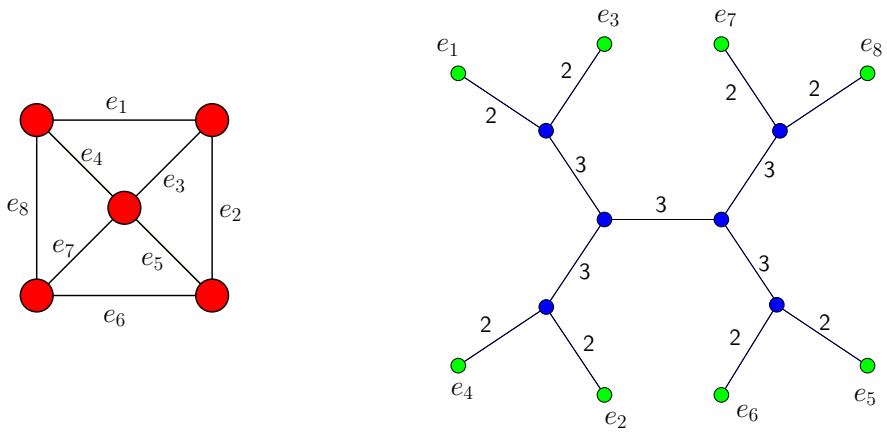

Figure 1: A graph and one of its branch decompositions, of width 3.

Definition 1. *A branch decomposition T of a graph G is* connected *if for every e-cut in T each of the resulting two sets of edges forms a connected subgraph of G.*

Let T be a branch decomposition of a graph $G = (V, E)$. For a subtree A resulting from an e-cut of a branch decomposition T, we use $E(A)$ to denote the edge subset of E corresponding to the leaves of A. For two disjoint sub-trees A and B of T, we denote by $\partial(A, B)$ the subset of vertices of V having at least one incident edge in $E(A)$ and at least one incident edge in $E(B)$. In other words, $\partial(A, B)$ is the set of vertices in G separating edges corresponding to leaves of A from edges corresponding to the leaves of B. We denote by $\delta(A, B) = |\partial(A, B)|$ the number of vertices in $\partial(A, B)$.

Definition 2. *A* quartet *in a branch decomposition T is an ordered set (A_1, A_2, B_1, B_2) of four mutually disjoint subtrees of T satisfying the following:*

1. *there is an edge $e = \{x, y\}$ of T such that the roots a_1 and a_2 of A_1 and A_2 are both adjacent to x in T, and the roots b_1 and b_2 of B_1 and B_2 are both adjacent to y in T; (cf. the left graph in Figure 2.)*
2. *$\partial(A_1, B_1) \neq \emptyset$ and $\partial(A_2, B_2) \neq \emptyset$;*
3. *$\partial(A_1, A_2) = \emptyset$;*

Let us notice that by the above definition, the leaves corresponding to the subtrees A_1, A_2, B_1, B_2 form a 4-partition of E.

In Figure 3, we provide Algorithm Make-it-Connected which proceeds as follows. Given a quartet (A_1, A_2, B_1, B_2) in S, the algorithm replaces this quartet by (A_1, B_1, A_2, B_2), resulting in a tree S' obtained by connecting a_1 and b_1 to x, and a_2 and b_2 to y (See Fig. 2).

Actually, if (A_1, A_2, B_1, B_2) and (A_1, A_2, B_2, B_1) are both quartets in S, then the algorithm considers the two possible replacements, and chooses the one that has smaller width. Clearly S' is also a branch decomposition of G. Note however that neither (A_1, B_1, A_2, B_2), nor (A_2, B_2, A_1, B_1) is a quartet in S' since $\partial(A_i, B_i) \neq \emptyset$ for $i = 1, 2$, by the definition of the quartet (A_1, A_2, B_1, B_2) in S. Algorithm Make-it-Connected proceeds by successive replacements of quartets in the branch decomposition. The algorithm stops when there is no quartet in the current branch-decomposition (we later prove that such a situation eventually occurs).

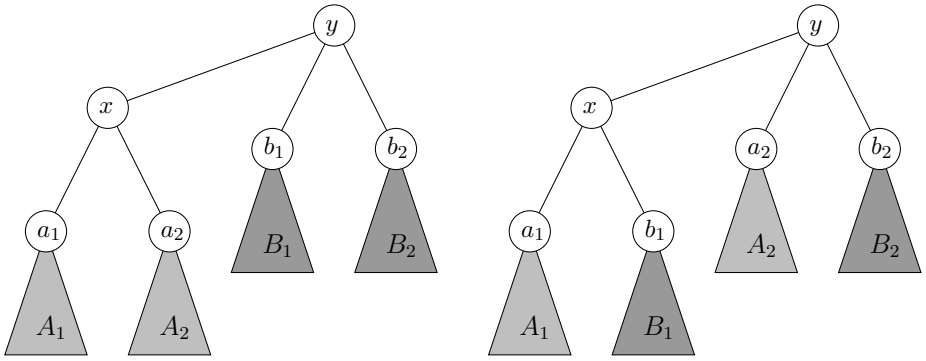

Figure 2: Replacing quartets in Make-it-Connected

The next Lemma is the most crucial part in the proof of Theorem 1.

Lemma 1. *Let T be a branch decomposition of a 2-edge-connected graph G with width k. Then algorithm Make-it-Connected returns a connected branch decomposition T' of G with width at most k, in time $O(m^3)$.*

Proof. The proof of the lemma proceeds through a sequence of claims.

Claim 1. *The replacement of a quartet as specified in Algorithm Make-it-Connected does not increase the width of the branch-decomposition.*

Proof. The only possible change in the width can occur because of the cut separating $A_1 \cup A_2$ from $B_1 \cup B_2$. We consider two cases depending whether or not (A_1, A_2, B_2, B_1) is a quartet. If (A_1, A_2, B_2, B_1) is also a quartet (i.e., $\partial(A_1, B_2) \neq \emptyset$ and $\partial(A_2, B_1) \neq \emptyset$), then

$$\delta(A_1 \cup B_1, A_2 \cup B_2) + \delta(A_1 \cup B_2, A_2 \cup B_1) \leq \delta(E(B_1)) + \delta(E(B_2)). \tag{2}$$

Indeed, if $u \in \partial(A_1 \cup B_1, A_2 \cup B_2) \setminus \partial(E(B_1))$ then $u \in \partial(E(B_2))$ because $\partial(A_1, A_2) = \emptyset$. Similarly, if $u \in \partial(A_1 \cup B_2, A_2 \cup B_1) \setminus \partial(E(B_2))$, then $u \in \partial(E(B_1))$. Therefore, every vertex

counted on the left hand side of Equation 2 appears as many times on the right hand side. Thus Equation 2 holds. Hence $\delta(A_1 \cup B_1, A_2 \cup B_2) + \delta(A_1 \cup B_2, A_2 \cup B_1) \leq 2k$, and thus the smallest of the two boundaries is of size $\leq k$. If (A_1, A_2, B_2, B_1) is not a quartet, then assume, w.l.o.g., that $\partial(A_1, B_2) = \emptyset$. We get $\delta(A_1 \cup B_1, A_2 \cup B_2) \leq \delta(E(B_1)) \leq k$. $\diamond$

Claim 2. *The branch decomposition T' returned by Algorithm Make-it-Connected is connected.*

```
input: branch decomposition T of a 2-edge-connected graph of width k;
output: connected branch decomposition T' of width ≤ k;
begin
    S := T;
    while there exists a quartet (A1, A2, B1, B2) in S do
        replace (A1, A2, B1, B2) in S by (A1, B1, A2, B2) to get S';
        if (A1, A2, B2, B1) is not a quartet in S then S := S'
        else
            replace (A1, A2, B1, B2) in S by (A1, B2, A2, B1) to get S'';
            if ω(S') ≤ ω(S'') then S := S' else S := S'';
        endif
    endwhile
    T' := S;
end
```

Figure 3: Algorithm Make-it-Connected

Proof. Targeting towards a contradiction, let us assume that there is an e-cut that splits T' into two subtrees A and B such that the edges of $E(A)$ form a disconnected subgraph of G. Then there exists an e-cut such that A is the union of two disjoint subtrees A_1 and A_2 with $\partial(A_1, A_2) = \emptyset$. Among all such cuts, we choose an e-cut with the maximum number $|E(A)|$. The other subtree B of this e-cut contains at least two leaves because otherwise removing the single edge corresponding to the leaf of B would result in disconnecting the graph G, a contradiction with the fact that G is 2-edge-connected. Thus B is the union of two disjoint subtrees B_1 and B_2 which roots are adjacent to the root of B. Since $|E(A)|$ is maximum, we have that $\partial(A, B_i) \neq \emptyset$ for $i = 1, 2$. Moreover, since G is connected, and $\partial(A_1, A_2) = \emptyset$, we have $\partial(A_i, B) \neq \emptyset$ for $i = 1, 2$. Therefore, either $\partial(A_1, B_1) \neq \emptyset$ and $\partial(A_2, B_2) \neq \emptyset$, or $\partial(A_1, B_2) \neq \emptyset$ and $\partial(A_2, B_1) \neq \emptyset$, or both. In each of the cases, there is a quartet in T',

which is a contradiction because the tree returned by Algorithm Make-it-Connected has no quartet. ◇

Claim 3. *Algorithm Make-it-Connected terminates.*

Proof. We use a potential argument, based on a measure defined in [48] for carvings. Any internal vertex x of the branch-decomposition S is of degree 3, and thus it defines three subtrees S_1, S_2, S_3 whose roots are connected to x. Then let

$$\phi(S_1, S_2, S_3) = \begin{cases} 0, & \text{if } \partial(S_i, S_j) \neq \emptyset \text{ for any } i \neq j; \\ |E(S_\ell)| - 1, & \text{if } \partial(S_i, S_j) = \emptyset \text{ for some } i \neq j, \text{ where } \ell \notin \{i, j\}. \end{cases}$$

This function is well defined: G is connected and thus, if $\partial(S_i, S_j) = \emptyset$ for some $i \neq j$, then $\partial(S_i, S_\ell) \neq \emptyset$ and $\partial(S_j, S_\ell) \neq \emptyset$ for $\ell \in \{1, 2, 3\} \setminus \{i, j\}$. Now, we define a potential function ϕ defined on any branch-decomposition S with set of internal vertices $I(S)$ by

$$\phi(S) = \sum_{x \in I(S)} \phi(S_1^x, S_2^x, S_3^x).$$

We show that after every step of Algorithm Make-it-Connected ϕ strictly decreases. For that purpose, it is sufficient to prove that

$$\phi(A_1 \cup B_1, A_2, B_2) + \phi(A_1, B_1, A_2 \cup B_2) < \phi(A_1, A_2, B_1 \cup B_2) + \phi(A_1 \cup A_2, B_1, B_2)$$

for every quartet (A_1, A_2, B_1, B_2). By the definition of a quartet, $\partial(A_1, A_2) = \emptyset$, and we get that $\phi(A_1, A_2, B_1 \cup B_2) = |E(B_1)| + |E(B_2)| - 1$. Hence, for

$$L = \phi(A_1 \cup B_1, A_2, B_2) + \phi(A_1, B_1, A_2 \cup B_2) \text{ and } R = \phi(A_1 \cup A_2, B_1, B_2) + |E(B_1)| + |E(B_2)| - 1,$$

it is enough to prove that $L < R$ for every quartet (A_1, A_2, B_1, B_2). Since $R > 0$, the lemma holds if $L = 0$. Thus we restrict our analysis to the case $L > 0$, which means that either $\phi(A_1 \cup B_1, A_2, B_2) > 0$ or $\phi(A_1, B_1, A_2 \cup B_2) > 0$. W.l.o.g. we can examine the case where $\phi(A_1, B_1, A_2 \cup B_2) > 0$ which excludes that $\partial(A_1, A_2 \cup B_2) \neq \emptyset$ and $\partial(B_1, A_2 \cup B_2) \neq \emptyset$ hold simultaneously. Hence we consider two cases:

Case 1: $\partial(A_1, A_2 \cup B_2) = \emptyset$. If $\partial(A_1 \cup B_1, A_2) \neq \emptyset$ and $\partial(A_1 \cup B_1, B_2) \neq \emptyset$ then $L = |E(B_1)| - 1 < |E(B_1)| + |E(B_2)| - 1 \leq R$. Therefore, we consider two sub-cases:

- If $\partial(A_1 \cup B_1, A_2) = \emptyset$, then $L = |E(B_1)| + |E(B_2)| - 2 < |E(B_1)| + |E(B_2)| - 1 \leq R$.

- If $\partial(A_1 \cup B_1, B_2) = \emptyset$, then $\partial(B_1, B_2) = \emptyset$, and thus $R = |E(A_1)| + |E(A_2)| + |E(B_1)| + |E(B_2)| - 2$. It follows that $L < R$ because, by the definition of a quartet, $\partial(A_i, B_i) \neq \emptyset$ for every $i = 1, 2$.

Case 2: $\partial(B_1, A_2 \cup B_2) = \emptyset$. Then $\partial(B_1, B_2) = \emptyset$, and thus $R = |E(A_1)| + |E(A_2)| + |E(B_1)| + |E(B_2)| - 2$. Hence, $L < R$ because $\partial(A_i, B_i) \neq \emptyset$ for every $i = 1, 2$.

In all cases, the inequality $L < R$ holds, which completes the proof. $\diamond$

Now everything is settled to conclude with the proof of the lemma. By Claims 1 and 2, if Algorithm Make-it-Connected terminates, then it returns a connected branch decomposition of width $\leq k$. By Claim 3, the algorithm terminates. To compute the execution time of the algorithm, let us consider the potential function ϕ defined in the proof of Claim 3. Since $\phi(S_1, S_2, S_3) \leq m - 1$, we have that the potential cannot exceed $O(m^2)$. Thus there are $O(m^2)$ updates of the branch-decomposition. Each update is local to the subtree of six vertices interconnecting the roots a_1, a_2, b_1, b_2 of A_1, A_2, B_1, B_2. For each of the edges $\{x, a_1\}, \{x, b_1\}, \{y, a_2\}, \{y, b_2\}$, deciding whether the edge defines a quartet takes $O(m)$ times. Thus Algorithm Make-it-Connected terminates in $O(m^3)$ steps. $\square$

Expansions is a convenient tool for addressing the graph searching problem.

Connected Expansions. Let $G = (V, E)$ be a graph (with possible multiple edges and loops), and let $n = |V|$, and $m = |E|$. A *k-expansion* in G is a sequence $X_0, X_1, \ldots, X_r$ where $X_i \subseteq E$ for every $i = 0, \ldots, r$, $X_0 = \emptyset$, $X_r = E$, and satisfying the following:

- $|X_{i+1} \setminus X_i| \leq 1$ for every $i = 0, \ldots, r - 1$;
- $\delta(X_i) \leq k$ for every $i = 0, \ldots, r$.

A k-expansion $X_0, X_1, \ldots, X_r$ is *monotone*, if $X_{i+1} \subset X_i$ for every $i = 0, \ldots, r - 1$. Using the terminology of [16], a k-expansion is hence a crusade of frontier at most k. In this paper we deal only with expansions in graphs, however this notion can be defined for more general structures as well [18].

A k-expansion $X_0, X_1, \ldots, X_r$ of a graph G is *connected* if for each $i = 1, \ldots, r$, the subgraph formed by edges X_i is connected.

Lemma 2. *Given a connected branch decomposition T of width k for a graph G, one can compute in $O(m^3)$-time a monotone connected $(k \log_2 m)$-expansion $X_0, X_1, \ldots, X_m$ in G .*

Proof. We prove by induction on the number m of edges in G a slightly stronger statement: Given a connected branch decomposition T of width k for a graph G, and given any edge e of G, one can compute in $O(m^3)$-time a monotone connected $(k \log_2 m)$-expansion $X_0, X_1, \ldots, X_m$ in G with $X_1 = \{e\}$.

For any $m \geq 1$, let $\mathcal{P}_m$ be the following property: for any $k \geq 0$, given a connected branch decomposition T of width k for a graph G with m edges, and given any edge e of G, one can compute in $O(m^3)$-time a monotone connected $(k \log_2 m)$-expansion $X_0, X_1, \ldots, X_m$ in G with $X_1 = \{e\}$. We now show that $\mathcal{P}_m$ is satisfied.

If $m = 1$, then $\emptyset = X_0, X_1$, where X_1 is the only edge of G, is the connected 0-expansion in G, and thus $\mathcal{P}_1$ holds. If $m = 2$, then G is either a path of three vertices, or two vertices linked by an edge and a loop attached to one of the two vertices, or two loops attached to a vertex, or two vertices connected by a double edge. In the first three cases, $\mathrm{bw}(G) = 1$. In the latter case $\mathrm{bw}(G) = 2$. In all cases though, one can construct a connected $\mathrm{bw}(G)$-expansion X_0, X_1, X_2 in G starting from any edge. Thus $\mathcal{P}_2$ holds.

For $m > 2$, let us assume that $\mathcal{P}_q$ holds for every $2 \leq q \leq m-1$. There is a vertex x of T whose removal results in three disjoint subtrees T_1, T_2, and T_3, with $|E_i| \leq \lfloor m/2 \rfloor$ for every $i \in \{1, 2, 3\}$, where $|E_i|$ is the number of leaves of T_i. Since T is a connected branch decomposition, we have that the leaves of each of these subtrees form three connected subgraphs G_1, G_2, G_3 of G, and, for any $i \in \{1, 2, 3\}$, T_i is a connected branch decomposition of G_i. Given a set of edges X in $G_i = (V_i, E_i)$, we denote by $\partial_{G_i}(X)$ the set of vertices of G_i that has at least one incident edge in X, and at least one incident edge in $E_i \setminus X$.

Because $m > 2$ and $\lfloor m/2 \rfloor < m - 1$, we have that each G_i (with $m_i \leq \lfloor m/2 \rfloor$ edges) satisfies the induction assumption. Let e be an edge of G. Edge e belongs to some G_i. Assume, w.l.o.g., that e is an edge of G_1. By the induction assumption, there is a monotone connected $(k \log_2 m_1)$-expansion $X_0, X_1, \ldots, X_{m_1}$ in G_1 with $X_1 = \{e\}$. Removing the edge connecting x to the root of T_3 in T results in a connected subgraph $G_1 \cup G_2$. Thus, there is a vertex u in G that has at least one incident edge in G_1, and at least one incident edge f in G_2. Again, by the induction assumption, there is a $(k \log_2 m_2)$-expansion $Y_0, Y_1, \ldots, Y_{m_2}$ in G_2 with $Y_1 = \{f\}$. Finally, since G is connected and $G = G_1 \cup G_2 \cup G_3$, we have that there is a vertex v in G that has at least one incident edge in $G_1 \cup G_2$, and at least one incident edge g in G_3. By the induction assumption, we can select a monotone connected $(k \log_2 m_3)$-expansion $Z_0, Z_1, \ldots, Z_{m_3}$ in G_3 with $Z_1 = \{g\}$. By putting three expansions together, we obtain a monotone connected expansion

$$X_0, X_1, \ldots, X_{m_1}, X_{m_1} \cup Y_1, \ldots, X_{m_1} \cup Y_{m_2}, X_{m_1} \cup Y_{m_2} \cup Z_1, \ldots, X_{m_1} \cup Y_{m_2} \cup Z_{m_3}$$

in G. It remains to bound the "frontier" of this expansion. We have

$$\delta(X_i) \leq \delta_{G_1}(X_i) + k \leq k \log_2 m_1 + k \leq k \log_2 \lfloor m/2 \rfloor + k \leq k \log_2 m/2 + k \leq k \log_2 m.$$

We also have

$$\delta(X_{m_1} \cup Y_i) \leq \delta_{G_2}(Y_i) + \delta(X_{m_1} \cup Y_i, G_3) \leq k \log_2 \lfloor m/2 \rfloor + k \leq k \log_2 m.$$

Finally, we have

$$\delta(X_{m_1} \cup Y_{m_2} \cup Z_i) \leq \delta_{G_3}(Z_i) + \delta(Z_i, G_1 \cup G_2) \leq k \log_2 \lfloor m/2 \rfloor + k \leq k \log_2 m.$$

Hence $\mathcal{P}_m$ is satisfied.

To complete the proof of the lemma, we observe that the time $\tau(m)$ needed to construct the expansion in an m-edge graph satisfies $\tau(m) \leq 3\tau(m/2)$, and thus the complexity of the construction is $3^{\log_2 m} = O(m^3)$. □

Now everything is ready to proceed with the proof of the main result of this section.

Proof of Theorem 1. If $\mathrm{bw}(G) \leq 1$, then G is isomorphic to $K_{1,p}$, the tree with at most one non-leaf vertex. In this case, $\mathbf{cs}(G) = \mathbf{s}(G)$.

$$\mathbf{cs}(G) = \mathbf{s}(G) = \begin{cases} 0, & \text{if } p = 0; \\ 1, & \text{if } p \leq 2; \\ 2, & \text{if } p > 2 \end{cases}$$

Claim 4. *Given a branch decomposition T of width $k \geq 2$ for a connected graph G, one can compute in $O(m^3)$-time a monotone connected $k(1 + \log_2 m)$-expansion $X_0, X_1, \ldots, X_m$ in G.*

Proof. Indeed, if G is 2-edge-connected, then by application of Lemma 1, one can compute in $O(m^3)$ time a connected branch decomposition T' of G of width $\leq k$. The requested expansion is then obtained by Lemma 2. If G is not 2-edge-connected, then we add a double edge to each isthmus (i.e., cut-edge) in G so that the resulting graph G' is 2-edge-connected. Since $k \geq 2$, we obtain that $\mathrm{bw}(G') = \mathrm{bw}(G)$. More precisely, given a branch decomposition T of G, one can construct a branch decomposition T' of G' such that $\omega(T') = \omega(T)$. We construct G' and T', and then compute a connected branch decomposition T'' of G' in time $O(m^3)$. We have $\omega(T'') \leq \omega(T') \leq \omega(T) = k$. By application of Lemma 2, we obtain a monotone connected $(k \log m')$-expansion in G'. By removing in this expansion the second

occurrence of every double edge added to isthmuses, we obtain a connected $(k \log_2 m')$-expansion in G. We complete the proof by noticing that $m' \leq 2m$. $\diamond$

Finally, by Claim 4, there is a monotone connected $k(1 + \log_2 m)$-expansion $X_0, X_1, \ldots, X_m$ in G. This expansion can be transformed into a monotone connected strategy of $k(1+\log_2 m)+1$ searchers as follows. Suppose that for some $i \geq 1$, $k(1+\log_2 m)+1$ searchers are able to clean the set of edges X_i and that the only vertices occupied by the searchers at this step are the vertices of $\partial(X_i)$. Moreover, every vertex of $\partial(X_i)$ contains exactly one searcher. For X_1, which consists of one edge, $1 \leq \delta(X_1) \leq 2$, and this condition clearly holds. Let $e = \{x, y\} = X_{i+1} \setminus X_i$. Expansion $X_0, X_1, \ldots, X_m$ is connected and thus at least one of the endpoints of e, say x, is in $\partial(X_i)$. Because $\delta(X_i) \leq k(1 + \log_2 m)$, there is at least one unused searcher. We put this searcher on x, clear e by sliding from x to y, and then remove searchers from all vertices of $\partial(X_i) \setminus \partial(X_{i+1})$. We also remove one searcher from y if $y \in \partial(X_i) \cap \partial(X_{i+1})$, and thus arrive to the situation when all edges of X_{i+1} are cleared, and the searchers occupy the vertices of $\partial(X_{i+1})$. By repeating these arguments for each $1 \leq i \leq m - 1$, we construct a monotone search strategy of at most $k(1 + \log_2 m) + 1$ searchers. To conclude, for any graph G, $\mathrm{bw}(G) \leq \mathbf{s}(G)$ [16, 25]. Hence, we construct a monotone search strategy of at most $\mathbf{s}(G)(1 + \log_2 m) + 1$ searchers. $\square$

Let us remark, that the proof of Theorem 1 implies a more general result, namely,

$$\mathbf{mcs}(G)/\mathbf{s}(G) = O(\log n).$$

4. Connected search in trees

4.1. Price of connectivity in trees

In this section we show that the price of connectivity in trees is at most 2. In particular, we prove the following theorem.

Theorem 2. *For any tree T that is not a line, it holds that $\mathbf{s}(T) \leq \mathbf{cs}(T) \leq 2 \cdot \mathbf{s}(T) - 2$ and this inequality is tight.*

Our first step is the following lemma on the monotonicity of *cs* on trees.

Lemma 3. *For any tree T with connected search number at most k, there exists a monotone connected search strategy for T using at most k searchers. In other words, $\mathbf{cs}(T) = \mathbf{mcs}(T)$ for any tree T.*

Proof. Let T be a tree, with $\mathbf{cs}(T) \leq k$. Let $\widehat{T}$ be the tree obtained from T by subdividing every edge e of T into two consecutive edges e' and e''. Consider a connected search strategy for T, and replace every slide action along an edge e in this strategy by two consecutive slide actions along the corresponding edges e' and e''. That way we get a connected search strategy for $\widehat{T}$ using the same number of searchers as the original strategy for T. As a consequence,

$$\mathbf{cs}(\widehat{T}) \leq \mathbf{cs}(T).$$

Claim 5. *There exists a connected k-expansion in $\widehat{T}$.*

Proof. To prove the claim, let us consider a connected search strategy S in $\widehat{T}$ using at most k searchers. Let $F = X_0, X_1, \ldots, X_t$ be the sequence of subsets of edges defined as follows: $X_0 = \emptyset$, and, for $i \geq 1$, X_i is the set of clear edges after step i of S. Since at most one edge is cleared at every step of S, it follows that $|X_i \setminus X_{i-1}| \leq 1$, i.e., F is an expansion. As S is using at most k searchers, we obtain that the frontier of each set in F is of size at most k. Finally, all X_i's are connected for $1 \leq i \leq t$ because S is a connected strategy. $\diamond$

The core of the proof of the lemma is the following result:

Claim 6. *There exists a monotone connected k-expansion in $\widehat{T}$.*

Proof. By Claim 5, there exists a connected k-expansion in $\widehat{T}$. Let us choose a connected k-expansion $X_0, X_1, \ldots, X_t$ in $\widehat{T}$ that satisfies:

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

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

Let us show that $X_0, X_1, \ldots, X_t$ is monotone. If for some $i \geq 1$, $|X_i \setminus X_{i-1}| = 0$, i.e., if $X_i \subseteq X_{i-1}$, then $X_0, X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_t$ is also a connected k-expansion, contradicting (C1). Therefore $|X_i \setminus X_{i-1}| = 1$ for every $i \geq 1$. Now, we show that $X_{i-1} \subseteq X_i$ for every $i \geq 1$. First, observe that

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

for every $i \geq 1$. Indeed, assume for the purpose of contradiction that $\delta(X_{i-1} \cup X_i) < \delta(X_i)$. Then $X_{i-1} \cup X_i$ is connected because otherwise $\delta(X_{i-1} \cup X_i) = \delta(X_{i-1}) + \delta(X_i) \geq \delta(X_i)$. Therefore,

$$X_0, X_1, \ldots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \ldots, X_r$$

is a connected k-expansion because $|(X_{i-1} \cup X_i) \setminus X_{i-1}| = |X_i \setminus X_{i-1}| \leq 1$ and $|X_{i+1} \setminus (X_{i-1} \cup X_i)| \leq |X_{i+1} \setminus X_i| \leq 1$. This connected k-expansion would yield a contradiction with (C1), and Equation 3 follows. To prove $X_{i-1} \subseteq X_i$ for every $i \geq 1$, we use the submodularity of the connectivity function δ, stating that for any two edge-sets A and B of any graph, $\delta(A \cap B) + \delta(A \cup B) \leq \delta(A) + \delta(B)$. By combining submodularity with Equation 3, we get that $\delta(X_{i-1} \cap X_i) \leq \delta(X_{i-1})$ for every $i \geq 1$. Therefore

$$X_0, X_1, \ldots, X_{i-2}, X_{i-1} \cap X_i, X_i, \ldots, X_t$$

is a k-expansion because $|(X_{i-1} \cap X_i) \setminus X_{i-2}| \leq |X_{i-1} \setminus X_{i-2}| \leq 1$ and $|X_i \setminus (X_{i-1} \cap X_i)| = |X_i \setminus X_{i-1}| \leq 1$. The fact that this expansion is connected follows from the fact that both X_{i-1} and X_i are subtrees of $\widehat{T}$, and therefore their intersection is also a subtree of $\widehat{T}$, and thus connected. By (C2), $|X_{i-1} \cap X_i| \geq |X_{i-1}|$, and thus $X_{i-1} \subseteq X_i$. Therefore, the considered expansion is a monotone connected k-expansion, which completes the proof. $\diamond$

Claim 7. *There exists a monotone connected search strategy in $\widehat{T}$ using at most k searchers.*

Proof. Let $X_0, X_1, \ldots, X_t$ be a monotone connected k-expansion in $\widehat{T}$ whose existence is guaranteed by Claim 6. For $i = 1, \ldots, t$, let $e_i = \{x_i, y_i\} = X_i \setminus X_{i-1}$. If one of the endpoints of e_i is of degree 1, then x_i is set to be that vertex. We construct a monotone connected search strategy that successively clears the edges $e_1, e_2, \ldots, e_t$, as follows. Initially, place k searchers in x_1. Clear edge e_1 by sliding one searcher from x_1 to y_1 along e_1. No recontamination occurs because either $k = 1$ and x_1 is incident to e_1 only, or $k > 1$ and $k - 1$ searchers remains at x_1. Assume now that all edges $e_1, \ldots, e_{i-1}$ have been cleared (without recontamination). The edge $e_i = \{x_i, y_i\}$ is incident to X_{i-1} because the expansion $X_0, X_1, \ldots, X_t$ is connected. Assume, w.l.o.g., that $x_i \in \partial(X_{i-1})$. If $\delta(X_{i-1}) < k$, then there is at least one free searcher, which can be slid from x_i to y_i along e_i to clear that edge without recontamination. If $\delta(X_{i-1}) = k$, then we claim that at least one endpoint of e_i is not in $\partial(X_i)$. Indeed, since $x_i \in \partial(X_{i-1})$, if $x_i \in \partial(X_i)$ then $\deg(x_i) > 2$. As a consequence, y_i has to be of degree exactly 2 because every edge has one of its endpoints incident to exactly one other edge. If in turn $y_i \in \partial(X_i)$ then the unique edge $f_i \neq e_i$ incident to y_i is contaminated, and therefore $y_i \notin \partial(X_{i-1})$. We get a contradiction because then $\delta(X_i) = \delta(X_{i-1}) + 1 > k$, contradicting the fact that we are dealing with a k-expansion. So, as claimed, at least one endpoint of e_i is not in $\partial(X_i)$. If $x_i \in \partial(X_i)$ and $y_i \notin \partial(X_i)$ then, as shown before, $\deg(y_i) = 2$, and the unique edge $f_i \neq e_i$ incident to y_i belongs to X_{i-1}. Therefore, there is at least one searcher occupying y_i, which can be slid along e_i to clear it without recontamination. If $x_i \notin \partial(X_i)$

then this vertex is occupied by at least one searcher because $x_i \in \partial(X_{i-1})$. This searcher can be slid along e_i from x_i to y_i, clearing e_i without recontamination. One proceeds that way until all edges have been cleared. During the process, no recontamination occurs and the successive sets of clear edges are always connected. Hence, the constructed search strategy is monotone and connected. By construction, it uses at most k searchers. $\diamond$

To complete the proof of Lemma 3, we observe that we can directly get a monotone connected search strategy in T using at most k searchers from the monotone connected search strategy in $\widehat{T}$ using at most k searchers by merely replacing the clearing of each edge in $\widehat{T}$ by the clearing of the corresponding subdivided edge in T. $\square$

The proof of Theorem 2 is based on the notion of *k-caterpillar* recursively defined as follows:

- a 0-caterpillar is a path graph,
- for $k \geq 1$, a graph G is a k-caterpillar if it is a tree containing a path P, called *spine*, such that, for any connected component C of $G \setminus V(P)$, C is a $(k-1)$-caterpillar with an extremity of its spine adjacent to a vertex in P.

The *spine* of a 0-caterpillar is the graph itself. Notice that, according to the above definition, an 1-caterpillar is 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. Clearly, every tree is a k-caterpillar for k large enough. The notion of k-caterpillar is related to the notion of *caterpillar dimension* introduced in [49] (see also [50]).

Our second step for the proof of Theorem 2 is to prove that k-caterpillars are exactly the graphs that can be connectedly cleared with at most $k+1$ searchers.

Given a tree T and two vertices v, w of T, we denote 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 length of a path from its root to the leaves. We denote by B_k the complete binary tree of depth k rooted on its unique vertex of degree 2, and by D_k the tree obtained by connecting the three roots of three copies of B_{k-1} to a unique new root vertex. We denote by $T_1 \preceq T_2$ the relation "T_1 is a contraction of T_2". Finally, given a tree T_1 rooted on x_1 and a tree T_2 containing a vertex x_2, 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 vertex x_1 is either x_2 or the result of contracting a series of edges, some of them containing x_2 as end-point.

Lemma 4. *Let T be a tree and e be an edge of it. Let also T' be the tree created if we contract e in T. Then* $\mathbf{mcs}(T') \leq \mathbf{mcs}(T)$.

Proof. Let w_e be the vertex created in T' after contracting $e = \{v, u\}$. Given a monotone search strategy $\mathcal{S}'$ for T that uses $\leq k$ searchers with n steps, we make a strategy $\mathcal{S}'$ for T' as follows: Suppose that e is cleaned at the ith step of this strategy. Then, the new strategy is constructed by taking the first $i-1$ steps of $\mathcal{S}$ followed by the last $n-i-1$ steps of $\mathcal{S}$ and then replacing each action concerning v or u with a same type action concerning w_e. It is easy to verify that the new strategy is monotone and that it uses at most k searchers. □

Lemma 5. *For any tree T and $k \geq 1$, the following three properties are equivalent:*

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

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

(3) $\mathbf{cs}(T) \geq k+1$.

Proof. (1) ⇒ (2): We start by a preliminary statement. 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 induces 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 statement, it is enough to prove that if $D_k \not\preceq T$, then T contains a vertex v such that $B_k \not\preceq_v T$. Towards a contradiction, assume that for every vertex v 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 $B_k \preceq_{z_1} T_z[z_1]$ or $B_k \preceq_z T_{z_1}[z]$. In both cases, we get $D_k \preceq T$, a contradiction.

(2) ⇒ (3): We first claim that $\mathbf{mcs}(D_k) \geq k+1$. For this, we prove that, for any connected search strategy in D_k, there is a step in which at least $k+1$ searchers are required for a monotone search of T. As this is obvious when $k = 1$, we assume that $k > 1$.

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 an edge e incident to the root of D_k is being cleaned. Assume, w.l.o.g., that e is an edge of T_1, which means that after step i_1, T_2 and T_3 are still completely contaminated. Let $i_2 > i_1$ be the first step during which an edge incident to a leaf f of T_2 or T_3, say of T_2 is reached by a searcher. The path P from the root r to this leaf should be clean and has length k. Moreover, at step i_2, 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 for a monotone search. Moreover, there is one additional searcher used to clear f. Hence, at least $k+1$ searchers are required at step i_2.

Applying inductively Lemma 4 for the edges that are contracted in T in order to create D_k, one can prove that $\mathbf{mcs}(T) \geq \mathbf{mcs}(D_k)$. Therefore $\mathbf{mcs}(T) \geq k+1$ and the result follows from Lemma 3.

(3) $\Rightarrow$ (1): We show the stronger statement 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 spine $P_{i,j}$ 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. □

We define $\mathcal{M}_k = \{G \mid \mathbf{s}(G) \leq k\}$ and $\mathcal{D}_k = \{G \mid \mathbf{cs}(G) \leq k\}$ and denote by $\mathbf{obs}(\mathcal{M}_k)$ (resp. $\mathbf{obs}(\mathcal{D}_k)$) the set of all the contraction minimal graphs that do not belong in $\mathcal{M}_k$ (resp. $\mathcal{D}_k$). According to Lemma 5, $\mathbf{obs}(\mathcal{D}_k)$ contains a unique graph that is D_k. This comes to a contrast to the fact that the size of $\mathbf{obs}(\mathcal{M}_k)$ increases rapidly as shown by Parsons in [12] (in fact, $|\mathbf{obs}(\mathcal{M})| = 2^{\Omega(k \log k)}$, as indicated in [51]).

Proof of Theorem 2. Let T be a tree, and assume that $\mathbf{s}(T) \leq k$. 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., vertices at distance k from the root). Parsons [12] has proved that $M_k \in \mathbf{obs}(\mathcal{M}_k)$. Therefore $M_k \not\preceq T$.

Observe that 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 $D_{2k-2} \not\preceq T$, which implies, by Lemma 5, that $\mathbf{cs}(T) \leq 2k-2 = 2\mathbf{s}(T)-2$.

To prove that the bound is tight, we first consider D_{2k-1}. We have $\mathbf{s}(D_{2k-1}) \leq \mathbf{cs}(D_{2k-1}) = 2k$ and $M_k \preceq D_{2k-1}$, which implies that $\mathbf{s}(D_{2k-1}) \geq k+1$. 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 $\mathbf{cs}(D_{2k-1}) = 2\mathbf{s}(D_{2k-1}) - 2$.

Finally, let us consider the graph M_k. It is easy to observe that $D_{k+1} \not\preceq M_k$ and that $D_k \preceq M_k$ which, from Lemma 5, implies that $\mathbf{cs}(M_k) = k+1 = \mathbf{s}(M_k)$. □

4.2. Computing optimal connected search strategies in trees

In this section, we show that computing the connected search number of trees can be achieved in polynomial time.

Theorem 3. *There is a linear-time algorithm that, given any tree T, computes the connected search number and an optimal monotone connected search strategy for T.*

The proof of Theorem 3 is constructive. A monotone connected strategy depends on the choice of the initial vertex x where all searchers are originaly placed. For every tree T, let $\mathbf{cs}_x(T)$ denote the minimum number of searchers required to clear T by a monotone connected strategy starting from vertex x. Hence,

$$\mathbf{cs}(T) = \min_{x \in V(T)} \mathbf{cs}_x(T).$$

Our algorithm computes $\mathbf{cs}_x(T)$ for all vertices x of the tree T. In fact, it computes the related values $\mathbf{cs}^+_x(T)$, $x \in V(T)$, where

$$\mathbf{cs}^+_x(T) = \max\{1, \mathbf{cs}_x(T)\}.$$

The tree T rooted at vertex x is denoted by T_x. For a vertex y of T, let $T_x[y]$ denote the subtree of T_x rooted at y, consisting of y and all its descendants in T_x. Our algorithm relies mostly on the following lemma:

Lemma 6. *Let $y_1, y_2, \ldots, y_d$ be the $d \geq 1$ children of vertex y in the tree T_x.*

- *If $d = 1$ then $\mathbf{cs}^+_y(T_x[y]) = \mathbf{cs}^+_{y_1}(T_x[y_1])$.*

- *If $d \geq 2$ then by ordering the y_i's such that $\mathbf{cs}^+_{y_i}(T_x[y_i]) \geq \mathbf{cs}^+_{y_{i+1}}(T_x[y_{i+1}])$ for every i, $1 \leq i < d$, we have*

$$\mathbf{cs}^+_y(T_x[y]) = \max\{\mathbf{cs}^+_{y_1}(T_x[y_1]),\ \mathbf{cs}^+_{y_2}(T_x[y_2]) + 1\}.$$

Proof. First observe that $\mathbf{cs}_y(T_x[y]) \geq \mathbf{cs}_{y_1}(T_x[y_1])$ for $d \geq 1$. Indeed, $T_x[y_1]$ cannot be cleared in a monotone connected way by fewer than $\mathbf{cs}_{y_1}(T_x[y_1])$ searchers reaching subtree $T_x[y_1]$ by edge $\{y, y_1\}$.

Assume $d = 1$. Then, by the previous observation, we get $\mathbf{cs}^+_y(T_x[y]) \geq \mathbf{cs}^+_{y_1}(T_x[y_1])$. Conversely, $\mathbf{cs}^+_{y_1}(T_x[y_1])$ searchers are sufficient to clear $T_x[y]$ by a monotone connected strategy whenever y has a unique child y_1, by moving $\mathbf{cs}^+_{y_1}(T_x[y_1])$ searchers from y to y_1, and then using $\mathbf{cs}_{y_1}(T_x[y_1])$ searchers to clear $T_x[y_1]$. Thus $\mathbf{cs}_y(T_x[y]) \leq \mathbf{cs}^+_{y_1}(T_x[y_1])$, from which we derive $\mathbf{cs}^+_y(T_x[y]) \leq \mathbf{cs}^+_{y_1}(T_x[y_1])$. Hence the lemma holds for $d = 1$. So assume now that $d > 1$. We consider two cases.

Case 1: $\mathbf{cs}^+_{y_1}(T_x[y_1]) > \mathbf{cs}^+_{y_2}(T_x[y_2])$. Then $\mathbf{cs}^+_{y_1}(T_x[y_1]) = \mathbf{cs}_{y_1}(T_x[y_1]) \geq 2$. In that case, $\mathbf{cs}_{y_1}(T_x[y_1])$ searchers suffice to clear $T_x[y]$ starting from y, by clearing $T_x[y_1]$ last among the children of y, and by letting one searcher occupying vertex y while the other subtrees are successively cleared. Indeed, every subtree $T_x[y_i]$ with $i > 1$ requires at most $\mathbf{cs}^+_{y_2}(T_x[y_2]) < \mathbf{cs}_{y_1}(T_x[y_1])$ searchers to be cleared. So, $\mathbf{cs}_y(T_x[y]) \leq \mathbf{cs}_{y_1}(T_x[y_1])$, and thus $\mathbf{cs}^+_y(T_x[y]) \leq \mathbf{cs}^+_{y_1}(T_x[y_1])$. To prove equality, assume that $\mathbf{cs}^+_y(T_x[y]) < \mathbf{cs}^+_{y_1}(T_x[y_1])$. Then, since $\mathbf{cs}^+_{y_1}(T_x[y_1]) = \mathbf{cs}_{y_1}(T_x[y_1])$, we get $\mathbf{cs}_y(T_x[y]) < \mathbf{cs}_{y_1}(T_x[y_1])$, a contradiction.

Case 2: $\mathbf{cs}^+_{y_1}(T_x[y_1]) = \mathbf{cs}^+_{y_2}(T_x[y_2])$. First, we consider two sub-cases for proving $\mathbf{cs}^+_y(T_x[y]) \leq \mathbf{cs}^+_{y_2}(T_x[y_2]) + 1$.

- If $\mathbf{cs}_{y_1}(T_x[y_1]) = 0$ or $\mathbf{cs}_{y_2}(T_x[y_2]) = 0$ then $T_x[y]$ consists in a set of d paths pending from vertex y, in which case two searchers are sufficient to clear $T_x[y]$ starting from y. Therefore, $\mathbf{cs}_y(T_x[y]) \leq 2$, from which we derive $\mathbf{cs}^+_y(T_x[y]) \leq \mathbf{cs}^+_{y_2}(T_x[y_2]) + 1$.

- If $\mathbf{cs}_{y_1}(T_x[y_1]) \neq 0$ and $\mathbf{cs}_{y_2}(T_x[y_2]) \neq 0$ then $\mathbf{cs}^+_{y_1}(T_x[y_1]) = \mathbf{cs}_{y_1}(T_x[y_1])$ and $\mathbf{cs}^+_{y_2}(T_x[y_2]) = \mathbf{cs}_{y_2}(T_x[y_2])$. In this case, $\mathbf{cs}_{y_2}(T_x[y_2])+1$ searchers are sufficient to clear $T_x[y]$ from y by letting one searcher occupying vertex y while all subtrees are successively cleared. Indeed, every subtree $T_x[y_i]$ with $i = 1, \ldots, d$, requires at most $\mathbf{cs}^+_{y_i}(T_x[y_i]) \leq \mathbf{cs}^+_{y_2}(T_x[y_2]) = \mathbf{cs}_{y_2}(T_x[y_2])$ searchers to be cleared starting from y_i. Thus $\mathbf{cs}_y(T_x[y]) \leq \mathbf{cs}_{y_2}(T_x[y_2])+1$, and therefore $\mathbf{cs}^+_y(T_x[y]) \leq \mathbf{cs}^+_{y_2}(T_x[y_2]) + 1$.

To prove equality, let us assume, for the purpose of contradiction, that $\mathbf{cs}^+_y(T_x[y]) < \mathbf{cs}^+_{y_2}(T_x[y_2])+1$. Thus $\mathbf{cs}_y(T_x[y]) < \mathbf{cs}^+_{y_2}(T_x[y_2]) + 1$. It follows from this inequality that if $\mathbf{cs}^+_{y_2}(T_x[y_2]) = 1$

then $\mathbf{cs}_y(T_x[y]) \leq 1$, yielding a contradiction because $d > 1$. If otherwise $\mathbf{cs}^+_{y_2}(T_x[y_2]) > 1$ then we get $\mathbf{cs}_y(T_x[y]) < \mathbf{cs}_{y_2}(T_x[y_2]) + 1$. In other words, there exists a monotone connected search strategy using at most $\mathbf{cs}_{y_2}(T_x[y_2])$ searchers for $T_x[y]$ starting from y. Let $a \in \{1, 2\}$ be such that $T_x[y_a]$ is completely cleared before $T_x[y_b]$ is completely cleared, where $b \in \{1, 2\} \setminus \{a\}$. Then let t be the first step of the strategy at which $T_x[y_a]$ becomes completely cleared. During all steps t', $t' \leq t$, at least one searcher must occupy a vertex outside $T_x[y_a]$ because otherwise $\{y, y_a\}$ would be recontaminated. Thus clearing $T_x[y_a]$ from y_a has been achieved with strictly less than $\mathbf{cs}_{y_2}(T_x[y_2])$ searchers, a contradiction since $\mathbf{cs}^+_{y_2}(T_x[y_2]) > 1$ insures that $\mathbf{cs}_{y_1}(T_x[y_1]) \geq \mathbf{cs}_{y_2}(T_x[y_2])$. This completes Case 2, and the proof of the lemma. □

We remark that the above proof could become shorter (but also less self-contained) by making use of the results in Section 4.1. Note that a straightforward application of Lemma 6 enables to compute $\mathbf{cs}(T_x)$ in $O(n)$ time, resulting in an $O(n^2)$-time algorithm for computing $\mathbf{cs}(T)$. We show that this complexity can be reduced to $O(n)$, and, more importantly, that an optimal search strategy can also be computed in linear time.

Proof of Theorem 3. Let T be a tree. For every vertex x of T, we define the following labeling λ_x of the edges incident to x. Let $e = \{x, y\}$ be an edge incident to x. If y is a leaf, then $\lambda_x(e) = 1$. Otherwise, let $y_1, \ldots, y_d$ be the $d \geq 1$ neighbors of y in T distinct from x. If $d = 1$, we define $\lambda_x(e) = \lambda_y(\{y, y_1\})$. If $d > 1$, then assume, w.l.o.g., that $\lambda_y(\{y, y_i\}) \geq \lambda_y(\{y, y_{i+1}\})$ for every i, $1 \leq i < d$. We then define

$$\lambda_x(e) = \max\{\lambda_y(\{y, y_1\}),\ \lambda_y(\{y, y_2\}) + 1\}.$$

Note that every edge $e = \{x, y\}$ is assigned two labels: $\lambda_x(e)$ and $\lambda_y(e)$. The following result is straightforward.

Claim 8. *All edges can be labeled in $O(n)$ time.*

The following result establishes the relationship between the labels and the connected search numbers.

Claim 9. *For every edge $e = \{x, y\}$ of T, we have $\lambda_x(e) = \mathbf{cs}^+_y(T_x[y])$.*

Proof. The proof is by induction on the height $h(y)$ of $T_x[y]$, i.e., on the length of the longest simple path from y to the leaves of $T_x[y]$. The lemma holds for $h(y) = 0$, i.e., when y is a leaf, since then $\lambda_x(e) = \mathbf{cs}^+_y(T_x[y]) = 1$. Let us assume that the lemma holds whenever

$0 \leq h(y) < k$ for $k > 0$, and let us consider the case when $h(y) = k$. Let $y_1, y_2, \ldots, y_d$ be the $d \geq 1$ children of y in $T_x[y]$, where, w.l.o.g., $\lambda_y(\{y, y_i\}) \geq \lambda_y(\{y, y_{i+1}\})$. By definition of λ_x, if $d > 1$ then

$$\lambda_x(\{x, y\}) = \max\{\lambda_y(\{y, y_1\}), \lambda_y(\{y, y_2\}) + 1\}.$$

For every i, $1 \leq i \leq d$, the height of $T_y[y_i]$ is $h(y_i) < k$. Thus, by induction hypothesis,

$$\lambda_y(\{y, y_i\}) = \max\{\mathbf{cs}^+_{y_1}(T_x[y_1]),\ \mathbf{cs}^+_{y_2}(T_x[y_2]) + 1\}.$$

Similarly, for $d = 1$, we get $\lambda_y(\{y, y_i\}) = \mathbf{cs}^+_{y_1}(T_x[y_1])$. The claim thus follows by Lemma 6. $\diamond$

Consider now the following labeling μ of the vertices of T, which assigns to each vertex x a label $\mu(x)$ as follows. Let $e_1, e_2, \ldots, e_d$ be the d edges incident to x in T. If $d = 0$ then $\mu(x) = 1$. If $d = 1$ then $\mu(x) = \lambda_x(e_1)$. If $d \geq 2$ then assume, w.l.o.g., that $\lambda_x(e_i) \geq \lambda_x(e_{i+1})$ for each i, $1 \leq i < d$. Set

$$\mu(x) = \max\{\lambda_x(e_1), \lambda_x(e_2) + 1\}.$$

Claim 10. *For every vertex x of tree T, $\mu(x) = \mathbf{cs}^+_x(T)$.*

Proof. To prove the claim, add a virtual vertex x' to T and connect x' to x, resulting in a tree T'. Let λ' be the edge-labeling in T'. By definition, we have $\mu(x) = \lambda'_{x'}(\{x', x\})$. In view of Claim 9, we get $\mu(x) = \mathbf{cs}^+_x(T_{x'}[x])$. It follows that $\mu(x) = \mathbf{cs}^+_x(T)$. $\diamond$

It follows from Claim 8 that all $\mu(x)$'s can be computed in $O(n)$ time. Therefore, the connected search number of every tree T can be compute in linear time. Indeed, unless T is reduced to a single vertex, we have $\mathbf{cs}_x(T) = \mathbf{cs}^+_x(T)$ for every vertex x of T. We now show that an optimal monotone connected search strategy can also be computed in linear time.

Given a tree T and the labelings $\{\lambda_x, x \in V(T)\}$ and μ, consider the connected search strategy S constructed as follows. Let x be such that $\mu(x) = \min_y \mu(y)$. For each vertex y, order locally its incident edges in T according to the labels assigned by λ_y listed in increasing order. A monotone connected search strategy of T_x is obtained by starting with $\mu(x)$ searchers in x, and performing a tour in T_x according to a depth-first search (DFS) traversal respecting the local ordering of the edges. That is, at vertex y, the edges e incident to y with smallest labels $\lambda_y(e)$ are visited first. During this tour, the searchers are moved according to two simple rules:

- when moving from a vertex y to one of its children, z, slide $\lambda_y(\{y, z\})$ searchers along $\{y, z\}$ from y to z;

- when returning from z to y, slide these $\lambda_y(\{y,z\})$ searchers from z back to y along $\{y,z\}$.

The following claim follows from a simple induction on the depth of the tree once we have observed that when sliding $\lambda_y(\{y,z\})$ searchers along $\{y,z\}$ from y to z, we actually move $\mathbf{cs}_z^+(T_x[z])$ searchers to z (cf. Claim 9). This number of searchers is sufficient to clear $T_x[z]$ from z in a connected monotone way.

Claim 11. *The above search strategy is monotone, connected, and uses the optimal number of searchers.*

Since the DFS traversal can be performed in $O(n)$ time in n-vertex trees, this completes the proof of Theorem 3. (Note that the number of actions performed by the computed search strategy is $O(n \cdot \mathbf{cs}(T))$ because searchers are actually moved one by one in the search strategy; nevertheless the computed strategy can be coded compactly in $O(n)$ time and space). □

Note that, as opposed to what was claimed in [35], Theorem 3 does not trivially extend to the weighted version of the problem for which clearing an edge may require more than one searcher, and guarding a node may also require more than one searcher (see, e.g., [52], page 111 for a counterexample). In fact, it is known [53] that weighted versions of the pathwidth problem are NP-hard even for the case of trees.

5. Concluding Remarks and Open Problems

In this paper we initiated the study of the complexity of the main variant of the graph searching problem under the connectivity demand. To conclude, we first discuss an important byproduct of the design of our algorithm for trees.

Horton-Strahler number. The property satisfied by the connected search number, as stated in Lemma 6, is precisely the one defining the so-called *Horton-Strahler number*, originally proposed in hydrology by Horton [26] and Strahler [27, 28]. The Horton-Strahler number was initially defined as a measure of the propensity of a river to flood. It later appeared in many different contexts, including register allocation [30], mathematical biology [31, 32] (concerning the study of bifurcations in natural trees and in the respiratory system), and, recently, social networks [33]. The design of our linear-time algorithm for computing the connected search number $\mathbf{cs}(T)$ of a tree T is actually based on the fact that the "rooted"

connected search number $\mathbf{cs}_x(T)$ is precisely the Horton-Strahler number of a river with shape T, and whose mouth is vertex x.

Interestingly enough, the edge-labeling λ and the vertex-labeling μ used in the design of our algorithm shows that the Horton-Strahler number of a river T depends on the shape of the river basin modeled by T, but not much on the position of the mouth. (Indeed, it is easy to check that the labeling μ satisfies $|\mu(x) - \mu(y)| \leq 1$ for every two vertices x and y.) This observation holds for all hierarchical structures modeled by trees, such as the ones modeling arithmetical evaluations, or the ones modeling the hierarchical structure of social networks. Although all these trees are inherently rooted, only the structure of the tree actually matters, while the position of the root has almost no impact on the Horton-Strahler number.

Perhaps more interesting, the notion of connected search number may serve as a generalization of the Horton-Strahler number to arbitrary structures (i.e., graphs), beyond the simple case of trees. Determining what could be the interpretation of such a generalization in the framework of complex hydrological systems like swamps is an intriguing question (although is beyond the competences of the authors). Moreover, it would be of high interest to compute and compare the connected search numbers of various types of social networks. For instance, a small connected search number may indicate that information can be spread from one source to everyone while simultaneously using only few people as transmitters at each step. Conversely, a large connected search number may rather indicate that a rumor cannot spread in the network if less than a certain number of people are simultaneously acting as propagators.

Open problems. We conclude with a several open problems.

- Is it true that for any connected graph G, $\mathbf{cs}(G)/\mathbf{s}(G) \leq 2$?
- It is not hard to show that deciding if $\mathbf{cs}(G) \leq k$ for some k is NP-hard. We do not know if the problem belongs to NP and this is another open question. Let us remark, that by the result of Yang et al. [19], an optimal search strategy can use recontamination.
- The graph searching problem is fixed parameter tractable with a standard parameterization by the number of searchers. For connected search problem we even do not know if $\mathbf{cs}(G) \leq k$ can be decided in polynomial time when k is not part of the input. Can it be that the problem is NP-hard already for small values of k like $k = 3$ or 4?

Final remarks. While this paper was under review, Dariusz Dereniowski solved the first open question stated in this paper, by proving that, for any connected graph G, $\mathbf{cs}(G)/\mathbf{s}(G) \leq 2$ [54]. Moreover, the same author also proved that the weighted version of the problem is strongly NP-hard, even in the case of trees [55].

Acknowledgments. Many thanks to Binh Minh Bui Xuan for attracting our attention to the Horton-Strahler number of trees.

References

[1] S. Hoory, N. Linial, A. Wigderson, Expander graphs and their applications, Bull. Amer. Math. Soc. (N.S.) 43 (2006) 439–561 (electronic).

[2] C. Gotsman, On the optimality of valence-based connectivity coding, Comput. Graph. Forum 22 (2003) 99–102.

[3] M. Isenburg, P. Lindstrom, S. Gumhold, J. Snoeyink, Large mesh simplification using processing sequences, in: Proc. of the 14th IEEE Visualization (VIS 2003), IEEE Computer Society, Washington, DC, USA, 2003, p. 61.

[4] N. R. Jennings, An agent-based approach for building complex software systems, Commun. ACM 44 (2001) 35–41.

[5] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, C. H. Papadimitriou, The complexity of searching a graph, J. Assoc. Comput. Mach. 35 (1988) 18–44.

[6] F. V. Fomin, D. M. Thilikos, An annotated bibliography on guaranteed graph searching, Theor. Comput. Sci. 399 (2008) 236–245.

[7] R. H. Möhring, Graph problems related to gate matrix layout and PLA folding, in: Computational graph theory, volume 7 of *Comput. Suppl.*, Springer, Vienna, 1990, pp. 17–51.

[8] L. M. Kirousis, C. H. Papadimitriou, Searching and pebbling, Theoret. Comput. Sci. 47 (1986) 205–218.

[9] M. Franklin, Z. Galil, M. Yung, Eavesdropping games: a graph-theoretic approach to privacy in distributed systems, J. Assoc. Comput. Mach. 47 (2000) 225–243.

[10] G. Gottlob, Z. Miklós, T. Schwentick, Generalized hypertree decompositions: NP-hardness and tractable variants, J. Assoc. Comput. Mach. 56 (2009) 1–32.

[11] T. D. Parsons, Pursuit-evasion in a graph, in: Proc. Internat. Conf., Western Mich. Univ., Kalamazoo, Mich., 1976, Theory and applications of graphs, Springer, Berlin, 1978, pp. 426–441. Lecture Notes in Math., Vol. 642.

[12] T. D. Parsons, The search number of a connected graph, in: Proc. of the Ninth Southeastern Conference on Combinatorics, Graph Theory, and Computing, Florida Atlantic Univ., Boca Raton, Fla., Congress. Numer., XXI, Utilitas Math., Winnipeg, Man., 1978, pp. 549–554.

[13] N. N. Petrov, A problem of pursuit in the absence of information on the pursued, Differentsial'nye Uravneniya 18 (1982) 1345–1352, 1468.

[14] L. M. Kirousis, C. H. Papadimitriou, Interval graphs and searching, Discrete Math. 55 (1985) 181–184.

[15] N. Robertson, P. D. Seymour, Graph minors. XX. Wagner's conjecture, J. Combin. Theory Ser. B 92 (2004) 325–357.

[16] D. Bienstock, P. Seymour, Monotonicity in graph searching, J. Algorithms 12 (1991) 239–245.

[17] A. S. LaPaugh, Recontamination does not help to search a graph, J. Assoc. Comput. Mach. 40 (1993) 224–245.

[18] F. V. Fomin, D. M. Thilikos, On the monotonicity of games generated by symmetric submodular functions, Discrete Appl. Math. 131 (2003) 323–335.

[19] B. Yang, D. Dyer, B. Alspach, Sweeping graphs with large clique number, Disc. Math. 309 (2009) 5770–5780.

[20] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, T. Kloks, Approximating treewidth, pathwidth, frontsize, and shortest elimination tree, J. Algorithms 18 (1995) 238–255.

[21] F. V. Fomin, P. Fraigniaud, N. Nisse, Nondeterministic graph searching: from pathwidth to treewidth, Algorithmica 53 (2009) 358–373.

[22] H. L. Bodlaender, D. M. Thilikos, Computing small search numbers in linear time, in: Proc. First International Workshop on Parameterized and Exact Computation, (IWPEC 2004), Bergen, Norway, pp. 37–48.

[23] M. R. Fellows, M. A. Langston, Nonconstructive tools for proving polynomial-time decidability, J. Assoc. Comput. Mach. 35 (1988) 727–739.

[24] N. Robertson, P. D. Seymour, Graph minors. XIII. The disjoint paths problem, J. Combin. Theory Ser. B 63 (1995) 65–110.

[25] N. Robertson, P. D. Seymour, Graph minors. X. Obstructions to tree-decomposition, J. Combin. Theory Ser. B 52 (1991) 153–190.

[26] R. E. Horton, Erosional development of streams and their drainage basins: hydrophysical approach to quantitative morphology, Geol. Soc. of America Bull. 56 (1945) 275–370.

[27] A. N. Strahler, Hypsometric (area-altitude) analysis of erosional topology, Geol. Soc. of America Bull. 63 (1952) 1117–1142.

[28] A. N. Strahler, Quantitative analysis of watershed geomorphology, Trans. of the Amer. Geoph. Un. 8 (1957) 913–920.

[29] A. Gleyzer, M. Denisyuk, A. Rimmer, Y. Salingar, A fast recursive GIS algorithm for computing Strahler stream order in braided and nonbraided networks, J. Amer. Water Resourc. Assoc. 40 (2004) 937–946.

[30] P. Flajolet, J.-C. Raoult, J. Vuillemin, The number of registers required for evaluating arithmetic expressions, Theoret. Comput. Sci. 9 (1979) 99–125.

[31] R. Borchert, N. A. Slade, Bifurcation ratios and the adaptive geometry of trees, Botanical Gazette 142 (1981) 394.

[32] K. Horsfield, Some mathematical properties of branching trees with application to the respiratory system, Bull. of Math. Biol. 38 (1976) 305–315.

[33] A. Arenas, L. Danon, A. Díaz-Guilera, P. M. Gleiser, R. Guimerá, Community analysis in social networks, The European Physical Journal B - Condensed Matter and Complex Systems 38 (2004) 373–380.

[34] R. Breisch, An intuitive approach to speleotopology, Southwestern Cavers (A publication of the Southwestern Region of the National Speleological Society) VI (1967) 72–78.

[35] L. Barrière, P. Flocchini, P. Fraigniaud, N. Santoro, Capture of an intruder by mobile agents, in: Proc. of the 14th annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2002), ACM Press, 2002, pp. 200–209.

[36] L. Barrière, P. Fraigniaud, N. Santoro, D. M. Thilikos, Searching is not jumping, in: Proc. of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2003), volume 2880 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 34–45.

[37] F. V. Fomin, D. M. Thilikos, I. Todinca, Connected graph searching in outerplanar graphs, Electronic Notes in Discrete Mathematics 22 (2005) 213–216. 7th International Colloquium on Graph Theory. Short communication.

[38] P. Fraigniaud, N. Nisse, Connected treewidth and connected graph searching, in: Proc. of the 7th Latin American Symposium on Theoretical Informatics (LATIN 2006), volume 3887 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 479–490.

[39] N. Nisse, Connected graph searching in chordal graphs, Discrete Appl. Math. 157 (2009) 2603–2610.

[40] R. J. Nowakowski, N. Zeh, Boundary-optimal triangulation flooding, Internat. J. Comput. Geom. Appl. 16 (2006) 271–290.

[41] P. Fraigniaud, N. Nisse, Monotony properties of connected visible graph searching, Information and Computation 206 (2008) 1383–1393.

[42] L. Blin, P. Fraigniaud, N. Nisse, S. Vial, Distributed chasing of network intruders, Theor. Comput. Sci. 399 (2008) 12–37.

[43] P. Flocchini, M. J. Huang, F. L. Luccio, Decontamination of chordal rings and tori using mobile agents, Int. J. of Found. of Comp. Sc. 18 (2007) 547–564.

[44] P. Flocchini, M. J. Huang, F. L. Luccio, Decontamination of hypercubes by mobile agents, Networks 52 (2008) 167–178.

[45] D. Ilcinkas, N. Nisse, D. Soguet, The cost of monotonicity in distributed graph searching, Distributed Computing 22 (2009) 117–127.

[46] N. Nisse, D. Soguet, Graph searching with advice, Theor. Comput. Sci. 410 (2009) 1307–1318.

[47] D. Bienstock, Graph searching, path-width, tree-width and related problems (a survey), in: Reliability of computer and communication networks (New Brunswick, NJ, 1989), volume 5 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, Amer. Math. Soc., Providence, RI, 1991, pp. 33–49.

[48] P. D. Seymour, R. Thomas, Call routing and the ratcatcher, Combinatorica 14 (1994) 217–241.

[49] J. Matoušek, On embedding trees into uniformly convex Banach spaces, Israel J. Math. 114 (1999) 221–237.

[50] N. Linial, A. Magen, M. E. Saks, Trees and Euclidean metrics, in: Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC 1998), Dallas, Texas, ACM, New York, 1999, pp. 169–175.

[51] A. Takahashi, S. Ueno, Y. Kajitani, Minimal acyclic forbidden minors for the family of graphs with bounded path-width, Disc. Math. 127 (1994) 293–304.

[52] F. Huc, Conception de Réseaux Dynamiques Tolérants aux Pannes, Ph.D. thesis, Nice Sophia Antipolis, Projet MASCOTTE, (CNRS - UNSA) INRIA, 2008.

[53] R. Mihai, I. Todinca, Pathwidth is NP-hard for weighted trees, Frontiers in Algorithmics (2009) 181–195.

[54] D. Dereniowski, From pathwidth to connected pathwidth, in: 28th International Symposium on Theoretical Aspects of Computer Science (STACS), LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011, pp. 416–427.

[55] D. Dereniowski, Connected searching of weighted trees, Theor. Comput. Sci. 412 (2011) 5700–5713.