Network Decontamination

Nicolas Nisse

CHAPTER 1

Network Decontamination *

Nicolas Nisse

Université Côte d'Azur, Inria, CNRS, I3S, France

Abstract. The Network Decontamination problem consists of coordinating a team of mobile agents in order to clean a contaminated network. The problem is actually equivalent to tracking and capturing an invisible and arbitrarily fast fugitive. This problem has natural applications in network security in computer science or in robotics for search or pursuit-evasion missions. Many different objectives have been studied: the main one being the minimization of the number of mobile agents necessary to clean a contaminated network.
Many environments (continuous or discrete) have also been considered. In this Chapter, we focus on networks modeled by graphs. In this context, the optimization problem that consists of minimizing the number of agents has a deep graph-theoretical interpretation. Network decontamination and, more precisely, *graph searching* models, provide nice algorithmic interpretations of fundamental concepts in the Graph Minors theory by Robertson and Seymour.
For all these reasons, graph searching variants have been widely studied since their introduction by Breish (1967) and mathematical formalizations by Parsons (1978) and Petrov (1982). This chapter consists of an overview of the algorithmic results on graph decontamination and graph searching.

Keywords: Graph Searching, Path- and Tree-Decompositions, (Distributed) Graph Algorithms, Computational Complexity.

1 Introduction

Network Decontamination is a problem in which a team of mobile agents, called *searchers*, aims at *clearing* the links and nodes of an *infected* network. Alternatively, it can be defined as a *pursuit-evasion* game between a malicious intruder, called the *fugitive*, and a team of searchers that must *capture* the fugitive.

Since its introduction by Breisch [Bre67], Parsons [Par78a] and Petrov [Pet82], this field has received a lot of attention due to its numerous applications in network security and distributed computing, in robotics and differential games, and in graph theory. Previous surveys on network decontamination have been proposed [Bie91, Als04, FS06, FT08, CHI11, Bre12]. Most of them mainly focus

* This work has been partially supported by ANR program "Investments for the Future" under reference ANR-11- LABX-0031-01, the Inria Associated Team AlDyNet.

either on a centralized setting or on a distributed setting. This chapter aims at presenting an up-to-date (as exhaustive as possible) overview on graph decontamination both in distributed and centralized settings.

Lost in a cave. In 1967, Breisch opened the field of network decontamination by asking the following question:

"A person is lost in a particular cave and is wandering aimlessly. Is there an efficient way for the rescue party to search for the lost person? What is the minimum number of searchers required to explore a cave so that it is impossible to miss finding the victim if it is in the cave?" [Bre67].

Parsons [Par78a] and Petrov [Pet82] independently formalized the problem in a continuous setting where the objective is, for a team of mobile agents, the *searchers*, to capture an invisible and arbitrarily fast *fugitive*, in an environment modeled by a continuous embedding of a graph G on a surface. In this model, both the fugitive and the searchers move simultaneously in a continuous way from a point of G (corresponding to a vertex or in the interior of an edge) to another. The searchers *capture* the fugitive if, at some time, the fugitive occupies the same point as a searcher. We recall this technical definition for completeness.

Definition 1. *[Par78a] For any $k \in \mathbb{N}^*$, let $\mathcal{C}_k(G)$ be the set of families $F = \{s_1, \cdots, s_k\}$ such that, for every $1 \leq i \leq k$, $s_i : [0, \infty[\to G$ is a continuous function. A* search plan *for G is a family $F \in \mathcal{C}_k(G)$ such that, for every continuous function $f : [0, \infty[\to G$, there exists $t_f \in [0, \infty[$ and $i \in \{1, \cdots, k\}$ such that $s_i(t_f) = f(t_f)$.*

Intuitively, k represents the number of searchers. A search plan of $\mathcal{C}_k(G)$ is therefore a set of trajectories determined for each searcher (s_i represents the trajectory of searcher i), which ensures that, whatever be the trajectory f of the fugitive in G, there is a searcher that will occupy the same point as the fugitive at some time t_f. In other words, a search plan ensures that whatever be the strategy used by the fugitive, it must eventually be captured. Note that this definition does not constrain the speeds of the searchers and of the fugitive. Note also that, this model of pursuit-evasion game is actually equivalent to the *network decontamination* problem where a team of searchers must clear an infected network (e.g., a system of tunnels contaminated by some toxic gas, a computer network infected by a virus, *etc.*).

The continuous model of Parsons and Petrov can equivalently be defined in a discrete setting where environments are modeled by graphs [Par78b, Gol89a, Gol89b]. This latter formulation (formal definitions and examples are postponed to Section 2.1) is often referred to as *Graph Searching*[1] in the literature. Besides its natural applications in robotics or network security, one of the reasons for the vast literature on graph searching is probably its close relationship with some of the cornerstones of the Graph Minors theory [RS83, RS04]. Precisely, graph

[1] We should emphasize that there is another different topic of graph theory, related to Depth/Breadth First Search, called *Graph Searching*, a.k.a. Graph Traversals (e.g. [CDH$^+$16]).

searching provides an algorithmic interpretation of *tree- and path-decompositions* of graphs [RS90] that are important (algorithmic) tools of modern graph theory (see [CM93, DH08, BFL$^+$09, FLS18]). This relationship led to numerous results common to graph searching and graph decompositions (see Sections 2 and 4.1).

Pursuit-evasion games. Before starting our survey on graph searching, let us briefly mention different approaches for studying pursuit-evasion games. Roughly, the field may be divided into two main branches: pursuit-evasion games in continuous environments (polygonal environments, polyhedral surfaces, *etc.*) or in graphs. For the former approach, in a continuous setting, the reader is invited to see [GLL$^+$99, CHI11, BKIS12, KS15, ABC$^+$15] and references therein. In the case of discrete environments (i.e., in graphs), the field of pursuit-evasion games may also be divided into (at least) two different families of problems where results and tools are very different: *Cops and robber games* (see, e.g., [BN11] and Chapter 1 of [Nis14]) and Graph Searching. The main differences between these two approaches are (1) the different speeds of the fugitive, and (2) that Cops and Robber games are played turn-by-turn by two players while, in graph searching, the searchers and the fugitive move simultaneously. Roughly, both Graph Searching and Cops and Robber games are related to graph structural properties, but Cops and Robber games also rely on graph metric properties.

In this Chapter, we focus on graph environments, with searchers and an arbitrarily fast fugitive moving simultaneously, i.e., we speak about Graph Searching.

Organization of the chapter. The main graph searching variants are formally defined in Section 2, where relationships with graph decompositions and algorithmic results are presented. In Section 3, we focus on the *connected* variant of graph searching and on distributed algorithms for graph decontamination. Finally, Section 4 is devoted to the study of several alternative graph searching models. In this paper, the network decontamination terminology will be mainly used in Sections 2 and 3, and the pursuit-evasion terminology is used in Section 4.

We assume that the reader is familiar with graph terminology (see [Die12]). In particular, see [BLS99] for the definitions of the graph classes mentioned throughout the chapter.

2 Graph Searching

This section is devoted to the presentation of the basics of graph searching. We focus on computational complexity, algorithms in a centralized setting, and on the relationship between variants of graph searching and graph parameters and decompositions.

2.1 The seminal model: Edge-search

Graph searching aims at clearing the vertices and edges of a graph using a team of mobile agents, called *searchers*. Let us now formally define the seminal variant of graph searching, *a.k.a. edge-searching* [Par78b].

Network decontamination terminology. Initially all vertices and edges of a graph[2] $G = (V, E)$ are *contaminated*. A vertex is *cleared* when it is occupied by a searcher (in particular, initially, no vertices are occupied). An edge $e \in E$ is *cleared* if a searcher slides along e. Once a vertex/edge has been cleared, it is said *clear*. However, an unoccupied clear vertex is *recontaminated* as soon as there is a path free of searchers from it to a contaminated vertex. Similarly, an edge is recontaminated as soon as one of its endpoints is recontaminated.

A *strategy* consists of a finite sequence of *steps*, or *moves*, where each step consists of either sliding a searcher along an edge, or placing a searcher at some vertex of the graph, or removing a searcher from a vertex of G. The number of searchers *used* by a strategy is the maximum number of searchers present in the graph among all its steps. A strategy is *winning* if, eventually, it results in a state where all vertices and edges are (simultaneously) clear.

Simple examples. As a warm-up, let us consider the following examples.

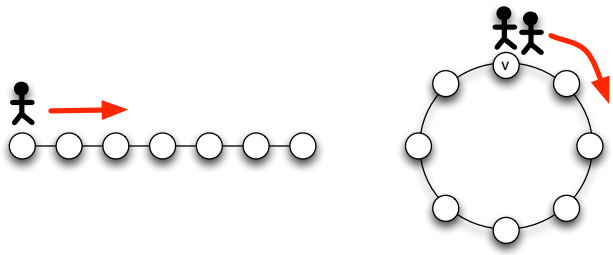

Fig. 1: Schematic overviews of optimal edge-search strategies in paths and cycles.

Paths. Let P_n be an n-node path. A strategy in P_n consists of, first, placing a searcher at one end of P_n and, then, sequentially sliding this searcher along every edge until it reaches the other end of P_n. It is easy to see that, when the searcher reaches the second end, every node and edge of P_n have been cleared and have never been recontaminated. Hence, such a strategy is winning.

Cycles. As a second example, let us consider the cycle C_n on $n \geq 3$ vertices. The first step of any strategy can only consist of placing a searcher at some vertex $v \in V(C_n)$ which becomes clear. Now, removing this searcher from v would result in the recontamination of v by its neighbors (and so it would result in the initial state). On the other hand, sliding the searcher along an edge from v to one of its neighbors u would result in a symmetrical state where only u is clear and occupied (since v and the edge uv would be recontaminated by the other neighbor of v). Hence, the only meaningful move is to place a second searcher at v (actually we may imagine placing this second searcher at other vertices but it would lead to other recontaminations) and slide it "around" the cycle until it comes back to v. During every sliding step, the

[2] Unless stated otherwise, all graphs considered in this chapter are simple, undirected, and connected.

searcher at v has *guarded* the vertex v, preventing the recontamination of all edges traversed by the second searcher. Therefore, the presented strategy using 2 searchers is winning.

Universal strategy. The last example is a *universal* strategy, i.e., which is winning in any (connected) graph $G = (V, E)$ with n vertices. During the n first steps, let us place one searcher at every vertex of G. Let $v \in V$ be any vertex of G and consider the searcher A at v. Sequentially, let us slide this searcher A along the edges of G until all edges of G have been traversed at least once. At this step, all edges not incident to v have both their ends occupied and have been cleared by A, therefore they are all clear. Finally, for every neighbor $u \in N(v)$[3] of v, let us slide the searcher at u along the edge uv from u to v. Clearly, the presented strategy is winning and uses n searchers.

Search number. As illustrated by the above examples, in any n-node graph, there exists a winning strategy using n searchers. On the other hand, a single searcher may not be sufficient to ensure the existence of a winning strategy (as shown in any cycle). Therefore, a natural optimization problem is to determine what is the minimum number of searchers required to clear a given graph. Precisely, the *search number* of a graph G, denoted by $\mathtt{s}(G)$, is the minimum integer $k \geq 1$ such that there is a winning search strategy for G using k searchers [Par78b].

Most of the work on graph searching has been dedicated to compute the search number of graphs and to design *optimal* strategies (i.e., winning strategies using the minimum number of searchers), both in centralized and distributed settings. However, several other objectives (see Sections 3.2 and 4.4) have been considered such as minimizing the "cost" of a strategy, its "length", the number of moves of the searchers or the number of "rounds" of a strategy, *etc.*

Monotonicity. Before going further, let us define a crucial notion when dealing with search strategies. A search strategy is *monotone* if, when following it, no vertices nor edges are recontaminated. Said differently, in a monotone strategy, it is forbidden to remove a searcher from a vertex v if v has at least one incident contaminated edge and no other searcher is occupying v. Moreover, sliding a searcher from a vertex v to one of its neighbors $u \in N(v)$ is allowed only: if v is occupied by another searcher; or if all edges incident to v are already clear; or if $vu \in E$ is the only edge incident to v that is still contaminated.

One of the first challenges concerning Graph Searching has been to answer the following question: "does recontamination help?". In other words, does there always exist an optimal strategy that is monotone? This latter question was first asked (and conjectured to be true) by Megiddo *et al.* [MHG+88]. At a first glance, this question looks "intuitively obviously true" (why would it be useful to let vertices be recontaminated?) but it is actually not obvious at all

[3] Given a graph $G = (V, E)$ and $v \in V$, $N(v)$ denotes the set of neighbors of v, i.e., $N(v) = \{u \in V \mid uv \in E\}$.

and, moreover, we will see (Sections 3 and 4.2) that there are variants of graph searching where recontamination actually helps. The conjecture of Megiddo *et al.* has been first proved by LaPaugh [LaP93] and an elegant proof of it by Bienstock and Seymour [BS91] is sketched in the next sub-section.

Theorem 1. *[LaP93, BS91] "Recontamination does not help", i.e., in any graph G, there exists a winning monotone strategy using $\mathtt{s}(G)$ searchers.*

We refer to Theorem 1 by saying that the edge-search variant is *monotone*. To see the importance of this theorem, let us do the following remarks.

- First, there always exists a winning monotone strategy with a number of steps which is polynomial (actually linear) in the size of the graph (since each edge and vertex is cleared exactly once). Therefore, such a strategy constitutes a polynomial-size certificate for the search number, i.e., given a graph G and an integer $k \geq 1$ as inputs, the problem of deciding if $\mathtt{s}(G) \leq k$ is in NP. We are not aware of another method to prove this fact.
- Second, monotone strategies are much easier to imagine and design, and they are much easier to manipulate to prove lower bounds. For instance, the universal strategy presented above allows to show that $\mathtt{s}(K_n) \leq n$ where K_n is the complete graph with $n \geq 1$ vertices. For any $n > 3$, this result is tight, i.e., $\mathtt{s}(K_n) = n$ for every $n > 3$. The general technique to prove such a lower bound is to consider an optimal monotone strategy (which exists by Th. 1) and assume, for the purpose of contradiction, that it uses less than n searchers. Finally, it can be shown that because at most $n-1$ searchers are used, there must be a step with recontamination, leading to a contradiction.
- Last but not least, the monotonicity result allows to establish the equivalence between graph searching and other graph parameters such as the parameters related to graph decompositions that are the corner stone of the Graph Minor theory (see Sections 2.2 and 4.1).

2.2 Mixed/Node-search and Pathwidth

The edge-search model provides a natural way to describe the seminal problems of Breisch [Bre67] and Parsons [Par78a] and it is the main model studied in a distributed setting (see Sections 3.2 and 3.3). Variants "close" to edge-search have been proposed because they are somehow easier to work with and, moreover, provide alternative definitions for graph parameters known in other contexts.

Node-search. Kirousis and Papadimitriou defined *node graph searching* because of its relationship with *pebble games* [KP85, KP86, Bie91]. In this setting, a strategy is defined as a sequence of moves like in edge-searching with two main differences. First, only two moves are allowed: placement/removal of a searcher at/from a vertex (so searchers do not slide along edges). Second, an edge becomes clear as soon as both its ends are occupied. In this variant, recontamination and monotone strategies are defined as in edge-search. The corresponding graph invariant is the *node search number*, denoted by $\mathtt{ns}$. For any graph G,

$\mathtt{ns}(G)-1 \leq \mathtt{s}(G) \leq \mathtt{ns}(G)+1$ [KP86]. Moreover, the three cases are possible since $\mathtt{s}(P_n) = 1 < \mathtt{ns}(P_n) = 2$ (where P_n is a path on n nodes), $\mathtt{s}(G) = \mathtt{ns}(G) = 2$ if G is a star with at least three leaves and $\mathtt{s}(K_{3,3}) = 5 > \mathtt{ns}(K_{3,3}) = 4$ [KP86]. Simple (and polynomial-time) transformations allow to "transpose" node-search to edge-search and *vice versa*. Indeed, Kirousis and Papadimitriou proved that, for any graph G, $\mathtt{s}(G^{//}) = \mathtt{ns}(G) + 1$ and that $\mathtt{s}(G) = \mathtt{ns}(G^{++}) - 1$ where $G^{//}$ (resp., G^{++}) is obtained from G by replacing each edge by two parallel edges (resp., by three edges in series) [KP86]. As we will see below, the node-search variant is important because monotone node-strategies provide an algorithmic interpretation of path-decompositions of graphs.

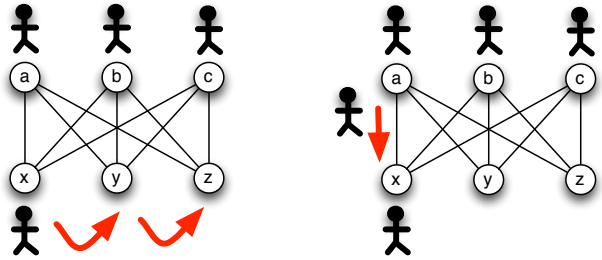

Fig. 2: Schematic overviews of optimal edge-search and node-search strategies in $K_{3,3}$. In both cases, one searcher remains at each of the vertices in $\{a, b, c\}$. In node-search (left), the fourth searcher goes sequentially to x, y and then z. In edge-search (right), a fourth search first goes to x while the fifth searcher sequentially clears the edges incident to x, then the fourth searcher goes to y and the fifth searcher clears the edges incident to it, and so on.

Mixed-search. To prove Theorem 1, Bienstock and Seymour defined the notion of *mixed-search strategy* [BS91] as an edge-search strategy with the difference that an edge is cleared either when it is crossed by a searcher or when both its ends are occupied by a searcher. The corresponding graph invariant is the *mixed search number*, denoted by $\mathtt{mixs}$. Again, there is a close relationship with edge-search. Precisely, for any graph G, $\mathtt{mixs}(G) \leq \mathtt{ns}(G) \leq \mathtt{mixs}(G)+1$ [BS91] and inequalities are tight. Moreover, $\mathtt{mixs}(G^+) = \mathtt{s}(G)$ for any graph G where G^+ is obtained from G by subdividing each edge once [BS91].

As mentioned above, mixed-searching has been introduced because it allows an elegant proof of Theorem 1. We aim at sketching this (a bit technical) proof because it is instructive since many studies on graph searching use a similar formalism, representing search-strategies by a sequence of tuples of sets of edges or vertices.

Sketch of proof of Theorem 1.[BS91] A *crusade* in a graph $G = (V, E)$ is a sequence $(E_0, \cdots, E_\ell)$ of subsets of E such that $E_0 = \emptyset$, $E_\ell = E$ and $|E_i - E_{i-1}| \leq 1$ for every $1 \leq i \leq \ell$. The crusade uses k searchers if $|\delta(E_i)| \leq k$ for every $0 \leq i \leq \ell$, where $\delta(E_i)$ is the set of vertices incident with an edge in E_i and an edge in $E \setminus E_i$. A crusade is *progressive* if $E_{i-1} \subseteq E_i$ for every $1 \leq i \leq \ell$.

The proof consists of first easily showing that, if there is a mixed strategy using k searchers, then there is a crusade using at most k searchers. The second easy step of the proof is to show that, if there is a progressive crusade using k searchers, then there is a monotone mixed strategy using at most k searchers. The key of the proof is to prove that if there is a crusade using k searchers then there exists a progressive crusade using at most k searchers. The latter step is proved by considering a crusade $\mathcal{C}$ (using k searchers) such that $\sum_{0\leq i\leq \ell}(|\delta(E_i)|+1)$ is minimum and, under the previous assumption, $\sum_{0\leq i\leq \ell}|E_i|$ is minimum. Then, using the submodularity of the function δ (i.e., for every $A, B \subseteq E$, $|\delta(A\cup B)|+|\delta(A\cap B)|\leq|\delta(A)|+|\delta(B)|$), it can be shown that $\mathcal{C}$ is progressive. $\diamond$

This proves that mixed-searching is monotone. Noticing that the simple transformations presented above preserve monotonicity, this implies that both node-search and edge-search are monotone too.

Additionally, the monotonicity of mixed search allows to prove that, for any graph G, $\texttt{mixs}(G)-1$ actually equals the *proper pathwidth* of G [TUK95].

Path-decomposition and Pathwidth. *Pathwidth* is an important structural measure that appears in the Graph Minor theory [RS83, Bie91, Bod98] but also in other domains such as VLSI design [DKL87, Kin92, FL94]. Given a graph $G=(V,E)$, a *path-decomposition* is a sequence $P=(X_0,\cdots,X_\ell)$ of subsets of vertices of G, called *bags*, such that (1) $\bigcup_{0\leq i\leq \ell}X_i=V$; (2) for every $uv\in E$, there exists $0\leq i\leq \ell$ with $\{u,v\}\subseteq X_i$; and (3) for every $0\leq i\leq j\leq k\leq \ell$, $X_i\cap X_k\subseteq X_j$. The *width* of P is the size of its largest bag minus one, and the *pathwidth* of G, denoted by $\texttt{pw}(G)$, is the minimum width of a path-decomposition of G (see an example on Fig. 3).

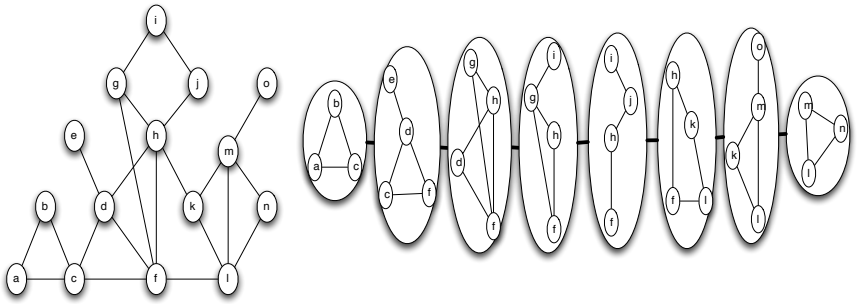

Fig. 3: Example of a graph G (left) and one of its path-decompositions $P=(X_0,\cdots,X_7)$ of width 3 (right). Prove that it is optimal, i.e., that $\texttt{pw}(G)=3$. A (monotone winning) node-search strategy corresponding to P is then $S=(a,b,c,\bar{a},\bar{b},d,e,f,\bar{c},\bar{e},h,g,\bar{d},i,\bar{g},j,\bar{i},\bar{j},k,l,\bar{f},\bar{h},m,o,\bar{k},\bar{o},n)$, where x means to place a searcher at vertex x, while $\bar{x}$ means to remove the searcher at x.

From any path-decomposition $P = (X_0, \cdots, X_\ell)$ of a graph G, it is easy to derive a node-search strategy for G: for i from 0 to ℓ, sequentially place a searcher at every vertex of X_i and then sequentially remove the searchers from the vertices in $X_i \setminus X_{i-1}$ (see an example on Fig. 3). From the properties of path-decompositions, for every $0 \leq i < \ell$, $S = X_i \cap X_{i+1}$ *separates* $A = \bigcup_{0 \leq j \leq i} X_j \setminus X_{i+1}$ from $B = \bigcup_{i < j \leq \ell} X_j \setminus X_i$ [Bod98] and therefore, the searchers at S prevent B from recontaminating A. Hence, it is easy to see that such a strategy is winning and monotone and that the number of searchers used equals the width of P plus one. Reciprocally, from any monotone winning strategy using k searchers, it is easy to derive a path-decomposition of width $k-1$ (where each bag corresponds to the set of vertices occupied by a searcher at each step). Therefore, by Theorem 1 (applied to node-search):

Theorem 2. *[KP85, KP86, EST94] For any graph G, $\mathtt{ns}(G) = \mathtt{pw}(G) + 1$.*

Among other important consequences, Theorem 2 allows to transpose the numerous results concerning pathwidth to node-search and, sometimes, to edge-search and mixed-search by using the simple transformations seen above.

Note that pathwidth, and so node-search number, may be equivalently defined in terms of a measure, called *vertex-separation*, of linear layouts of vertices [Kin92]. Similarly, the mixed-search number of a graph G can be defined in terms of *linear width* (some measure on the linear layouts of the edges) [Thi00].

2.3 Complexity and algorithms.

This subsection is devoted to the computational complexity of the edge-, node- and mixed graph searching problems. Algorithms to compute the search numbers (and corresponding strategies) in general graphs and particular graph classes are also presented.

Hardness. Given an n-node graph G and an integer $k \geq 1$ as inputs, the problem of deciding if $\mathtt{s}(G) \leq k$ has first been proved to be NP-hard in [MHG+88]. Then, Monien and Sudborough proved that this problem is NP-hard in the class of planar graphs with maximum degree 3 [MS88]. This latter result also holds for both node-search and mixed-search. Later on, Gustedt proved that deciding the pathwidth (and so the node-search number) is NP-hard in the class of chordal graphs [Gus93]. Edge-search is also NP-hard in chordal graphs [PTK+00]. Moreover, assuming the *Small Set Expansion Conjecture*, the problem of deciding the pathwidth of a graph is NP-hard to approximate within a constant factor [WAPL14].

Exact generic algorithms. On the positive side, all graph searching variants mentioned so far are *closed under taking minor*. That is, for any minor[4] H of a

[4] A minor of a graph G is any subgraph of any graph obtained from G by contracting some edges.

graph G, $\mathtt{s}(H) \leq \mathtt{s}(G)$ (resp., $\mathtt{ns}(H) \leq \mathtt{ns}(G)$ and $\mathtt{mixs}(H) \leq \mathtt{mixs}(G)$). Therefore, from the Graph Minor theory [RS04, Die12], it follows that, for any fixed $k \geq 1$, the set of *minimal obstructions* for having a search-number at most k is finite, and so each search-number admits a *Fixed Parameter Tractable* (FPT) algorithm [RS95, CFK+15] (where the parameter is the size of the solution). In the case of node-search, Ellis *et al.* first designed an algorithm in time $O(n^{2k^2+4k+8})$ via dynamic programming [EST87]. They also gave structural characterizations of graphs with node-search number at most 3 [EST94]. Then, Bodlaender and Kloks gave the first constructive FPT algorithm for deciding whether $\mathtt{ns}(G) \leq k$ in time $k^{O(k^3)}n$ [BK96]. In the case of mixed-search (and linear width), Bodlaender and Thilikos gave a constructive FPT algorithm [BT04]. Thilikos also designed a linear-time algorithm to decide whether a graph has mixed-search (resp., edge-search) number at most two by fully characterizing the set of minimal obstructions [Thi00]. In addition to parameterized algorithms, other algorithms have been proposed to compute the pathwidth of general graphs. The best known exact exponential-time algorithm computing the pathwidth runs in time $O(1.89^n)$ [KKK+16] (see also [SV09]). Moreover, using the definition(s) of graph searching in terms of vertex-layout, several Integer Linear Programs solving these problems have been proposed [PSS13, CMN16, Cou16, Mal18].

Graph classes. Search problems can be solved in various graph classes in polynomial time. The case of trees has been particularly studied [Par78a, MHG+88, EST94, PHH+00]. In particular, Skodinis designed a linear-time algorithm for computing the node-search number of trees and a corresponding strategy [Sko03]. A generic and distributed algorithm for computing, in time $O(n \log n)$, any of the search numbers in n-node trees (only the initial setting of the algorithm differ) has been designed in [CHM12], where the interesting notion of hierarchical decomposition of trees is introduced. The algorithms for trees are all based on the so-called Parsons' lemma. Since trees are particularly interesting in graph searching, we sketch its proof below. In the following, given a tree T, a vertex $v \in V(T)$ and a connected component T' of $T \setminus v$, let $T' \cup v$ denote the subtree induced by the vertices of T' and v.

Lemma 1 (Parsons' lemma [Par78b]). *For any $k \in \mathbb{N}^*$ and any tree T, $\mathtt{s}(T) \geq k+1$ if and only if T has a vertex v with at least three components T_1, T_2, T_3 of $T \setminus v$ such that $\mathtt{s}(T_i \cup v) \geq k$ for every $i \in \{1,2,3\}$.*

Sketch of proof. The "if" part follows from monotonicity. Indeed, assume there is a monotone search strategy using at most k searchers in T and let v, T_1, T_2, and T_3 be as defined in the lemma. By monotonicity, we may assume that $T_1 \cup v, T_2 \cup v$ and $T_3 \cup v$ are cleared in this order. However, to clear $T_2 \cup v$, there must be a step at which all k searchers are occupying vertices of T_2, which would imply a recontamination of $T_2 \cup v$ from T_3, a contradiction.

On the other hand, if there exists no vertex v as in the lemma, it is possible to find a subpath P, called an *avenue*, such that, for every connected component T_u of $T \setminus P$ (where u is the unique neighbor of T_u in P), $\mathtt{s}(T_u \cup u) < k$ (see

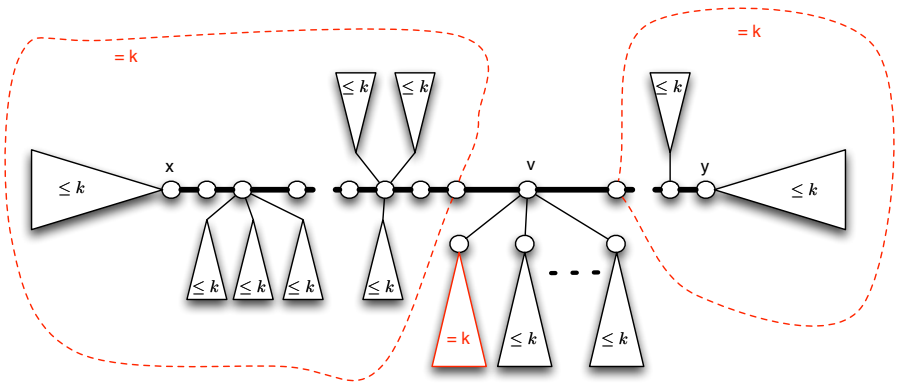

Fig. 4: Schematic overview of a tree T with $\mathtt{s}(T) = k+1$. A triangle labelled with "$\leq k$" represents a subtree with search number at most k. The node v has at least three components (in red) with search number k. The bold path (between x and y) is an avenue in T.

Fig. 4). Then, a strategy using k searchers consists of sliding one searcher from one end of P to its other end, while sequentially clearing the components of $T \setminus P$ using the $k-1$ remaining searchers. The avenues can be recursively computed by a dynamic programming algorithm. ◇

The above strategy implies that $\mathtt{s}(T) = O(\log n)$ in any n-node tree T. Moreover, this bound is tight (consider a rooted tree where all internal vertices have degree 3 and all leaves are at the same distance from the root).

In contrast, we should mention that graph searching is NP-hard in weighted trees, where weights on vertices represent the number of searchers required to preserve a vertex from recontamination, and edge-weights represent the number of searchers that must simultaneously traverse an edge to clear it [MT09].

Many other graph classes have been studied. The pathwidth is polynomial-time computable in *circular arc graphs* (in time $O(n^2)$) [ST07], *unicyclic graphs* [EM04, YZC07], *biconvex bipartite graphs* [PY07], in some subclasses of *chordal graphs* [Gus93], in *hypercubes* [CK06], in *cographs* [BM93] (in linear time), *etc.* The pathwidth can also be computed in time $n^{O(1)}$ (in the proposed algorithm, the exponent is larger than 11) in *outerplanar graphs* (using the fact that these graphs have bounded treewidth) and 2-approximation algorithms (using the *dual* of outerplanar graphs) running in time $O(n \log n)$ are designed in [BF02, CHS07].

The other variants have also been studied. The mixed search number can be computed in linear time in *interval graphs*, in polynomial time in *split graphs* [FHM10], and in linear time in *permutation graphs* [HM08]. The edge search number can be computed in linear time in *cographs* [GHM12] and in polynomial time in split graphs and interval graphs [PTK$^+$00].

Open Questions. *We would like to conclude this section with some intriguing open questions. First, note that there are no known graph classes where the complexity of deciding the edge/node/mixed search number differs. Moreover, for any graph G and any distinct $x, y \in \{\mathtt{s}, \mathtt{ns}, \mathtt{mixs}\}$ as inputs, the complexity of computing $x(G) - y(G)$ is not known.*

3 Connected Graph Searching and Distributed Setting

In all models defined in Section 2, removing a searcher and placing it at any vertex is allowed. Such a *jump* may however be unrealistic or even impossible in practical applications. Removing a searcher from a vertex v and placing it at another vertex u may be replaced by a sequence of slides along the edges of a path from v to u. However, this would imply that the searcher travels in an unsafe environment (through a part that is still contaminated) and moreover, it may lead to strategies that are not monotone. To handle this problem, Barriere *et al.* proposed a new variant, called *connected graph searching*, where removing a searcher is not allowed and in which, at every step, the subgraph induced by the clear edges and vertices must be connected [BFFS02].

Two main questions were asked with the introduction of connected graph searching in [BFFS02, BFST03]. First, what is "the cost of connectivity" in terms of the number of searchers? That is, does there exist a constant c such that any graph G admits a connected search strategy using at most $c \cdot \mathtt{s}(G)$ searchers? Second, is connected graph searching monotone?

3.1 Cost of connectivity

A *connected search strategy* $\mathcal{S}$ in a graph $G = (V, E)$ and using $k \geq 1$ searchers can be defined as follows. First, a vertex $v_0 \in V$, called *homebase*, is chosen and all the k searchers are placed at it. Then, $\mathcal{S}$ is a sequence of moves, where each move consists of sliding a searcher at $u \in V$ along an edge $e = uv \in E$ and such a move is allowed only if, after the sliding, there is path of clear edges from v_0 to v, i.e., the clear part must always be connected (here we only consider the edge-search variant where an edge is cleared when a searcher slides along it). The *connected search number* of a graph G, denoted by $\mathtt{cs}(G)$, is the smallest k such that there exists a connected search strategy that clears G using k searchers. Clearly, the choice of the homebase has an impact on the number of searchers (e.g., consider a path where the homebase is not one of its ends). Hence, the connected search number is defined with respect to the "best" possible homebase.

(Non) Monotonicity. Connected strategies clearly allow recontamination. Monotone connected search strategies are defined in a similar way: first, a vertex $v_0 \in V$ is chosen and all the k searchers are placed at it, then, the strategy consists of a sequence of moves, where each move consists of sliding a searcher at $u \in V$ along an edge $e = uv \in E$ only if either u is still occupied by a searcher after the move, or all incident edges of u but possibly e were already clear before

the move. One important and surprising result is that, contrary to the classical graph searching, in the connected variant, recontamination may help [YDA09]. It is interesting to mention that their counter-example G has about 400.000 vertices and is such that $\texttt{cs}(G) = 281$ while any monotone connected strategy requires at least 290 searchers (see Fig. 5). We are not aware of a smaller example.

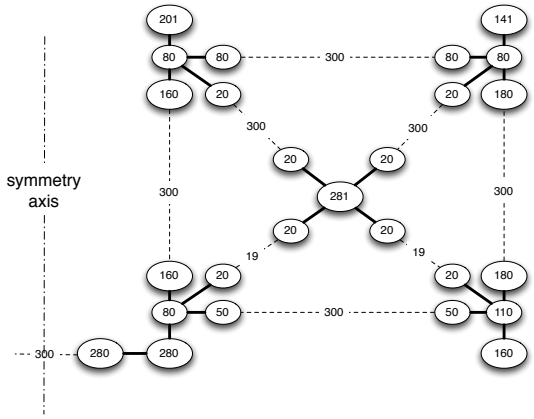

Fig. 5: Schematic overview of a graph G such that $\texttt{cs}(G) = 281 < \texttt{mcs}(G) = 290$ [YDA09]. Any circle with label x inside represents a clique of size x. A bold edge between two cliques A and B (with $|A| \leq |B|$) represents a perfect matching between the vertices of A and a subset of size $|A|$ of the vertices of B. A dotted line with label ℓ between two cliques of same size x represents a "path" of ℓ cliques of size x where two consecutive cliques are joint by a perfect matching.

Hence, the *monotone connected search number* of a graph G, denoted by $\texttt{mcs}(G)$, may be strictly larger than its connected search number $\texttt{cs}(G)$. A consequence of this result is that it is not known whether the problem of computing the connected search number of a graph is in NP. As far as we know, there are no lower or (non-trivial) upper bounds on the number of steps of connected search strategies. Another difference between the search number and its (monotone or not) connected counter part is that `mcs` and `cs` are not minor-closed. Hence, it is not clear whether the problem of computing the (monotone) connected search number of graphs admits a fixed parameter tractable algorithm (nor even a polynomial-time algorithm when the number k of searchers is fixed). However, both parameters are closed under taking contractions [BGTZ16].

Recontamination does not help for connected graph searching in trees, i.e., $\texttt{mcs}(T) = \texttt{cs}(T)$ in any tree T [BFFS02, BFF$^+$12] (proof *à la* Bienstock and Seymour). Besides, in any n-node tree T and for any homebase v_0, there exists a monotone connected strategy, starting from v_0 and using at most $1 + \texttt{cs}(T)$ searchers and, moreover, $\texttt{cs}(T) = O(\log n)$ [BFFS02]. In the same paper, it is

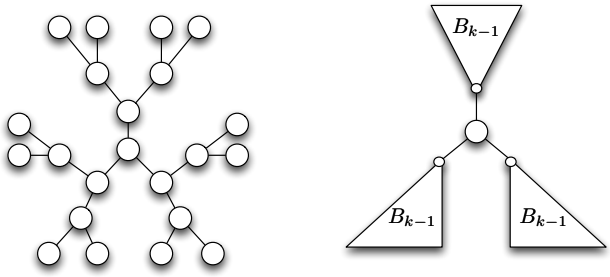

Fig. 6: On the left, the tree D_3 such that $\mathtt{s}(D_3) = 3 < \mathtt{cs}(D_3) = 4$. On the right, the recursive construction of the tree D_k where B_{k-1} is the complete binary tree of depth $k-1$ for any $k \geq 2$. For any tree T, $\mathtt{cs}(T) \geq k+1$ if and only if T admits D_k as a minor [BFST03, BFF$^+$12].

shown that computing the connected search number can be done in polynomial time in trees. Recently, it has been shown that recontamination does not help in the class of graphs with connected search number at most two [BGTZ16]. That is, for any graph G, $\mathtt{mcs}(G) = 2$ if and only if $\mathtt{cs}(G) = 2$. The result follows from the characterization of this class of graphs by exhibiting the family of 177 minimal-contraction obstructions.

Cost in number of searchers. Let us start with a simple example. Consider the complete rooted tree D_3 with all internal vertices with degree three and all leaves at distance 3 from the root. It is easy to see that $\mathtt{s}(D_3) = 3 < \mathtt{cs}(D_3) = 4$. Therefore, connectivity has some price in terms of minimum number of searchers. In any tree T, $\mathtt{cs}(T) \leq 2\mathtt{s}(T) - 2$ and this bound is tight [BFST03, BFF$^+$12]. The proof relies on the fact that $\mathtt{cs}(T)$ is closed under taking minors in the class of trees and that $\mathtt{cs}(T) \geq k$ if and only if T contains some specific tree D_k as a minor (in contrast with the classical search number, the set of minimal obstructions for connected search number in trees is reduced to a single tree).

Therefore, the question of the cost of connectivity arises naturally: how far is the connected search number of a graph from its pathwidth? In other words, does there exist a constant $c \geq 2$ bounding the ratio between connected search number and search number in any graph? Several partial results have been proposed [Nis09, BFF$^+$12] before Dereniowski closed the question:

Theorem 3. *[Der12b]* $\mathtt{mcs}(G) \leq 2\mathtt{s}(G) + 3$ *in any graph* G.

To prove Theorem 3, Dereniowski designed a polynomial time algorithm that transforms any monotone search strategy using k searchers into a connected one using at most $2k+3$ searchers. His result shows that the ratio between monotone connected search number and connected search number is bounded by 2.

Complexity and algorithms. On the complexity point of view, computing $\texttt{cs}$ is NP-hard since $\texttt{cs}(G^*) = \texttt{s}(G) + 1$ for any graph G where G^* is obtained from G by adding a universal vertex. Dereniowski proved that weighted connected graph searching is also NP-hard in weighted trees [Der11]. On the positive side, a polynomial time approximation with approximation ratio 3 is designed for this problem in trees [Der12a]. Similar results were proved while with different terminology (speaking about edge-width instead of weight) [BTK11]. The connected search number of outerplanar graphs has been investigated in [FTT05]. Very recently, Dereniowski *et al.* proved that the problem of deciding if $\texttt{cs}(G) \leq k$ can be solved in polynomial-time when k is fixed [DOR18].

Open Questions. *Is the problem of deciding (when k is part of the input) whether $\texttt{cs}(G) \leq k$ in* NP*? Is it FPT in k?*

Internal Graph Searching. To conclude this subsection, let us mention *internal graph searching* that can be defined as monotone connected graph searching but where there may be several homebases. That is, initially, one or more vertices are chosen and some searchers are placed at them. Then, the only allowed moves are to slide searchers if it does not create recontamination. This variant has been first introduced in [BFST03] and an interesting heuristic has been proposed in [FNS07]. In this paper, the initial vertices (the homebases) are chosen randomly and then the searchers grow the cleared part around these vertices in a BFS manner, then the best obtained strategies are used to generate next generations of strategies using a classical genetic algorithm.

3.2 Distributed (Monotone) Connected Graph Searching

A major reason for which the connectivity constraint has been introduced is that it ensures safe communications between the searchers during the execution of the strategy. For instance, when the searchers have to coordinate themselves but have no way to communicate when they are far from each other, possible solutions would be either to leave some messages on the vertices or to use a searcher for carrying instructions between other searchers. In both cases, the connectivity constraint helps since it allows to avoid that messages are left on contaminated vertices that the searcher crosses when moving in the contaminated area to transfer instructions.

In this subsection, we study the clearing of graphs in such environments where the searchers have only local vision of their environment and must communicate to coordinate the clearing.

Distributed model. The k searchers are modeled by synchronous autonomous mobile computing entities (automata) with distinct IDs from 1 to k. Otherwise searchers are all identical, run the same program, and use at most $O(\log n)$ bits of memory, where n is the number of vertices of the network. A network is modeled by an undirected connected graph G. A priori, the network is asynchronous. However, as explained below, any synchronous algorithm can be transposed into

an asynchronous environment by adding an extra searcher traveling in the (connected) clear part of the graph to synchronize the moves of all searchers. Moreover, the network is anonymous, that is, the vertices are not labelled. The edges incident to any vertex u are labelled from 1 to its degree, so that the searchers can distinguish the different edges incident to a vertex. Every vertex of the network has a zone of local memory, the *whiteboard* in which searchers can read, erase, and write symbols (unless stated otherwise, whiteboards have size $O(\log n)$ bits and are only used for face-to-face communication between searchers occupying a same vertex). It is moreover assumed that searchers can access these whiteboards in fair mutual exclusion. The goal is then to design an algorithm, called a *search protocol*, such that the fewest number of searchers running this algorithm achieves the clearing of the graph in a connected way.

Universal algorithms. In this section, we present several search protocols that have been designed to clear any graph $G = (V, E)$. In this setting, the searchers do not know in advance in which graph they are launched. That is, when occupying some vertex u, a searcher executes the algorithm only based on its current state (the *memory* of the searcher), on the content of the whiteboard at u, and on the degree of u. [BFNV08] designed a general algorithm allowing $\texttt{mcs}(G)+1$ searchers to connectedly clear any graph G. Since the extra searcher (compared to the centralized case) cannot be avoided due to the asynchronicity of the network, this is optimal. Roughly, this algorithm orders all possible sequences of moves in some well-suited lexicographical order and tries them one after the other (sequentially increasing the number of searchers that are used) until the graph is clear. For this purpose, the searchers use whiteboards of size $O(|E| \log |V|)$ bits where they write all their moves. At the end, a description of the strategy is then stored in a distributed way on the whiteboards. This algorithm has however two drawbacks: it takes an exponential amount of time (which cannot be avoided unless $P = NP$) and the clearing is not monotone.

To deal with monotonicity, [NS09] proposed to address the problem by providing a small amount of information (*advice*) to the searchers, following the framework of [FIP06]. Precisely, it is shown that the minimum number of bits of information that must *a priori* be distributed in an n-node graph G in order to clear it monotoneously with the optimal number of searchers is $\Theta(n \log n)$ [NS09]. Roughly, this piece of information encodes a spanning tree "along which" the clearing must be performed.

Another approach to handle monotonicity is to allow the use of more searchers. More precisely, the *cost* of a search protocol $\mathcal{P}$ in a graph G with homebase v_0 is measured by the ratio between the number of searchers it uses to clear G and the search number $\texttt{mcs}(G)$ of G. This ratio, maximized over all graphs and all starting vertices, is called the *competitive ratio* of the protocol $\mathcal{P}$. [INS09] proved that monotonicity has an important cost (i.e., may increase significantly the minimum number of searchers) in a distributed setting since any search protocol (clearing any graph in a monotone connected way) has competitive ratio $\Omega(\frac{n}{\log n})$ and that this lower bound holds in the class of trees with maximum degree 3. On the positive side, this bound is tight: there exists a search protocol

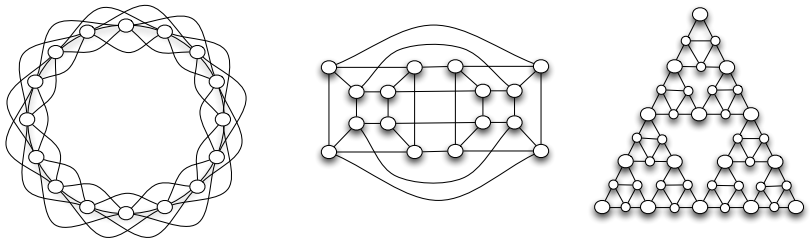

Fig. 7: Illustrations of a chordal ring (left), the hypercube of dimension 4 (center) and the Sierpiński graph built by 4 iterations (right).

with competitive ratio $O(\frac{n}{\log n})$ [INS09]. The idea behind the algorithm is to "control" a (partial) spanning tree of the clear part and to determine the next edge to be cleared according to it in such a way that this tree does not contain a "high" ternary tree as a minor (since such a minor would lead to the use of many searchers).

Specific topologies. Many distributed search protocols specialized for particular graph classes have also been designed. A distributed algorithm that computes the connected search number of trees has been proposed in [BFFS02, BFF$^+$12]. Then, a *self-stabilizing* algorithm for clearing trees has been designed [MM09] and further improved in [BMM10]. The latter algorithm allows $1+\log n$ searchers to clear any n-node tree and stabilizes in $O(n \log n)$ moves after initialization. Moreover, it is a *non-silent* algorithm, meaning that it continues to clear the tree indefinitely.

Topologies that are commonly used for interconnection networks (see examples on Fig 7) have been studied. Precisely, the following topologies have been considered: grids [FLS05], chordal rings and tori [FHL07], hypercubes [FHL08], and Sierpiński graphs [Luc09]. In this setting, the authors compare the number of searchers, moves, and the number of rounds of their algorithms in two models. In the first model, a particular searcher is used to coordinate the clearing while, in the latter one, the searchers are endowed with some visibility ability: they can see whether their neighbors are clear or contaminated, empty or occupied. All designed algorithms use the fact that all these graph classes admit relatively well-structured centralized strategies and, moreover, the symmetries of these topologies allow the searchers to benefit from some *sense of direction* (for instance, clockwise orientation in chordal rings or standard compass-labelling in grids). For instance, in a grid starting from one of its corners, the strategy first makes the searchers occupy all vertices of the first column and then move "in parallel" from one column to the next one until the grid is clear. In the case with visibility, the searchers can locally decide when they have to go to the next column without recontamination. Table 1 summarizes the obtained results (note

that these results consider the clearing of vertices only or, said differently, an edge is cleared in the same way as in node-search).

Open Questions. *The studies of the tradeoffs between the number of searchers, moves, and time steps are left as open problems.*

Topology	**Model**	**# searchers**	**# moves**	**# time steps**
$m \times n$ **Grids** [FLS05]	coordinator	$m+1^*$	$\frac{m^2+4mn-5m-2}{2}$	$mn-2$
$(m \leq n)$	visibility	m^*	$\frac{m^2+2mn-3m}{2}$	$m+n-2^*$
$m \times n$ **Tori** [FHL07]	coord.	$2m+1^*$	$2mn-4m-1$	$mn-2m$
$(m \leq n)$	vis.	$2m^*$	$mn-2m$	$n-2$
n-node **Chordal rings** [FHL07]	coord.	$2\ell+1^*$	$4n-6\ell-1$	$3n-4\ell-1$
with largest chord of length ℓ	vis.	$2\ell^*$	$n-2\ell$	$\lceil \frac{n-2\ell}{2(\ell-\ell')} \rceil$
and ℓ' second largest chord				
Hypercubes [FHL08]	coord.	$\Theta(\frac{n}{\sqrt{\log n}})^*$	$O(n \log n)$	$O(n \log n)$
(dimension n)	vis.	$n/2$	$O(n \log n)$	$O(\log n)^*$
Sierpiński graphs [Luc09]	coord.	$n+1^*$	$O(n3^n)$, $\Omega(3^n)$	$O(3^n)$, $\Omega(3^n/n)$
built by n iterations	vis.	$n+2$	–	–

Table 1: Monotone connected search in specific topologies. Results marked with a star $(*)$ are known to be optimal.

A search protocol has also been designed for *partial grids* (i.e., connected subgraphs of $n \times n$ grids) that uses $O(\sqrt{n})$ searchers [DU16]. The algorithm strongly uses sense of direction and the algorithm in [BDK15] as a subprocedure. The algorithm is optimal since some partial grids require this amount of searchers and moreover, the authors prove that, for any search protocol, there are partial grids (with search number $O(\log n)$) that force the algorithm to use $\Omega(\sqrt{n})$ searchers [DU16].

To conclude this subsection, let us mention the *cloning* variant proposed in [FHL08]. In this model, the searchers may clone themselves, i.e., the searchers are not restricted to appear at the homebase but, at any step, a searcher at v may create new searchers at v (this essentially allows to decrease the number of moves and time steps). Various topologies have been studied in this setting: hypercubes [FHL08], graph products [ISZ07], grids and tori [ISZ08], and pyramids [SIS06].

3.3 Exclusive Graph Searching and Look/Compute/Move

Exclusive graph searching is defined as mixed graph searching with the extra *exclusivity constraint* (each vertex can be occupied by at most one searcher at a time) and such that searchers cannot jump from one vertex to another one, i.e., searchers can only slide along edges [BBN17].

Exclusive graph searching addresses two limitations of classical variants as far as practical applications are concerned. First, as in internal graph searching, the unrealistic assumption that searchers may jump is got rid of. Second, classical variants assume that any vertex can be simultaneously occupied by several searchers. This assumption may be unrealistic in several contexts. Typically, placing several searchers at the same vertex may simply be impossible in a physical environment in which, e.g., the searchers are modeling physical searchers moving in a network of pipes. In the case of software agents deployed in a computer network, maintaining several searchers at the same node may consume local resources (e.g., memory, computation cycles, etc.). The exclusivity constraint aims at dealing with this problem.

More formally, given a connected n-node graph G, an *exclusive search strategy* in G, using $k \leq n$ searchers consists of (1) placing the k searchers at k different vertices of G, and (2) performing a sequence of slidings ensuring the exclusivity constraint. An edge becomes clear whenever either a searcher slides along it, or one searcher is placed at each of its extremities (as in mixed-search). The exclusive-search number of G, denoted by $\texttt{xs}(G)$ is the smallest k for which there exists a winning search strategy in G. Exclusive graph searching behaves very differently from classical variants. For instance, $\texttt{xs}(S_n) = n - 1$ for any star S_n with $n \geq 3$ leaves. More important, it is not monotone even in trees and it is not closed under taking subgraphs [BBN17]. It has been proved that, for any graph G with maximum degree Δ, $\texttt{ns}(G) \leq \texttt{xs}(G) \leq (\Delta-1)(\texttt{ns}(G)+1)$ [BBN17]. Surprisingly, computing the monotone exclusive search number is NP-hard in split graphs (where pathwidth can be polynomially computed) and can be solved in polynomial time in a subclass of star-like graphs (where pathwidth is NP-hard) [MNP17]. A linear-time algorithm in cographs is also proposed in [MNP17]. A polynomial-time algorithm that computes the monotone exclusive search number of trees has been designed in [BBN17]. It is based on a lemma in the same vein as Parsons' lemma (while more technical) and then follows the same principles as the algorithm of Ellis *et al.* for edge-search (see Lemma 1) but the proof is more technical due to the non-closedness under subgraph.

Open Questions. *Is the problem of deciding (when k is part of the input) whether $\texttt{xs}(G) \leq k$ in* NP*? Is it polynomial when k is fixed? Is it FPT in k? Also the study of the exclusive search number in various graph classes is still open.*

Distributed exclusive graph searching has been studied in the `Look-Compute-Move` model where searchers have very weak abilities (they are anonymous and oblivious) but can "see" the whole network (see Chapters 8 and 9 for more details). Algorithms for paths and trees, using the optimal number of searchers (or more), have been designed in [BBN12] and the case of cycles is studied in [DSN$^+$15, DNN17]. The algorithm in cycles relies on a subprocedure that places the searchers in an adequate configuration that can also be used to solve other coordination problems such as *gathering* and *perpetual exploration.*

Open Questions. *One intriguing remaining question in the cycle is whether* 4 *searchers can exclusively clear any cycle with at least* 10 *vertices in the* `Look-`

`Compute-Move` *model. Indeed, for any n-node cycle with* $n > 10$*, it is possible to clear it with* $k \in \{5, \cdots, n-3\} \cup \{n\}$ *searchers and not possible with* ≤ 3 *searchers or* $k \in \{n-2; n-1\}$ *searchers [DSN+15].*

4 Plethora of alternative models

Recall that, in the Introduction, it was mentioned that the network decontamination problem can be equivalently seen as a pursuit-evasion game between a team of searchers and a *fugitive* (an intruder, a lost spelunker...). Variants of graph searching that have been described so far can all be stated in terms of capturing a lucky invisible and arbitrarily fast fugitive in a graph. By "lucky" (or "omniscient"), we mean that the objective is the design of a strategy that captures the fugitive in the worst case, i.e., whatever be the behavior of the fugitive. From now on, we use the pursuit-evasion terminology (except in Subsection 4.3) because it fits the proposed models better.

4.1 Visible/Inert fugitive and Tree-like structures

Visible fugitive. A first natural extension of node-search concerns the case of a visible fugitive. In this variant, the fugitive occupies a vertex that is known by the searchers but may move at any step to another (known) vertex. In particular, if a step of a strategy consists of placing a searcher at the vertex occupied by the fugitive, the latter may simultaneously (before the searcher "lands") move to any vertex it can access (through a path free of searchers). The *visible search number* of a graph G, denoted by $\texttt{vns}(G)$, is the minimum number of searchers required to catch a visible fugitive in this setting. For instance, $\texttt{vns}(T) = 2$ for any tree T (not reduced to a single vertex) while, for any $n \in \mathbb{N}^*$, there are n-node trees T such that $\texttt{ns}(T) = \Theta(\log n)$ (see Section 2.3). The visible search number shares a relationship with *treewidth*[5], denoted by $\texttt{tw}(G)$, that is similar to the relationship between pathwidth and node-search number. Precisely:

Theorem 4. *[ST93] For any graph* G*,* $\texttt{vns}(G) = \texttt{tw}(G) + 1$*.*

As in the case of pathwidth and node-search, it is easy to show that monotone visible node-search strategies are equivalent to tree-decompositions. Again, the difficulty is to prove that there always exists an optimal strategy that is monotone. Seymour and Thomas proved the monotonicity of visible graph searching by defining a dual structure for the tree-decompositions, namely the *brambles* (initially called *screens*) [ST93], that actually corresponds to a winning strategy for the fugitive. Given a graph $G = (V, E)$, a *bramble* is a family $\mathcal{B} = (B_i)_{0 \leq i \leq \ell}$ of subsets of vertices such that (1) B_i induces a connected subgraph of G for each i and (2) the sets B_i are pairwise *touching* (i.e., any two sets intersect or there exists an edge linking them). The *order* of $\mathcal{B}$ is the minimum size of a

[5] Due to the huge number of works on treewidth, we have decided not to detail them (nor the definition of treewidth) and refer the reader to [Bod98, Die12, CFK+15].

hitting set, i.e., the smallest number of vertices in V that intersect each set in $\mathcal{B}$. The treewidth of a graph G is at most $k-1 \in \mathbb{N}$ (and so $\texttt{vns}(G) \leq k$) if and only if the maximum order of a bramble of G is k [ST93]. Given a graph G with a bramble $\mathcal{B}$ of order $k+1$, it is easy to describe a winning strategy for the fugitive against $\leq k$ searchers. Indeed, at every step, the fugitive can move (since the sets are connected and pairwise touching) to a set of $\mathcal{B}$ whose vertices are occupied by no searcher. The notion of bramble is very useful to prove lower bounds on the visible search number of graphs. For instance, it is easy to show that $\texttt{vns}(G_{n\times n}) \leq n+1$ in any $n \times n$ grid $G_{n\times n}$ and a bramble of order n in $G_{n\times n}$ can also easily be found. Altogether, this proves that $\texttt{vns}(G_{n\times n}) = n+1$ (such a result is rather technical without the help of brambles).

The connected capture of a visible fugitive has been studied in [FN08]. As its invisible counterpart, it is not monotone. However, in contrast with the invisible case (Theorem 3), this variant may require $\Omega(\log n * \texttt{vns}(G))$ searchers in some n-node graphs G and this is asymptotically tight [FN08].

Non-deterministic Graph Searching. *Non-deterministic graph searching* generalizes both node-search and visible node-search [FFN09]. Given a fixed integer $q \geq 0$, a non-deterministic strategy aims at catching an invisible fugitive with the additional ability that the fugitive is visible during at most q steps of the game (the choice of when to see the fugitive is left to the searchers dynamically during the strategy). The minimum number of searchers required to catch the fugitive in this setting is denoted by $\texttt{ns}_q(G)$. By definition, $\texttt{ns}_0(G) = \texttt{ns}(G)$ (the fugitive is always invisible) and $\texttt{ns}_\infty(G) = \texttt{vns}(G)$ (the fugitive is always visible). Computing $\texttt{ns}_q(G)$ is NP-hard for any $q \geq 0$ and an exponential-time algorithm to compute it is presented in [FFN09]. The monotonicity of this variant is proved in [MN08] and a constructive FPT algorithm is designed in [BBM$^+$13]. A polynomial-time dynamic programming algorithm to compute a 2-approximation of $\texttt{ns}_q(T)$ in the class of trees T (exact for $q \leq 1$) is designed in [ACN15].

Open Questions. *The existence of an exact polynomial-time algorithm that computes $\texttt{ns}_q(T)$ in any tree T and for any $q > 1$ is still open.*

Another interesting open question is the definition of a dual structure (similar to brambles for visible node-search [ST93] or to blockage for node-search [BRST91]) for non-deterministic graph searching.

Inert fugitive. Another variant of node-search is related to tree-decompositions. A fugitive is *inert* (*a.k.a., lazy*) if it is invisible but can only move if a searcher is landing at the vertex it is currently occupying. In any graph G, the minimum number of searchers required to catch the fugitive in this setting also equals the treewidth of G plus one [RT11].

LIFO-search. Last but not least, let us mention a variant of graph searching related to another tree-like parameter of graphs. Namely, *LIFO-search* is a variant of node-search where the searchers are labelled with distinct integers and with the extra constraint that a searcher can be removed only if no searcher

with smaller label is present in the graph [GHT12]. In [GHT12], this variant is proved to be monotone and equivalent to the *tree-depth* of graphs [NdM08].

4.2 Directed graphs

During the last decade, several digraph decompositions have been proposed in order to try to bring to directed graphs the same algorithmic power as tree-decompositions provide for undirected graphs [GHK$^+$16]. Interestingly, most of these attempts have been defined through graph searching games. An important difference between directed graph searching games and undirected ones arises via the notion of monotonicity. In the directed case, there are two distinct definitions of monotonicity: a game is *cop-monotone* if each vertex is occupied at most once by a searcher, it is *robber-monotone* if the area reachable by the fugitive never increases. Clearly a cop-monotone game is robber-monotone. However, as shown below, the converse is not always true.

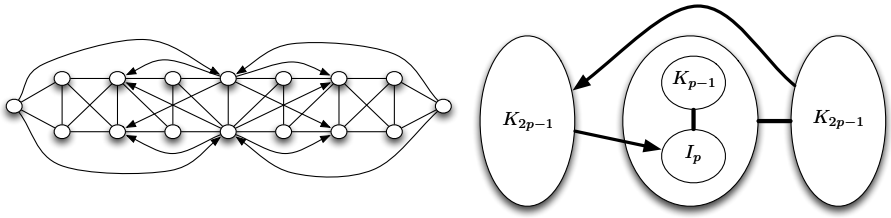

Fig. 8: On the left, a graph in which 4 searchers can capture a visible fugitive constrained to move in strongly connected components free of searchers (directed treewidth variant) but every robber-monotone strategy requires 5 searchers [Adl07].
On the right, the schematic overview of a graph where $3p-1$ searchers can capture a visible fugitive constrained to follow the orientation of the arcs (DAG-width variant) but every monotone strategy requires $4p-2$ searchers, for any $p \geq 2$ [KO11]. K_x denotes a clique on x vertices and I_x is an independent set on x vertices. A directed edge between two parts A and B means that there are edges from every vertex in A to every vertex in B. Undirected edges mean that there are edges between A and B in both directions.

Johnson *et al.* first defined the *directed tree-decomposition* which roughly "translates" the connectivity properties of tree-decomposition into strong connectivity properties in directed graphs [JRST01]. Their variant is closely related to the graph searching game where a visible fugitive has the extra constraint that it can move only in strongly connected components free of searchers. That is, the fugitive can go from vertices u to v if there is a directed path from u to v free of searchers and a directed path from v to u free of searchers. It has

been shown that, in this game, the non-monotone, the cop-monotone and the robber-monotone variants may differ [JRST01, Adl07] (see Fig. 8, left). Because of the non-monotonicity result, no min-max theorem can be expected via graph searching. However, [JRST01] proved a weaker result: if k searchers have a winning strategy in a digraph D, then $3k-1$ searchers have a robber-monotone winning strategy in D, which leads to a min-max theorem up to a constant ratio between directed treewidth and so-called *havens* [JRST01]. In [EHS13], it is proved that the cop-monotone version of this game is actually equivalent to the D-width defined by Safari [Saf05] leading to an exact algorithm for computing this variant. Moreover, [EHS13] showed that D-width and directed treewidth are actually equivalent (in the sense that one is bounded if and only if the other is bounded).

The DAG-decomposition is weaker than directed tree-decomposition (bounded DAG-width implies bounded directed treewidth) [BDH$^+$12]. It corresponds to the cop-monotone version of the game where the visible fugitive is constrained to follow the direction of arcs. While robber-monotone and cop-monotone variants coincide [BDH$^+$12], this game is not monotone [KO11] (see Fig. 8, right). However, a drawback of DAG-decomposition is that the best known upper bound of the size of such a decomposition with width k in an n-node digraph is $O(n^k)$ and that the corresponding optimization problem is PSPACE-complete [AKR16]. Another decomposition weaker than directed tree-decomposition is the Kelly-decomposition that corresponds to the robber-monotone variant of the the game where an inert fugitive is forced to follow the arcs [HK08]. Again, this game is not monotone [KO11]. A polynomial-time algorithm to recognize digraphs with Kelly-width at most 2 is given in [MTV10]. Approximation algorithms for computing directed treewidth, Dag- and Kelly-width, with approximation ratio $O(\log^{3/2} n)$ have been designed in [KKK15].

To conclude, let us mention that several directed path-decompositions and directed invisible graph searching variants have also been proposed. These variants mainly differ (1) in the abilities of the searchers and the fugitive: either both have to follow the direction of arcs, or only one of them, and (2) in the variant of graph searching that is considered: edge, node or mixed. More details can be found in [Bar06, YC07b, ADHY07, YC07a, Yan07, YC08c, YC08b, YC08a, YC09]. Contrary to their visible counterparts, all these directed variants are monotone.

4.3 Recontamination alternatives

In classical graph searching, a vertex is recontaminated instantaneously if it is not occupied and adjacent to a contaminated vertex. Flocchini *et al.* proposed several alternative definitions for recontamination. In a first variant, with *threshold immunity* or *local immunity*, a vertex can be recontaminated only if a sufficient number of its neighbors are contaminated [LPS06]. In a second model, with *temporal immunity*, a vertex can be recontaminated only t steps after it has been left by a searcher, where $t \geq 0$ is a parameter [FMS08].

Local immunity. In [LPS06], a clear (and unoccupied) vertex is recontaminated if more than half of its neighbors are contaminated. In this setting, any graph with maximum degree three can be cleared by at most 2 searchers and by a single one if moreover there is a pendant vertex. [LPS06] gave search protocols for tori that are optimal in terms of number of searchers and asymptotically tight for the number of moves. They also considered the case of trees. Their results have been extended by the generalization proposed in [FLPS16].

In [FLPS16], the parameterized version of this problem is considered where the parameter $m \geq 1$ represents the minimum number of neighbors of a vertex v that must be contaminated to recontaminate v. In $n_1 \times \cdots \times n_d$ d-dimensional grids, one searcher is sufficient for any $m \geq d$, $\Pi_{j=1}^{d-m} n_j$ searchers are sufficient otherwise. In the case of the $n_1 \times \cdots \times n_d$ d-dimensional torus, $2^d \cdot \Pi_{j=1}^{d-m} n_j$ searchers are sufficient for $m \leq d-1$ and 2^{2d-m} searchers are sufficient for $d \leq m \leq 2d$.

Open Questions. *It is not known whether these upper bounds are tight.*

Finally, a dynamic programming algorithm that computes optimal monotone search strategies in trees has also been designed [FLPS16].

Temporal immunity. A graph G has *temporal immunity* $t \geq 0$ if a vertex becomes recontaminated after having been exposed (i.e., unoccupied and with a contaminated neighbor) during more than t steps [FMS08]. As an example, assume that $t = 2$, then an n-node cycle can be cleared by a single searcher moving clockwise during $2n$ steps (note that the strategy is not monotone). [DJS16] defined the *immunity number* $\iota_k(G)$ of G as the minimum integer $t \geq 0$ such that $k \geq 1$ searchers can clear G with temporal immunity t.

A distributed algorithm for computing the minimum number of searchers needed to clear a tree with temporal immunity $t \geq 0$ has been designed and a structural characterization of trees with $\iota_k(T) = t$ is provided [FMS08]. Roughly, $\iota_k(T) = t$ if and only if T does not contain a subtree obtained from the complete ternary tree of height k whose all edges have been subdivided $\lceil \frac{t}{2} \rceil + 1$ times. Finally, an algorithm for clearing any tree of height at most h with $\lfloor \frac{2h}{t+2} \rfloor$ searchers is presented in [FMS08].

For any $k \in \{1, 2, 4\}$, any $n \times n$ grid with temporal immunity at least $(4-k)(n-1)-1$ can be cleared by k searchers [DFZ10], no results for other number of searchers are known. In the case of *strong grids*, k searchers are sufficient to clear them when the immunity is at least $\lfloor \frac{2(2n-1)}{k} \rfloor$ [DFZ10].

Finally, $\iota_1(G)$ has been studied in several classes of graphs [DJS16] such as paths: $\iota_1(P_n) = 0$ for every n; cycles: $\iota_1(C_n) = 2$ for every n, and equals $n-1$ if monotonicity is required; complete graphs: $\iota_1(K_n) = n-1$ for every n; complete bipartite graphs: $\iota_1(K_{n,m}) = 2m-1$ for $3 \leq m \leq n$; n-node trees: $\iota_1(T) \leq 30\sqrt{n}$; $p \times q$ grids: $p/2 \leq \iota_1 \leq p$ for $p \leq q$, *etc.* It can be shown that there are n-node trees T for which $\iota_1(T) = \Omega(n^{1/3+\epsilon})$ for some constant $\epsilon > 0$ [DJS16].

Open Questions. *A challenge would be to close the gap with the upper bound $30\sqrt{n}$ in trees. The question of general planar graphs is also open.*

4.4 Other models and objectives

To conclude this chapter, we would like to mention some variants of graph searching that differ from previous ones by: the objective that must be optimized, the way the fugitive is captured, the speed of the fugitive, etc. We only mention some of these variants, others may be found in [FT08].

Different objectives. The *cost* of a strategy is the sum of the number of occupied vertices over all steps of a strategy. This parameter (in the node-search variant) appears to equal the *profile* of the graph G (minimum number of edges of an interval supergraph of G) [FG00, Fom04], while, in the visible (or inert) variant, it equals the minimum *fill in* of G (minimum number of edges of a chordal supergraph). The cost in the case of edge-search has been studied in [DD13]. The *maximum occupation time* is the maximum over all vertices of the number of steps during which a vertex is occupied. This parameter coincides with the *bandwidth* of graphs [FHT05, Der09]. The *capture time* (minimum number of bags of a path-decomposition with a given width) has also been considered in [BH06, DKZ15].

Different speeds. The case of a fugitive with bounded *speed* (the fugitive has speed s if, at every step, it can move through a path of length at most s) has been considered in [Fom98, Fom99], and the case of an inert fugitive with bounded speed is considered in [DKT97].

Different rules. [DYY08] introduced the *fast searching game* in which the searchers cannot be removed and every edge can be traversed only once. This variant has been studied in [SY09, Yan11, SY11, Yan13, DDD13, XYZZ16, XY17]. See also [MNP08, KP16] (and references therein) for the so-called *brush game*.

Different applications. Surprisingly, a variant of graph searching has been defined to model the problem of routing reconfiguration in optical (WDM) networks [CHM$^+$09]. In this variant, an invisible fugitive is moving following the orientation of the arcs in a directed graph and it is captured as soon as the searchers constrain the moves of the fugitive to a component that is not strongly connected. Monotonicity [NS16], computational complexity [CS11, CHM12] as well as tradeoff between the number of searchers and the cost of strategies [CCM$^+$11] have been studied.

References

[ABC+15] Brendan P. W. Ames, Andrew Beveridge, Rosalie Carlson, Claire Djang, Volkan Isler, Stephen Ragain, and Maxray Savage. A leapfrog strategy for pursuit-evasion in a polygonal environment. *Int. J. Comput. Geometry Appl.*, 25(2):77–100, 2015.

[ACN15] Omid Amini, David Coudert, and Nicolas Nisse. Non-deterministic graph searching in trees. *Theor. Comput. Sci.*, 580:101–121, 2015.

[ADHY07] Brian Alspach, Danny Dyer, Denis Hanson, and Boting Yang. Arc searching digraphs without jumping. In *Proc. of first int. conf. on Combinatorial Optimization and Applications (COCOA)*, volume 4616 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2007.

[Adl07] Isolde Adler. Directed tree-width examples. *J. Comb. Theory, Ser. B*, 97(5):718–725, 2007.

[AKR16] Saeed Akhoondian Amiri, Stephan Kreutzer, and Roman Rabinovich. Dag-width is pspace-complete. *Theor. Comput. Sci.*, 655:78–89, 2016.

[Als04] Brian Alspach. Searching and sweeping graphs: a brief survey. *Mathematiche*, 59:65–37, 2004.

[Bar06] János Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

[BBM+13] Pascal Berthomé, Tom Bouvier, Frédéric Mazoit, Nicolas Nisse, and Ronan Pardo Soares. An Unified FPT Algorithm for Width of Partition Functions. Research Report RR-8372, INRIA, September 2013.

[BBN12] Lélia Blin, Janna Burman, and Nicolas Nisse. Brief announcement: Distributed exclusive and perpetual tree searching. In *Proc. of 26th International Symposium on Distributed Computing (DISC)*, volume 7611 of *Lecture Notes in Computer Science*, pages 403–404. Springer, 2012.

[BBN17] Lélia Blin, Janna Burman, and Nicolas Nisse. Exclusive graph searching. *Algorithmica*, 77(3):942–969, 2017.

[BDH+12] Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdrzálek. The dag-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012.

[BDK15] Piotr Borowiecki, Dariusz Dereniowski, and Lukasz Kuszner. Distributed graph searching with a sense of direction. *Distributed Computing*, 28(3):155–170, 2015.

[BF02] Hans L. Bodlaender and Fedor V. Fomin. Approximation of pathwidth of outerplanar graphs. *J. Algorithms*, 43(2):190–200, 2002.

[BFF+12] Lali Barrière, Paola Flocchini, Fedor V. Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, and Dimitrios M. Thilikos. Connected graph searching. *Inf. Comput.*, 219:1–16, 2012.

[BFFS02] Lali Barrière, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proc. of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 200–209, 2002.

[BFL+09] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. In *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 629–638. IEEE Computer Society, 2009.

[BFNV08] Lélia Blin, Pierre Fraigniaud, Nicolas Nisse, and Sandrine Vial. Distributed chasing of network intruders. *Theor. Comput. Sci.*, 399(1-2):12–37, 2008.

[BFST03] Lali Barrière, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M. Thilikos. Searching is not jumping. In *Proc. of 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 2880 of *Lecture Notes in Computer Science*, pages 34–45. Springer, 2003.

[BGTZ16] Micah J. Best, Arvind Gupta, Dimitrios M. Thilikos, and Dimitris Zoros. Contraction obstructions for connected graph searching. *Discrete Applied Mathematics*, 209:27–47, 2016.

[BH06] Franz-Josef Brandenburg and Stephanie Herrmann. Graph searching and search time. In *32nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 3831 of *Lecture Notes in Computer Science*, pages 197–206. Springer, 2006.

[Bie91] Daniel Bienstock. Graph searching, path-width, tree-width and related problems (A survey). In *Proc. of Reliability Of Computer And Communication Networks, a DIMACS Workshop*, volume 5 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 33–50. DIMACS/AMS, 1991.

[BK96] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[BKIS12] Deepak Bhadauria, Kyle Klein, Volkan Isler, and Subhash Suri. Capturing an evader in polygonal environments with obstacles: The full visibility case. *I. J. Robotics Res.*, 31(10):1176–1189, 2012.

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[BM93] Hans L. Bodlaender and Rolf H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Discrete Math.*, 6(2):181–188, 1993.

[BMM10] Jean R. S. Blair, Fredrik Manne, and Rodica Mihai. Efficient self-stabilizing graph searching in tree networks. In *Proc. of 12th Int. Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6366 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2010.

[BN11] Anthony Bonato and Richard J. Nowakovski. *The game of Cops and Robber on Graphs*. American Math. Soc., 2011.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.

[Bre67] Richard L. Breisch. An intuitive approach to speleotopology. *Southwestern Cavers*, 6:72–78, 1967.

[Bre12] Richard L. Breish. *Lost in a Cave: applying graph theory to cave exploration*. Greyhound press, 2012.

[BRST91] Daniel Bienstock, Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a forest. *J. Comb. Theory, Ser. B*, 52(2):274–283, 1991.

[BS91] Daniel Bienstock and Paul D. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.

[BT04] Hans L. Bodlaender and Dimitrios M. Thilikos. Computing small search numbers in linear time. In *Proc. of First Int. Workshop on Parameterized and Exact Computation (IWPEC)*, volume 3162 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2004.

[BTK11] Richard B. Borie, Craig A. Tovey, and Sven Koenig. Algorithms and complexity results for graph-based pursuit evasion. *Auton. Robots*, 31(4):317–332, 2011.

[CCM+11] Nathann Cohen, David Coudert, Dorian Mazauric, Napoleão Nepomuceno, and Nicolas Nisse. Tradeoffs in process strategy games with application in the WDM reconfiguration problem. *Theor. Comput. Sci.*, 412(35):4675–4687, 2011.

[CDH+16] Derek G. Corneil, Jérémie Dusart, Michel Habib, Antoine Mamcarz, and Fabien de Montgolfier. A tie-break model for graph search. *Discrete Applied Mathematics*, 199:89–100, 2016.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CHI11] Timothy H. Chung, Geoffrey A. Hollinger, and Volkan Isler. Search and pursuit-evasion in mobile robotics - A survey. *Auton. Robots*, 31(4):299–316, 2011.

[CHM+09] David Coudert, Florian Huc, Dorian Mazauric, Nicolas Nisse, and Jean-Sébastien Sereni. Reconfiguration of the Routing in WDM Networks with Two Classes of Services. In *Conference on Optical Network Design and Modeling (ONDM)*, Braunschweig, Germany, 2009.

[CHM12] David Coudert, Florian Huc, and Dorian Mazauric. A distributed algorithm for computing the node search number in trees. *Algorithmica*, 63(1-2):158–190, 2012.

[CHS07] David Coudert, Florian Huc, and Jean-Sébastien Sereni. Pathwidth of outerplanar graphs. *Journal of Graph Theory*, 55(1):27–41, 2007.

[CK06] L. Sunil Chandran and Telikepalli Kavitha. The treewidth and pathwidth of hypercubes. *Discrete Mathematics*, 306(3):359–365, 2006.

[CM93] Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993.

[CMN16] David Coudert, Dorian Mazauric, and Nicolas Nisse. Experimental evaluation of a branch-and-bound algorithm for computing pathwidth and directed pathwidth. *ACM Journal of Experimental Algorithmics*, 21(1):1.3:1–1.3:23, 2016.

[Cou16] David Coudert. A note on Integer Linear Programming formulations for linear ordering problems on graphs. Research report, Inria ; I3S ; Universite Nice Sophia Antipolis ; CNRS, February 2016.

[CS11] David Coudert and Jean-Sébastien Sereni. Characterization of graphs and digraphs with small process numbers. *Discrete Applied Mathematics*, 159(11):1094–1109, 2011.

[DD13] Dariusz Dereniowski and Danny Dyer. On minimum cost edge searching. *Theor. Comput. Sci.*, 495:37–49, 2013.

[DDD13] Dariusz Dereniowski, Öznur Yasar Diner, and Danny Dyer. Three-fast-searchable graphs. *Discrete Applied Mathematics*, 161(13-14):1950–1958, 2013.

[Der09] Dariusz Dereniowski. Maximum vertex occupation time and inert fugitive: Recontamination does help. *Inf. Process. Lett.*, 109(9):422–426, 2009.

[Der11] Dariusz Dereniowski. Connected searching of weighted trees. *Theor. Comput. Sci.*, 412(41):5700–5713, 2011.

[Der12a] Dariusz Dereniowski. Approximate search strategies for weighted trees. *Theor. Comput. Sci.*, 463:96–113, 2012.

[Der12b] Dariusz Dereniowski. From pathwidth to connected pathwidth. *SIAM J. Discrete Math.*, 26(4):1709–1732, 2012.

[DFZ10] Yassine Daadaa, Paola Flocchini, and Nejib Zaguia. Network decontamination with temporal immunity by cellular automata. In *Proc. of 9th Int. Conference on Cellular Automata for Research and Industry (ACRI)*, volume 6350 of *Lecture Notes in Computer Science*, pages 287–299. Springer, 2010.

[DH08] Erik D. Demaine and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[DJS16] Yassine Daadaa, Asif Jamshed, and Mudassir Shabbir. Network decontamination with a single agent. *Graphs and Combinatorics*, 32(2):559–581, 2016.

[DKL87] Narsingh Deo, Mukkai S. Krishnamoorthy, and Michael A. Langston. Exact and approximate solutions for the gate matrix layout problem. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 6(1):79–84, 1987.

[DKT97] Nick D. Dendris, Lefteris M. Kirousis, and Dimitrios M. Thilikos. Fugitive-search games on graphs and related parameters. *Theor. Comput. Sci.*, 172(1-2):233–254, 1997.

[DKZ15] Dariusz Dereniowski, Wieslaw Kubiak, and Yori Zwols. The complexity of minimum-length path decompositions. *J. Comput. Syst. Sci.*, 81(8):1715–1747, 2015.

[DNN17] Gianlorenzo D'Angelo, Alfredo Navarra, and Nicolas Nisse. A unified approach for gathering and exclusive searching on rings under weak assumptions. *Distributed Computing*, 30(1):17–48, 2017.

[DOR18] Dariusz Dereniowski, Dorota Osula, and Paweł Rzazewski. Finding small-width connected path decompositions in polynomial time. *CoRR*, abs/1802.05501, 2018.

[DSN+15] Gianlorenzo D'Angelo, Gabriele Di Stefano, Alfredo Navarra, Nicolas Nisse, and Karol Suchan. Computing on rings by oblivious robots: A unified approach for different tasks. *Algorithmica*, 72(4):1055–1096, 2015.

[DU16] Dariusz Dereniowski and Dorota Urbanska. Distributed searching of partial grids. *CoRR*, abs/1610.01458, 2016.

[DYY08] Danny Dyer, Boting Yang, and Öznur Yasar. On the fast searching problem. In *Proc. of 4th Int. Conference on Algorithmic Aspects in Information and Management (AAIM)*, volume 5034 of *Lecture Notes in Computer Science*, pages 143–154. Springer, 2008.

[EHS13] William Evans, Paul Hunter, and Mohammad Ali Safari. D-width and cops and robbers. Research report, 2013. unpublished.

[EM04] John A. Ellis and Minko Markov. Computing the vertex separation of unicyclic graphs. *Inf. Comput.*, 192(2):123–161, 2004.

[EST87] John A. Ellis, Ivan H. Sudborough, and Jonathan S. Turner. Graph separation and search number. Technical report, 1987. Report Number: WUCS-87-11.

[EST94] John A. Ellis, Ivan H. Sudborough, and Jonathan S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

[FFN09] Fedor V. Fomin, Pierre Fraigniaud, and Nicolas Nisse. Nondeterministic graph searching: From pathwidth to treewidth. *Algorithmica*, 53(3):358–373, 2009.

[FG00] Fedor V. Fomin and Petr A. Golovach. Graph searching and interval completion. *SIAM J. Discrete Math.*, 13(4):454–464, 2000.

[FHL07] Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Decontaminating chordal rings and tori using mobile agents. *Int. J. Found. Comput. Sci.*, 18(3):547–563, 2007.

[FHL08] Paola Flocchini, Miao Jun Huang, and Flaminia L. Luccio. Decontamination of hypercubes by mobile agents. *Networks*, 52(3):167–178, 2008.

[FHM10] Fedor V. Fomin, Pinar Heggernes, and Rodica Mihai. Mixed search number and linear-width of interval and split graphs. *Networks*, 56(3):207–214, 2010.

[FHT05] Fedor V. Fomin, Pinar Heggernes, and Jan Arne Telle. Graph searching, elimination trees, and a generalization of bandwidth. *Algorithmica*, 41(2):73–87, 2005.

[FIP06] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Oracle size: a new measure of difficulty for communication tasks. In *Proc. of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 179–187. ACM, 2006.

[FL94] Michael R. Fellows and Michael A. Langston. On search, decision, and the efficiency of polynomial-time algorithms. *J. Comput. Syst. Sci.*, 49(3):769–779, 1994.

[FLPS16] Paola Flocchini, Fabrizio Luccio, Linda Pagli, and Nicola Santoro. Network decontamination under m-immunity. *Discrete Applied Mathematics*, 201:114–129, 2016.

[FLS05] Paola Flocchini, Flaminia L. Luccio, and Lisa Xiuli Song. Size optimal strategies for capturing an intruder in mesh networks. In *Proc. of the Int. Conference on Communications in Computing (CIC)*, pages 200–206. CSREA Press, 2005.

[FLS18] Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Excluded grid minors and efficient polynomial-time approximation schemes. *J. ACM*, 65(2):10:1–10:44, 2018.

[FMS08] Paola Flocchini, Bernard Mans, and Nicola Santoro. Tree decontamination with temporary immunity. In *Proc. of 19th Int. Symposium on Algorithms and Computation (ISAAC)*, volume 5369 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2008.

[FN08] Pierre Fraigniaud and Nicolas Nisse. Monotony properties of connected visible graph searching. *Inf. Comput.*, 206(12):1383–1393, 2008.

[FNS07] Paola Flocchini, Amiya Nayak, and Arno Schulz. Decontamination of arbitrary networks using a team of mobile agents with limited visibility. In *6th Annual IEEE/ACIS International Conference on Computer and Information Science (ICIS*, pages 469–474. IEEE Computer Society, 2007.

[Fom98] Fedor V. Fomin. Helicopter search problems, bandwidth and pathwidth. *Discrete Applied Mathematics*, 85(1):59–70, 1998.

[Fom99] Fedor V. Fomin. Note on a helicopter search problem on graphs. *Discrete Applied Mathematics*, 95(1-3):241–249, 1999.

[Fom04] Fedor V. Fomin. Searching expenditure and interval graphs. *Discrete Applied Mathematics*, 135(1-3):97–104, 2004.

[FS06] Paola Flocchini and Nicola Santoro. Distributed security algorithms by mobile agents. In *Distributed Computing and Networking, 8th International Conference, ICDCN*, volume 4308 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[FT08] Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[FTT05] Fedor V. Fomin, Dimitrios M. Thilikos, and Ioan Todinca. Connected graph searching in outerplanar graphs. *Electronic Notes in Discrete Mathematics*, 22:213–216, 2005.

[GHK+16] Robert Ganian, Petr Hlinený, Joachim Kneis, Daniel Meister, Jan Obdrzálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *J. Comb. Theory, Ser. B*, 116:250–286, 2016.

[GHM12] Petr A. Golovach, Pinar Heggernes, and Rodica Mihai. Edge search number of cographs. *Discrete Applied Mathematics*, 160(6):734–743, 2012.

[GHT12] Archontia C. Giannopoulou, Paul Hunter, and Dimitrios M. Thilikos. Lifo-search: A min-max theorem and a searching game for cycle-rank and tree-depth. *Discrete Applied Mathematics*, 160(15):2089–2097, 2012.

[GLL+99] Leonidas J. Guibas, Jean-Claude Latombe, Steven M. LaValle, David Lin, and Rajeev Motwani. A visibility-based pursuit-evasion problem. *Int. J. Comput. Geometry Appl.*, 9(4/5):471–494, 1999.

[Gol89a] Petr A. Golovach. Equivalence of two formalizations of a search problem on a graph. *Vestnik Leningrad Univ. Math*, 22:13–19, 1989.

[Gol89b] Petr A. Golovach. A topological invariant in pursuit problems. *Differ. Equ.*, 25:657–661, 1989.

[Gus93] Jens Gustedt. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233–248, 1993.

[HK08] Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science*, 399(3):206–219, 2008.

[HM08] Pinar Heggernes and Rodica Mihai. Mixed search number of permutation graphs. In *Proc. of Second Annual Int. Workshop on Frontiers in Algorithmics FAW*, volume 5059 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2008.

[INS09] David Ilcinkas, Nicolas Nisse, and David Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing*, 22(2):117–127, 2009.

[ISZ07] Navid Imani, Hamid Sarbazi-Azad, and Albert Y. Zomaya. Capturing an intruder in product networks. *J. Parallel Distrib. Comput.*, 67(9):1018–1028, 2007.

[ISZ08] Navid Imani, Hamid Sarbazi-Azad, and Albert Y. Zomaya. Intruder capturing in mesh and torus networks. *Int. J. Found. Comput. Sci.*, 19(4):1049–1071, 2008.

[JRST01] Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

[Kin92] Nancy G. Kinnersley. The vertex separation number of a graph equals its pathwidth. *Inform. Process. Lett.*, 1992.

[KKK15] Shiva Kintali, Nishad Kothari, and Akash Kumar. Approximation algorithms for digraph width parameters. *Theor. Comput. Sci.*, 562:365–376, 2015.

[KKK+16] Kenta Kitsunai, Yasuaki Kobayashi, Keita Komuro, Hisao Tamaki, and Toshihiro Tano. Computing directed pathwidth in $o(1.89^n)$ time. *Algorithmica*, 75(1):138–157, 2016.

[KO11] Stephan Kreutzer and Sebastian Ordyniak. Digraph decompositions and monotonicity in digraph searching. *Theor. Comput. Sci.*, 412(35):4688–4703, 2011.

[KP85] Lefteris M. Kirousis and Christos H. Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55(2):181–184, 1985.

[KP86] Lefteris M. Kirousis and Christos H. Papadimitriou. Searching and pebbling. *Theor. Comput. Sci.*, 47(3):205–218, 1986.

[KP16] William B. Kinnersley and Pawel Pralat. Game brush number. *Discrete Applied Mathematics*, 207:1–14, 2016.

[KS15] Kyle Klein and Subhash Suri. Pursuit evasion on polyhedral surfaces. *Algorithmica*, 73(4):730–747, 2015.

[LaP93] Andrea S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.

[LPS06] Fabrizio Luccio, Linda Pagli, and Nicola Santoro. Network decontamination with local immunization. In *Proc. of 20th Int. Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2006.

[Luc09] Flaminia L. Luccio. Contiguous search problem in sierpinski graphs. *Theory Comput. Syst.*, 44(2):186–204, 2009.

[Mal18] Sven Mallach. Linear ordering based MIP formulations for the vertex separation or pathwidth problem. In *Proc. of 28th Int. Workshop on Combinatorial Algorithms (IWOCA)*, volume 10765 of *Lecture Notes in Computer Science*, pages 327–340. Springer, 2018.

[MHG+88] Nimrod Megiddo, S. Louis Hakimi, M. R. Garey, David S. Johnson, and Christos H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, 1988.

[MM09] Rodica Mihai and Morten Mjelde. A self-stabilizing algorithm for graph searching in trees. In *Proc. of 11th Int. Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 5873 of *Lecture Notes in Computer Science*, pages 563–577. Springer, 2009.

[MN08] Frédéric Mazoit and Nicolas Nisse. Monotonicity of non-deterministic graph searching. *Theor. Comput. Sci.*, 399(3):169–178, 2008.

[MNP08] Margaret-Ellen Messinger, Richard J. Nowakowski, and Pawel Pralat. Cleaning a network with brushes. *Theor. Comput. Sci.*, 399(3):191–205, 2008.

[MNP17] Euripides Markou, Nicolas Nisse, and Stéphane Pérennes. Exclusive graph searching vs. pathwidth. *Inf. Comput.*, 252:243–260, 2017.

[MS88] Burkhard Monien and Ivan H. Sudborough. Min cut is np-complete for edge weighted trees. *Theoretical Computer Science*, 58(1):209 – 229, 1988.

[MT09] Rodica Mihai and Ioan Todinca. Pathwidth is np-hard for weighted trees. In *Proc. of Third International Workshop on Frontiers in Algorithmics (FAW)*, volume 5598 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2009.

[MTV10] Daniel Meister, Jan Arne Telle, and Martin Vatshelle. Recognizing digraphs of kelly-width 2. *Discrete Applied Mathematics*, 158(7):741–746, 2010.

[NdM08] Jaroslav Nesetril and Patrice Ossona de Mendez. Grad and classes with bounded expansion i. decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008.

[Nis09] Nicolas Nisse. Connected graph searching in chordal graphs. *Discrete Applied Mathematics*, 157(12):2603–2610, 2009.

[Nis14] Nicolas Nisse. *Algorithmic complexity: Between Structure and Knowledge How Pursuit-evasion Games help.* 2014. Habilitation à Diriger des Recherches, Univ. Nice Sophia-Antipolis, https://tel.archives-ouvertes.fr/tel-00998854.

[NS09] Nicolas Nisse and David Soguet. Graph searching with advice. *Theor. Comput. Sci.*, 410(14):1307–1318, 2009.

[NS16] Nicolas Nisse and Ronan Pardo Soares. On the monotonicity of process number. *Discrete Applied Mathematics*, 210:103–111, 2016.

[Par78a] Torrence D. Parsons. Pursuit-evasion in a graph. In *International Conference on Theory and applications of graphs*, pages 426–441. Lecture Notes in Math., Vol. 642. Springer, Berlin, 1978.

[Par78b] Torrence D. Parsons. The search number of a connected graph. In *9th Southeastern Conf. on Combinatorics, Graph Theory, and Computing*, Congress. Numer., XXI, pages 549–554. Utilitas Math., 1978.

[Pet82] Nikolai N. Petrov. A problem of pursuit in the absence of information on the pursued. *Differ. Uravn.*, 18:1345–1352, 1982.

[PHH+00] Sheng-Lung Peng, Chin-Wen Ho, Tsan-sheng Hsu, Ming-Tat Ko, and Chuan Yi Tang. Edge and node searching problems on trees. *Theor. Comput. Sci.*, 240(2):429–446, 2000.

[PSS13] John Penuel, J. Cole Smith, and Siqian Shen. Integer programming models and algorithms for the graph decontamination problem with mobile agents. *Networks*, 61(1):1–19, 2013.

[PTK+00] Sheng-Lung Peng, Chuan Yi Tang, Ming-Tat Ko, Chin-Wen Ho, and Tsan-sheng Hsu. Graph searching on some subclasses of chordal graphs. *Algorithmica*, 27(3):395–426, 2000.

[PY07] Sheng-Lung Peng and Yi-Chuan Yang. On the treewidth and pathwidth of biconvex bipartite graphs. In *Proc. of 4th Int. on Theory and Applications of Models of Computation*, volume 4484 of *Lecture Notes in Computer Science*, pages 244–255. Springer, 2007.

[RS83] Neil Robertson and Paul D. Seymour. Graph minors. I. excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983.

[RS90] Neil Robertson and Paul D. Seymour. Graph minors. IV. tree-width and well-quasi-ordering. *J. Comb. Theory, Ser. B*, 48(2):227–254, 1990.

[RS95] Neil Robertson and Paul D. Seymour. Graph minors XIII. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[RS04] Neil Robertson and Paul D. Seymour. Graph minors. XX. wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

[RT11] David Richerby and Dimitrios M. Thilikos. Searching for a visible, lazy fugitive. *SIAM J. Discrete Math.*, 25(2):497–513, 2011.

[Saf05] Mohammad Ali Safari. D-width: A more natural measure for directed tree width. In *30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 3618 of *Lecture Notes in Computer Science*, pages 745–756. Springer, 2005.

[SIS06] Pooya Shareghi, Navid Imani, and Hamid Sarbazi-Azad. Capturing an intruder in the pyramid. In *Proc. of First Int. Computer Science Symposium in Russia on Computer Science - Theory and Applications (CSR)*, volume 3967 of *Lecture Notes in Computer Science*, pages 580–590. Springer, 2006.

[Sko03] Konstantin Skodinis. Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time. *J. Algorithms*, 47(1):40–59, 2003.

[ST93] Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, 58(1):22–33, 1993.

[ST07] Karol Suchan and Ioan Todinca. Pathwidth of circular-arc graphs. In *33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 4769 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2007.

[SV09] Karol Suchan and Yngve Villanger. Computing pathwidth faster than 2^n. In *Proc. of 4th Int. Workshop on Parameterized and Exact Computation*

(IWPEC), volume 5917 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 2009.

[SY09] Donald Stanley and Boting Yang. Lower bounds on fast searching. In *Proc. of 20th Int. Symposium on Algorithms and Computation (ISAAC)*, volume 5878 of *Lecture Notes in Computer Science*, pages 964–973. Springer, 2009.

[SY11] Donald Stanley and Boting Yang. Fast searching games on graphs. *J. Comb. Optim.*, 22(4):763–777, 2011.

[Thi00] Dimitrios M. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics*, 105(1-3):239–271, 2000.

[TUK95] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-path-width. *Theor. Comput. Sci.*, 137(2):253–268, 1995.

[WAPL14] Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth and related problems. *J. Artif. Intell. Res.*, 49:569–600, 2014.

[XY17] Yuan Xue and Boting Yang. The fast search number of a cartesian product of graphs. *Discrete Applied Mathematics*, 224:106–119, 2017.

[XYZZ16] Yuan Xue, Boting Yang, Farong Zhong, and Sandra Zilles. Fast searching on complete k-partite graphs. In *Proc. of 10th Int. Conference on Combinatorial Optimization and Applications (COCOA)*, volume 10043 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2016.

[Yan07] Boting Yang. Strong-mixed searching and pathwidth. *J. Comb. Optim.*, 13(1):47–59, 2007.

[Yan11] Boting Yang. Fast edge searching and fast searching on graphs. *Theor. Comput. Sci.*, 412(12-14):1208–1219, 2011.

[Yan13] Boting Yang. Fast-mixed searching and related problems on graphs. *Theor. Comput. Sci.*, 507:100–113, 2013.

[YC07a] Boting Yang and Yi Cao. Directed searching digraphs: Monotonicity and complexity. In *Proc. of 4th International Conference Theory and Applications of Models of Computation (TAMC)*, volume 4484 of *Lecture Notes in Computer Science*, pages 136–147. Springer, 2007.

[YC07b] Boting Yang and Yi Cao. Monotonicity of strong searching on digraphs. *J. Comb. Optim.*, 14(4):411–425, 2007.

[YC08a] Boting Yang and Yi Cao. Digraph searching, directed vertex separation and directed pathwidth. *Discrete Applied Mathematics*, 156(10):1822–1837, 2008.

[YC08b] Boting Yang and Yi Cao. Monotonicity in digraph search problems. *Theor. Comput. Sci.*, 407(1-3):532–544, 2008.

[YC08c] Boting Yang and Yi Cao. On the monotonicity of weak searching. In *Proc. of 14th Annual Int. Conference on Computing and Combinatorics (COCOON)*, volume 5092 of *Lecture Notes in Computer Science*, pages 52–61. Springer, 2008.

[YC09] Boting Yang and Yi Cao. Standard directed search strategies and their applications. *J. Comb. Optim.*, 17(4):378–399, 2009.

[YDA09] Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, 2009.

[YZC07] Boting Yang, Runtao Zhang, and Yi Cao. Searching cycle-disjoint graphs. In *Proc of first int. Conference on Combinatorial Optimization and Applications (COCOA)*, volume 4616 of *Lecture Notes in Computer Science*, pages 32–43. Springer, 2007.