

Searching by Heterogeneous Agents[‡]

Dariusz Dereniowski¹, Łukasz Kuszner², and Robert Ostrowski¹

¹Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Poland, deren@eti.pg.edu.pl, robostro@student.pg.gda.pl

²Faculty of Mathematics, Physics and Informatics, University of Gdańsk, Poland, lkuszner@inf.ug.edu.pl

Abstract

In this work we introduce and study a pursuit-evasion game in which the search is performed by heterogeneous entities. We incorporate heterogeneity into the classical edge search problem by considering edge-labeled graphs: once a search strategy initially assigns labels to the searchers, each searcher can be only present on an edge of its own label. We prove that this problem is not monotone even for trees and we give instances in which the number of recontamination events is asymptotically quadratic in the tree size. Other negative results regard the NP-completeness of the monotone, and NP-hardness of an arbitrary (i.e., non-monotone) heterogeneous search in trees. These properties show that this problem behaves very differently from the classical edge search. On the other hand, if all edges of a particular label form a (connected) subtree of the input tree, then we show that optimal heterogeneous search strategy can be computed efficiently.

Keywords: edge search, graph searching, mobile agent computing, monotonicity, pursuit-evasion

1 Introduction

Consider a scenario in which a team of searchers should propose a search strategy, i.e., a sequence of their moves, that results in capturing a fast and invisible fugitive hiding in a graph. This strategy should succeed regardless of the actions of the fugitive and the fugitive is considered captured when at some point it shares the same location with a searcher. In a strategy, the searchers may perform the following moves: a searcher may be placed/removed on/from a vertex of the graph, and a searcher may slide along an edge from currently occupied vertex to its neighbor. The fugitive may represent an entity that does not want to be captured but may as well be an entity that wants to be found but is constantly moving and the searchers cannot make any assumptions on its behavior. There are numerous models of graph searching that have been introduced and studied and these models can be produced by enforcing some properties of the fugitive (e.g., visibility, speed, randomness of its movements), properties of the searchers (e.g., speed, type of knowledge provided as an input or during the search, restricted movements, radius of capture), types of graphs (e.g., simple, directed) or by considering different optimization criteria (e.g., number of searchers, search cost, search time).

One of the central concepts in graph searching theory is *monotonicity*. Informally speaking, if a search strategy has the property that once a searcher traversed an edge (and by this action it has been verified that in this very moment the fugitive is not present on this edge) it is guaranteed (by the future actions of the searchers) that the edge remains inaccessible to the fugitive, then we say that the search strategy is *monotone*. In most graph searching models, including the edge search recalled above, it is not beneficial to consider search strategies that are not monotone. Such a property is crucial for two main reasons: firstly, knowing that monotone strategies include optimal ones reduces the algorithmic search space when finding good strategies and secondly, monotonicity places the problem in the class NP.

*Research partially supported by National Science Centre (Poland) grant number 2015/17/B/ST6/01887.

†A preliminary version of this paper appeared in the Proc. 11th International Conference on Algorithms and Complexity (CIAC 2019).

To the best of our knowledge, all searching problems studied to date are considering the searchers to have the same characteristics. More precisely, the searchers may have different ‘identities’ which allows them to differentiate their actions but their properties like speed, radius of capture or interactions with the fugitive are identical. However, there exist pursuit-evasion games in which some additional device (like a sensor or a trap) is used by the searchers [7, 8, 9, 36]. In this work we introduce a searching problem in which searchers are different: each searcher has access only to some part of the graph. More precisely, there are several *types* of searchers, and for each edge e in the graph, only one type of searchers can slide along e . We motivate this type of search twofold. First, referring to some applications of graph searching problems in the field of robotics, one can imagine scenarios in which the robots that should physically move around the environment to execute a search strategy may not be all the same. Thus some robots, for various reasons, may not have access to the entire search space. Our second motivation is an attempt to understand the concept of monotonicity in graph searching. In general, the graph searching theory lacks of tools for analyzing search strategies that are not monotone, where a famous example is the question whether the connected search problem belongs to NP [2]. (In a connected search we require that at each point of the strategy the subgraph that is guaranteed not to contain the fugitive is connected; for a formal definition see Section 2.1.) In the latter, the simplest examples that show that recontamination may be beneficial for some graphs are quite complicated [38]. The variant of searching that we introduce has an interesting property: it is possible to construct relatively simple examples of graphs in which multiple recontaminations are required to search the graph with the minimum number of searchers. Moreover, it is interesting that this property holds even for trees.

1.1 Related work

In this work we adopt two models of graph searching to our purposes. Those models are the classical *edge search* [33, 34], which is historically the first model studied, and its connected variant introduced in [3]. As an optimization criterion we consider minimization of the number of searchers a strategy uses.

The edge search problem is known to be monotone [5, 28] but the connected search is not [38]. See also [22] for a more unified approach for proving monotonicity for particular graph searching problems. Knowing that the connected search is not monotone, a natural question is what is the ‘price of monotonicity’, i.e., what is the ratio of the minimum number of searchers required in a monotone strategy and an arbitrary (possibly non-monotone) one? It follows that this ratio is a constant that tends to 2 [15]. We remark that if the searchers do not know the graph in advance and need to learn its structure during execution of their search strategy then this ratio is $\Omega(n/\log n)$ even for trees [26]. An example of recently introduced model of *exclusive graph searching* shows that internal edge search with additional restriction that at most one searcher can occupy a vertex behaves very differently than edge search. Namely, considerably more searchers are required for trees and exclusive graph searching is not monotone even in trees [6, 30]. Few other searching problems are known not to be monotone and we only provide references for further readings [12, 23, 38]. Also see [21] for a searching problem for which determining whether monotonicity holds turns out to be a challenging open problem.

Since we focus on trees in this work, we briefly survey a few known results for this class of graphs. An edge search strategy that minimizes the number of searchers can be computed in linear time for trees [31]. Connected search is monotone and can be computed efficiently for trees [2] as well. However, if one considers weighted trees (the weight of a vertex or edge indicate how many searchers are required to clean or prevent recontamination), then the problem turns out to be strongly NP-complete, both for edge search [32] and connected search [13]. On the other hand, due to [14, 15] both of these weighted problems have constant factor approximations. The class of trees usually turns out to be a very natural subclass to study for many graph searching problems — for some recent algorithmic and complexity examples see e.g. [1, 16, 18, 24]. See also [25] for an approximation algorithm for general graphs that performs by adopting optimal search strategies computed for spanning trees of the input graph.

We conclude by pointing to few works that use heterogeneous agents for solving different computational tasks, mostly in the area of mobile agent computing. These include modeling traffic flow [35], meeting [29] or rendezvous [17, 19, 20]. We also note that heterogeneity can be introduced by providing weights to mobile agents, where the meaning of the weight is specific to a particular problem to be solved [4, 10, 27], while in [11] authors consider patrolling by robots with distinct speeds and visibility ranges.

1.2 Our work — a short outline

We focus on studying monotonicity and computational complexity of our heterogeneous graph searching problem that we formally define in Section 2.1. We start by proving that the problem is not monotone in the class of trees (Section 3). Then in Section 4 we show that, also in trees, monotone search with heterogeneous searchers is NP-complete. In Section 5 we prove that the general, non-monotone, searching problem is NP-hard for trees.

Our investigations suggest that the essence of the problem difficulty is hidden in the properties of the availability areas of the searchers. For example, the problem becomes hard for trees if such areas are allowed to be disconnected. To formally argue that this is the case we give, in Section 6, a polynomial-time algorithm that finds an optimal search strategy for heterogeneous searchers in case when each color class induces a connected subtree. This result holds also for the connected version of the heterogeneous graph search problem.

Section 2 is concluded with Table 2.1 that points out the complexity and monotonicity differences between the classical and connected edge search with respect to our problem.

2 Preliminaries

In this work we consider simple edge-labeled graphs $G = (V(G), E(G), c)$, i.e., without loops or multiple edges, where $c: E(G) \rightarrow \{1, \dots, z\}$ is a function that assigns labels, called *colors*, to the edges of G . Then, if $c(\{u, v\}) = i$, $\{u, v\} \in E(G)$, then we also say that vertices u and v *have* color i . Note that vertices may have multiple colors, so by $c(v) := \{c(\{u, v\}) : \{u, v\} \in E(G)\}$ we will refer to the set of colors of a vertex $v \in V(G)$.

2.1 Problem formulation

We will start by recalling the classical *edge search* problem [33] and then we will formally introduce our adaptation of this problem to the case of heterogeneous searchers.

An (edge) *search strategy* \mathcal{S} for a simple graph $G = (V(G), E(G))$ is a sequence of moves $\mathcal{S} = (m_1, \dots, m_\ell)$. Each move m_i is one of the following actions:

- (M1) placing a searcher on a vertex,
- (M2) removing a searcher from a vertex,
- (M3) sliding a searcher present on a vertex u along an edge $\{u, v\}$ of G , which results in a searcher ending up on v .

We often write for brevity ‘move i ’ in place of ‘move m_i ’.

Furthermore, we recursively define for each $i \in \{0, \dots, \ell\}$ a set \mathcal{C}_i such that \mathcal{C}_i , $i > 0$, is the set of edges that are *clean* after the move m_i and \mathcal{C}_0 is the set of edges that are clean prior to the first move of \mathcal{S} . Initially, we set $\mathcal{C}_0 = \emptyset$. For $i > 0$ we compute \mathcal{C}_i in two steps. In the first step, let $\mathcal{C}'_i = \mathcal{C}_{i-1}$ for moves (M1) and (M2), and let $\mathcal{C}'_i = \mathcal{C}_{i-1} \cup \{\{u, v\}\}$ for a move (M3). In the second step compute \mathcal{R}_i to consists of all edges e in \mathcal{C}'_i such that there *exists* a path P in G such that no vertex of P is occupied by a searcher at the end of move m_i , one endpoint of P belongs to e and the other endpoint of P belong to an edge not in \mathcal{C}_{i-1} .¹ We stress out that it is enough that only one such path exists, and in particular, if a contaminated edge is adjacent to a clean edge e , then e becomes contaminated when their common vertex v is not occupied by a searcher. In such case, P consists of the vertex v only. Then, set $\mathcal{C}_i = \mathcal{C}'_i \setminus \mathcal{R}_i$. If $\mathcal{R}_i \neq \emptyset$, then we say that the edges in \mathcal{R}_i become *recontaminated* (or that *recontamination occurs in \mathcal{S}* if it is not important which edges are involved). If l_e is the number of times the edge e becomes recontaminated during a search strategy, then the value $\sum_{e \in E(G)} l_e$ is referred to as the number of *unit recontaminations*. Finally, we define $\mathcal{D}_i = E(G) \setminus \mathcal{C}_i$ to be the set of edges that are

¹We point out that another way of computing the set \mathcal{C}_i is possible. Namely, start again with the same set \mathcal{C}'_i . Then, check if the following condition holds: there exists an edge e in \mathcal{C}'_i that is adjacent to an edge not in \mathcal{C}'_i and their common vertex is not occupied by a searcher. In such case, remove e from \mathcal{C}'_i . Keep repeating such an edge removal from \mathcal{C}'_i until there is no such edge e . Then, set $\mathcal{C}_i = \mathcal{C}'_i$ and $\mathcal{R}_i = E(G) \setminus \mathcal{C}'_i$.

contaminated at the end of move m_i , $i > 0$, where again \mathcal{D}_0 refers to the state prior to the first move. Note that $\mathcal{D}_0 = E(G)$. We require from a search strategy that $\mathcal{C}_\ell = E(G)$.

Denote by $V(m_i)$ the vertices occupied by searchers at the end of move m_i . We write $|\mathcal{S}|$ to denote the number of searchers used by \mathcal{S} understood as the minimum number k such that at most k searchers are present on the graph in each move. Then, the *search number of G* is

$$s(G) = \min \{ |\mathcal{S}| \mid \mathcal{S} \text{ is a search strategy for } G \}.$$

If the graph induced by edges in \mathcal{C}_i is connected for each $i \in \{1, \dots, \ell\}$, then we say that \mathcal{S} is *connected*. We then recall the *connected search number of G* :

$$cs(G) = \min \{ |\mathcal{S}| \mid \mathcal{S} \text{ is a connected search strategy for } G \}.$$

We now adopt the above classical graph searching definitions to the searching problem we study in this work. For an edge-labeled graph $G = (V(G), E(G), c)$, a search strategy assigns to each of the k searchers used by a search strategy a color: the color of searcher j is denoted by $\tilde{c}(j)$. This is done prior to any move, and the assignment remains fixed for the rest of the strategy. Then again, a search strategy \mathcal{S} is a sequence of moves with the following constraints: in move (M1) that places a searcher j on a vertex v it holds $\tilde{c}(j) \in c(v)$; move (M2) has no additional constraints; in move (M3) that uses a searcher j for sliding along an edge $\{u, v\}$ it holds $\tilde{c}(j) = c(\{u, v\})$. Note that, in other words, the above constraints enforce the strategy to obey the requirement that at any given time a searcher may be present on a vertex of the same color and a searcher may only slide along an edge of the same color. To stress out that a search strategy uses searchers with color assignment \tilde{c} , we refer to as a *search \tilde{c} -strategy*. We write $\tilde{c}_\mathcal{S}(j)$ to refer to the number of searchers with color j in a search strategy \mathcal{S} .

Then we introduce the corresponding graph parameters $hs(G)$ and $hcs(G)$ called the *heterogeneous search number* and *heterogeneous connected search number* of G , where $hs(G)$ (respectively $hcs(G)$) is the minimum integer k such that there exists a (connected) search \tilde{c} -strategy for G that uses k searchers.

Whenever we write $s(G)$ or $cs(G)$ for an edge-labeled graph $G = (V, E, c)$ we refer to $s(G')$ and $cs(G')$, respectively, where $G' = (V, E)$ is isomorphic to G .

We say that a search strategy \mathcal{S} is *monotone* if no recontamination occurs in \mathcal{S} . Analogously, for the search numbers given above, we define *monotone*, *connected monotone*, *heterogeneous monotone* and *connected heterogeneous monotone* search numbers denoted by $ms(G)$, $mcs(G)$, $mhs(G)$ and $mhcs(G)$, respectively, to be the minimum number of searchers required by an appropriate monotone search strategy.

The decision versions of the combinatorial problems we study in this work are as follows:

Heterogeneous Graph Searching Problem (HGS)

Given an edge-labeled graph $G = (V(G), E(G), c)$ and an integer k , does it hold $hs(G) \leq k$?

Heterogeneous Connected Graph Searching Problem (HCGS)

Given an edge-labeled graph $G = (V(G), E(G), c)$ and an integer k , does it hold $hcs(G) \leq k$?

In the optimization versions of both problems an edge-labeled graph G is given as an input and the goal is to find the minimum integer k , a labeling \tilde{c} of k searchers and a (connected) search \tilde{c} -strategy for G .

2.2 Additional notation and remarks

For some nodes v in $V(G)$ we have $|c(v)| > 1$, such connecting nodes we will call *junctions*. Thus a node v is a junction if there exist two edges with different colors incident to v .

We define an *area* in G to be a maximal subgraph H of G such that for every two edges e, f of H , there exists a path P in H connecting an endpoint of e with an endpoint of f such that P contains no junctions. We further extend our notation to denote by $c(H)$ the color of all edges in area H . Note that two areas of the same color may share a junction. Let $Areas(G)$ denote all areas of G . Two areas are said to be *adjacent* if they include the same junction.

Fact 2.1. *If T is a tree and v is a junction that belongs to some area H in T , then v is a leaf (its degree is one) in H . \square*

	Monotone	Non-monotone	Complexity
edge search	arbitrary graphs [33, 34, 32, 31]		P for trees [31], NPC for weighted trees [32], NPC for arbitrary graphs [31]
connected edge search	trees [2, 3]	arbitrary graphs [38]	P for trees [2], NPC for weighted trees [13], NPH for arbitrary graphs [2]
HGS		trees [Theorem 1]	NPH for trees [Theorem 4]

Table 1: Monotonicity and complexity summary of our problems in comparison with the classical and connected edge search problems for trees and arbitrary graphs.

Fact 2.2. *If T is a tree, then any two different areas in T have at most one common node which is a junction.* \square

Lemma 2.1. *Given a tree $T = (V(T), E(T), c)$ and any area H in T , any search \tilde{c} -strategy for T uses at least $\mathfrak{s}(H)$ searchers of color $c(H)$.*

Proof. If there are less than $\mathfrak{s}(H)$ searchers of color $c(H)$, then the area H can not be cleaned, as searchers of other colors can only be placed on leafs of H . \square

We now use the above lemma to obtain a lower bound for the heterogeneous search number of a graph $G = (V(G), E(G), c : E(G) \rightarrow \{1, \dots, z\})$. Define

$$\beta(G) = \sum_{i=1}^z \max \{ \mathfrak{s}(H) \mid H \in \text{Areas}(G), c(H) = i \}.$$

Using Lemma 2.1 for each area we obtain the following:

Lemma 2.2. *For each tree T it holds $\mathfrak{hs}(T) \geq \beta(T)$.* \square

3 Lack of monotonicity

Restricting available strategies to monotone ones can lead to increase of heterogeneous search number, even in case of trees. We express this statement in form of the following main theorem of this section:

Theorem 1. *There exists a tree T such that $\mathfrak{mhs}(T) > \mathfrak{hs}(T)$.*

In order to prove this theorem we provide an example of a tree $T_l = (V, E, c)$, where $l \geq 3$ is an integer, which cannot be cleaned with $\beta(T_l)$ searchers using a monotone search strategy, but there exists a non-monotone strategy, provided below, which achieves this goal. Our construction is shown in Figure 1.

We first define three building blocks needed to obtain T_l , namely subtrees T'_1 , T'_2 and T''_l . We use three colors, i.e., $k \geq 3$. The construction of the tree T'_i , $i \in \{1, 2\}$, starts with a root vertex q_i , which has 3 further children connected by edges of color 1. Each child of q_i has 3 children connected by edges of color 2.

For the tree T''_l , $l \geq 3$, take vertices v_0, \dots, v_{l+1} that form a path with edges $e_x = \{v_x, v_{x+1}\}$, $x \in \{0, \dots, l\}$. We set $c(e_x) = x \bmod 3 + 1$. We attach one pendant edge with color $x \bmod 3 + 1$ and one with color $(x - 1) \bmod 3 + 1$ to each vertex v_x , $x \in \{1, \dots, l\}$. Next, we take a path P with four edges in which two internal edges are of color 2 and two remaining edges are of color 3. To finish the construction of T''_l , identify the middle vertex of P , incident to the two edges of color 2, with the vertex v_0 of the previously constructed subgraph.

We link two copies of T'_i , $i \in \{1, 2\}$, by identifying two endpoints of the path P with the roots q_1 and q_2 of T'_1 and T'_2 , respectively, obtaining the final tree T_l shown in Fig. 1.

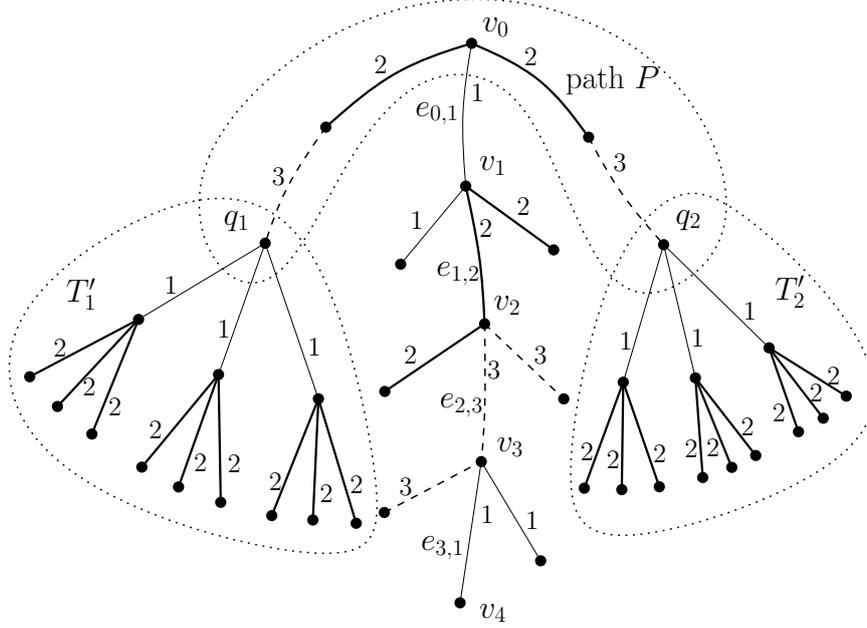


Figure 1: The construction of T_3 ($l = 3$) from the trees T'_1 , T'_2 and T'_3 . Regular, heavy and dashed edges have labels 1, 2 and 3, respectively.

Now, we are going to analyze a potential monotone search \tilde{c} -strategy \mathcal{S} using $\beta(T_l) = 3$ searchers. Thus, by Lemma 2.1, \mathcal{S} uses one searcher of each color. We define a notion of a *step* for $\mathcal{S} = (m_1, \dots, m_l)$ to refer to some particular moves of this strategy. We distinguish the following steps that will be used in the lemmas below:

1. step $t_i, i \in \{1, 2\}$, equals the minimum index j such that at the end of move m_j all searchers are placed on the vertices of T'_i (informally, this is the first move in which all searchers are present in T'_i);
2. step $t'_i, i \in \{1, 2\}$, is the maximum index j such that at the end of move m_j all searchers are placed on the vertices of T'_i (informally, this is the last move in which all searchers are present in T'_i);
3. steps t_3, t'_3 are, respectively, the minimum and maximum indices j such that at the end of move m_j all searchers are placed on the vertices in $V(P) \cup V(T''_3)$.

We skip a simple proof that all above steps are well defined, i.e., for any search strategy using 3 searchers for T each of the steps $t_i, t'_i, i \in \{1, 2, 3\}$, must occur (for trees T'_1 and T'_2 this immediately follows from $\mathfrak{s}(T'_i) = 3$ for $i \in \{1, 2\}$).

Lemma 3.1. *For each monotone \tilde{c} -search strategy \mathcal{S} for T_3 it holds: $t_1 \leq t'_1 < t_3 \leq t'_3 < t_2 \leq t'_2$ or $t_2 \leq t'_2 < t_3 \leq t'_3 < t_1 \leq t'_1$.*

Proof. Intuitively, we prove the lemma using the following argument: in the process of cleaning $T'_i, i \in \{1, 2\}$, all three searchers are required for some steps, and therefore a monotone strategy could not have partially cleaned T'_{3-i} or T''_3 prior to this point.

The arguments used to prove this lemma do not use colors, so atomic statements about search strategies for subgraphs can be analyzed using simple and well known results for edge search model. Furthermore, due to the symmetry of T , it is enough to analyze only the case when $t_1 < t_2$. Note that $t_i \leq t'_i, i \in \{1, 2, 3\}$, follows directly from the definition. The vertices $q_i, i \in \{1, 2\}$, have to be guarded at some point between move t_i and move t'_i because $\mathfrak{s}(T'_i) = 3$. Because each step $t_j, j \in \{1, 2, 3\}$, uses all searchers, it cannot be performed if a searcher preventing recontamination is required to stay outside of subtree related to the respective step. The subtrees T'_1 and T'_2 contain no common vertices, so $t_1 < t_2$ implies $t_2 > t'_1$, as stated in the lemma.

Suppose for a contradiction that $t_3 < t_i$ for each $i \in \{1, 2\}$. In move t_3 , since neither of moves t'_i has occurred, both subtrees T'_1, T'_2 contain contaminated edges. Moreover, some of the contaminated edges are incident to vertices q_i . Thus, any edge of T''_1 that is clean becomes recontaminated in the step $\min\{t_1, t_2\}$. Therefore, $t_1 < t_3$ as required.

Now we prove that $t'_1 < t_3$. Suppose for a contradiction that $t_1 < t_3 < t'_1$. Consider the move of index t'_1 . By $t_3 < t'_1$, T''_1 contains clean edges. By $t'_1 < t_2$, q_2 is incident to contaminated edges in T'_2 . Thus, there is a searcher outside of T'_1 which prevents recontamination of clean edges in T''_1 . Contradiction with the definition of t'_1 .

In move t_2 there are no spare searchers left to guard any contaminated area outside T'_2 which bypasses q_2 and could threaten recontamination of T'_1 , so all edges, including the ones in T''_1 , between those two trees should have been clean already. Therefore step t'_3 has to have already occurred, which allows us to conclude $t'_3 < t_2$. \square

Due to the symmetry of T_l , we consider further only the case $t_1 \leq t'_1 < t_3 \leq t'_3 < t_2 \leq t'_2$.

Lemma 3.2. *During each move of index $t \in [t'_1, t_2]$ there is a searcher on a vertex of P .*

Proof. By $t \geq t'_1$, q_1 is incident to some clean edges of T'_1 . By $t \leq t_2$, q_2 is incident to some contaminated edges from T'_2 . Hence there has to be a searcher on q_1, q_2 or a vertex of the path P between them to prevent recontamination. \square

Let $f_i, i \in \{1, \dots, l-1\}$, be the index of a move such that one of the edges incident to v_i is clean, one of the edges incident to v_i is being cleaned and all other edges incident to v_i are contaminated.

Notice that $s(T''_l) = 2$, and therefore an arbitrary search strategy \mathcal{S}' using two searchers to clean a subtree without colors that is isomorphic to T''_l follows one of these patterns: either the first searcher is placed, in some move of \mathcal{S}' , on v_1 and throughout the search strategy it moves from v_1 to v_{l-1} or the first searcher starts at v_{l-1} and moves from v_{l-1} to v_1 while \mathcal{S}' proceeds. If for each $i \in \{1, \dots, l-1\}$ the edge $\{v_{i-1}, v_i\}$ becomes clean prior to the edge $\{v_i, v_{i+1}\}$ — we say that such \mathcal{S}' cleans T''_l from v_1 to v_{l-1} and if the edge $\{v_{i-1}, v_i\}$ becomes clean after $\{v_i, v_{i+1}\}$ — we say that such \mathcal{S} cleans T''_l from v_{l-1} to v_1 .

Lemma 3.3. *Each move of index $f_i, i \in \{1, \dots, l-1\}$, is well defined. Either $f_1 < f_2 < \dots < f_{l-2} < f_{l-1}$ or $f_{l-1} < f_{l-2} < \dots < f_2 < f_1$.*

Proof. Consider a move of index f which belongs to $[t_3, t'_3]$ in a search strategy \mathcal{S} . By Lemma 3.1 and Lemma 3.2, a searcher is present on a vertex of P in the move of index f . Hence, only two searchers can be in T''_l in the move f , so \mathcal{S} cleans T''_l from v_1 to v_{l-1} or cleans T''_l from v_{l-1} to v_1 . Note that during an execution of such a strategy there occur moves which satisfy the definition of f_i , and therefore there exists well defined f_i . When \mathcal{S} cleans T''_l from v_0 to v_l , then $f_1 < f_2 < \dots < f_{l-2} < f_{l-1}$ is satisfied and when \mathcal{S} cleans T''_l from v_l to v_0 , then $f_{l-1} < f_{l-2} < \dots < f_2 < f_1$ is satisfied. \square

Lemma 3.4. *There exists no monotone search \tilde{c} -strategy that uses 3 searchers to clean T_l when $l \geq 7$.*

Proof. We use the following intuition in the proof: whenever a search strategy tries to clean the path composed of the vertices v_0, \dots, v_{l+1} , together with the corresponding incident edges, then it periodically needs searchers of all three colors on this path. While doing this, different vertices of the path P need to be guarded. More precisely, when the search moves along the former path, it needs to move along P as well. Due to the fact that l is large enough, the path P is not long enough to avoid recontamination.

The vertex $v_i, i \in \{1, \dots, l-1\}$, is incident to edges of colors $i \bmod 3+1$ and $(i-1) \bmod 3+1$, and therefore each move f_i uses both searchers of colors $i \bmod 3+1$ and $(i-1) \bmod 3+1$. By Lemma 3.2, the third searcher, which is of color $(i-2) \bmod 3+1$, stays on P .

Consider a sequence $f_6 < f_5 < \dots < f_2 < f_1$. Note that it implies that T''_3 is cleaned from v_{l-1} to v_1 . Let us show that it is impossible to place a searcher on the vertices of P such that no recontamination occurs in each $f_i, i \in \{1, \dots, 6\}$.

Consider the move of index f_6 , where searchers of colors 1 and 3 are in T''_l and 2 is on P . Before move f_6 an edge incident to v_6 is clean (by definition of f_6). No edge incident to v_1 is clean and, by Lemma 3.1, T''_j has a clean edge, $j \in \{1, 2\}$. In order to prevent recontamination of T''_j , the searcher is present on P , particularly on a vertex of the path from q_j to v_0 . It cannot be the vertex q_j , because $2 \notin c(q_j)$, so the edge of color 3 incident to q_j is clean, and the searcher is on one of the remaining two

vertices. Consider the move of index f_5 , in which the searcher of color 1 is on a vertex v of P . The vertex between q_j and v_0 cannot be occupied, due to its colors, and occupying q_j would cause recontamination — only the vertex v_0 is available, $v = v_0$. Consider the move of index f_4 . The vertex v_0 cannot be occupied, due to its colors. The edge e_0 cannot be clean before e_4 is clean, because T_l'' is cleaned from v_{l-1} to v_1 . Therefore, the searcher on v_0 cannot be moved towards q_2 . Monotone strategy fails.

The argument is analogical for a sequence $f_1 < f_2 < \dots < f_5 < f_6$. By Lemma 3.3, T_l'' is cleaned either from v_1 to v_{l-1} or the other way, which implies that considering the two above cases completes the proof. \square

Lemma 3.5. *There exists a non-monotone \tilde{c} -strategy \mathcal{S} that cleans T_l using three searchers for each $l \geq 3$.*

Proof. The strategy we describe will use one searcher for each of the three colors. The strategy first cleans the subtree T_1' (we skip an easy description how this can be done) and finishes by cleaning the path connecting q_0 with v_0 . Denote the vertex on the path from v_0 to q_2 as v .

Now we describe how the strategy cleans T_l'' from v_1 to v_{l-1} . For each $i \in \{1, \dots, l\}$, the vertex v_i is incident to edges of colors $i \bmod 3 + 1$ and $(i - 1) \bmod 3 + 1$ therefore each move f_i uses both searchers of colors $i \bmod 3 + 1$ and $(i - 1) \bmod 3 + 1$. By Lemma 3.2, the third searcher which is of color $(i - 2) \bmod 3 + 1$, stays on P . Informally, while progressing along T_l'' , the strategy makes recontaminations within the path P .

We will define j -progress, $j \in \{1, \dots, l - 1\}$, as a sequence of consecutive moves which clean edges of colors in $c(v_j)$ in T_l'' and contains the move of index f_j . Similarly, we introduce i -reconfig(u), $i \in \{1, 2, 3\}$, as a minimal sequence of consecutive moves, such that there is a searcher on some vertex u of P in the first move of i -reconfig(u) and the searcher of color i is present on P in the last move of i -reconfig(u). Let u_b be the occupied vertex of P after the last move of b -th i -reconfig(u) in \mathcal{S} . Additionally let $u_0 = v_0$. Clean T_l'' by iterating for each $j \in \{1, \dots, l - 1\}$ (in this order) the following: a -reconfig(u_{j-1}), followed by j -progress, where $a = (j - 2) \bmod 3 + 1$.

Because determining moves in j -progress is straightforward, as they correspond to those in monotone \tilde{c} -strategy when $f_1 < f_2 < \dots < f_{l-2} < f_{l-1}$, we focus on describing i -reconfig(u) for each $i \in \{1, 2, 3\}$. 1 -reconfig(u_0) consists of a sliding move from v_0 to v and a move which places the searcher of color 3 on $u_1 = v$. 2 -reconfig(u_1) consists of a sliding move from v to v_0 , which causes recontamination, and a move which places the searcher of color 1 on $u_2 = v_0$. 3 -reconfig(u_2) does not contain any sliding moves and places the searcher of color 2 on $u_3 = v_0$. Because $a = (j - 2) \bmod 3 + 1$ and $u_{j-1} = u_{j+2}$ a -reconfig(u_j) is identical to $a + 3$ -reconfig(u_{j+3}), thus we can describe a strategy which cleans T_l'' for any given l .

When T_l'' is clean, the vertex v_0 is connected to a clean edge and the remaining edges of path P can be searched without further recontaminations. The strategy cleans subtree T_2' in the same way as a monotone one.

Note that the proposed strategy requires new recontamination whenever a sequence of f_i of length 3 repeats itself. Thus, this \tilde{c} -strategy cleaning T_l has $\Omega(l)$ unit recontaminations. Note that the size of the tree T_l is $\Theta(l)$. \square

Lemma 3.5 provides a non-monotone search \tilde{c} -strategy which succeeds with fewer searcher than it is possible for a monotone one, as shown in lemma 3.4, which proves Theorem 1.

Theorem 2. *There exist trees such that each search \tilde{c} -strategy that uses the minimum number of searchers has $\Omega(n^2)$ unit recontaminations.*

Proof. As a proof we use a tree H_l obtained through a modification of the tree T_l . In order to construct H_l , we replace each edge on the path P with a path P_m containing m vertices, where each edge between them is in the same color as the replaced edge in T_l . Clearly $\text{hs}(H_l) = \text{hs}(T_l)$. Note that we can adjust the number of vertices in T_l'' and P_m of H_l independently of each other. While the total number of vertices is $n = \Theta(m + l)$, we take $m = \Theta(n)$, $l = \Theta(n)$ in H_l .

In order to clean H_l , we employ the strategy provided in theorem 3.5 adjusted in such a way, that any sliding moves performed on edges of P are replaced by $O(m)$ sliding moves on the corresponding paths of P_m . As shown previously, the number of times an edge of P in T_l , or path P_m in H_l , which contains $\Theta(m)$ elements, has to be recontaminated depends linearly on size of T_l'' . In the later case the \tilde{c} -strategy cleaning H_l has $\Omega(ml) = \Omega(n^2)$ unit recontaminations. \square

4 NP-hardness for trees

We show that the decision problem HGS is NP-complete for trees if we restrict available strategies to monotone ones. Formally, we prove that the following problem is NP-complete:

Monotone Heterogeneous Graph Searching Problem (MHGS)

Given an edge-labeled graph $G = (V(G), E(G), c)$ and an integer k , does it hold $\text{mhs}(G) \leq k$?

Thus, the rest of this section is devoted to a proof of the following theorem.

Theorem 3. *The problem MHGS is NP-complete in the class of trees.*

In order to prove the theorem, we conduct a polynomial-time reduction from Boolean Satisfiability Problem where each clause is limited to at most three literals (3-SAT). The input to 3-SAT consists of n variables x_1, \dots, x_n and a Boolean formula $C = C_1 \wedge C_2 \wedge \dots \wedge C_m$, with each clause of the form $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$, where the literal $l_{i,j}$ is a variable x_p or its negation, $\overline{x_p}$, $p \in \{1, \dots, n\}$. The answer to decision problem is YES if and only if there exist an assignment of Boolean values to the variables x_1, \dots, x_n such that the formula C is satisfied.

Given an input to 3-SAT, we construct a tree T_{SAT} which can be searched monotonously by the specified number of searchers if and only if the answer to 3-SAT is YES. We start by introducing the colors and, informally speaking, we associate them with respective parts of the input:

- color V_p , $p \in \{1, \dots, n\}$, represents the variable x_p ,
- color F_p (respectively T_p), $p \in \{1, \dots, n\}$, is used to express the fact that to x_p may be assigned the Boolean value false (true, respectively),
- color C_d , $d \in \{1, \dots, m\}$, is associated with the clause C_d .

We will also use an additional color to which we refer as R.

We denote the set of all above colors by \mathcal{Q} . Note that $|\mathcal{Q}| = 3n + m + 1$. In our reduction we set $k = |\mathcal{Q}| + 1 + m$ to be the number of searchers.

The construction of the tree starts with a path P of color R consisting of $l = 4n + 3m + 4 + 1$ vertices $v_i, i \in \{1, 2, \dots, l\}$. We add 2 pendant edges of color R to both v_1 and v_l . Define a subgraph H_z (see Figure 2(a)) for each color $z \in \mathcal{Q} \setminus \{R\}$: take a star of color R with three edges, attach an edge e of color z to a leaf of the star and then attach an edge e' of color R to e , so that the degree of each endpoint of e is two. For each $z \in \mathcal{Q} \setminus (\{R\} \cup \{C_1, \dots, C_m\})$ take a subgraph H_z and join it with P in such a way that the

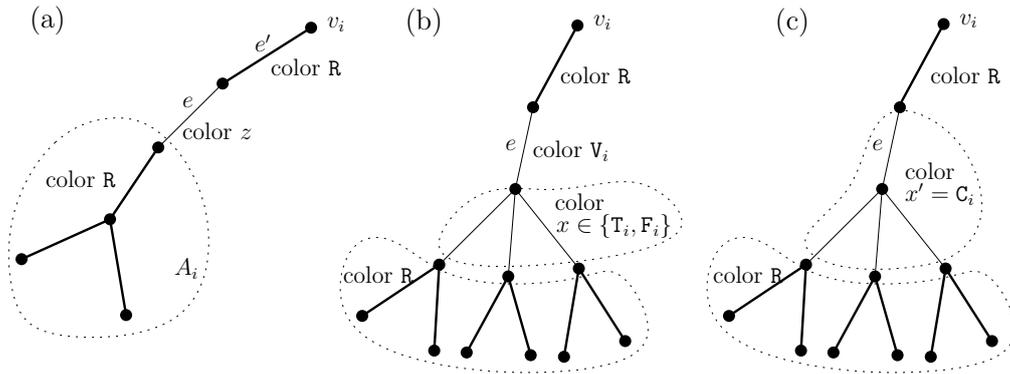


Figure 2: Construction of T : (a) the subgraph H_z ; (b) the subgraph L_x ; (c) the subgraph L'_x ,

endpoint of e' of degree one in H_z is identified with a different vertex in $\{v_2, \dots, v_a\}$, $a = 3n + 2m + 1$. For each $z \in \{C_1, \dots, C_m\}$ take two copies of H_z and identify each endpoint of e' of degree one in H_z with a different vertex in $\{v_2, \dots, v_a\}$, which has no endpoint of e' attached to it yet. The above attachments of the subgraphs H_z are performed in such a way that the degree of v_i is three for each $i \in \{2, \dots, a\}$ (see Figure 3). We note that, except for the requirement that no two subgraphs H_z are attached to the same v_i , there is no restriction as to which H_z is attached to which v_i . The star of color R in the subgraph H_z attached to the vertex v_i is denoted by A_i .

For each color x in $X = \{T_i, F_i \mid i \in \{1, \dots, n\}\}$ we define a subtree L_x (see Figure 2(b)). We start with a root having a single child and an edge of color R between them. Then we add an edge e of color V_i to this child, where i is selected so that it matches x which is either T_i or F_i . The leaf of e has three further children attached by edges of color x . We finish by attaching 2 edges of color R to each of the three previous children. For each color $x' \in X' = \{C_1, \dots, C_m\}$ construct a subtree $L'_{x'}$ (see Figure 2(c)) in the same shape but colored in a different way. The edges of color different than R in the construction of L_x are replaced by edges of color x' . We draw attention to the fact that $L'_{x'}$ contains an area of color x' that is a star with four edges. We attach to the path P five copies of subtree L_x for each $x \in X$ and five copies of $L'_{x'}$ for each $x' \in X'$ by unifying their roots with the vertex v_{a+1} of P (see Figure 3).

We attach five further copies of $L'_{x'}$ for each $x' \in X'$ by unifying their roots with the vertex v_{l-1} of P .

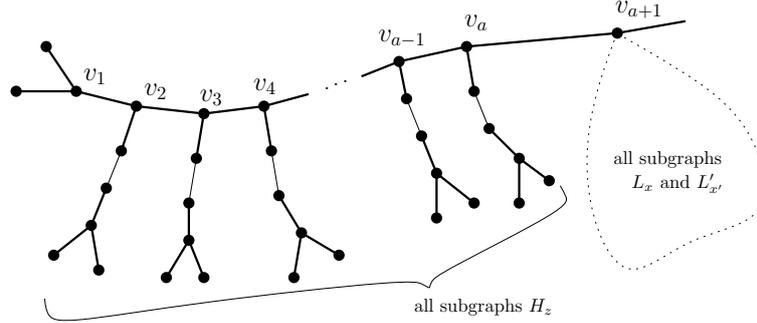


Figure 3: Construction of T : attachment of subgraphs H_z and the subgraphs L_x and $L'_{x'}$ to the path P

For each variable x_p we construct two subtrees, S_p and S_{-p} , in the following fashion (see Figure 4(a)): take a star of color R with three edges and attach an endpoint of a path with four edges to a leaf in this star of color R ; the consecutive colors of the path, starting from the endpoint at the star of color R are: V_p, R, F_p, R in S_{-p} and V_p, R, T_p, R in S_p . For each subtree S_p and S_{-p} , $p \in \{1, \dots, n\}$ attach the endpoint of its path of degree one to v_{a+1+p} . The star of color R in S_p attached to v_i is denoted by A_i

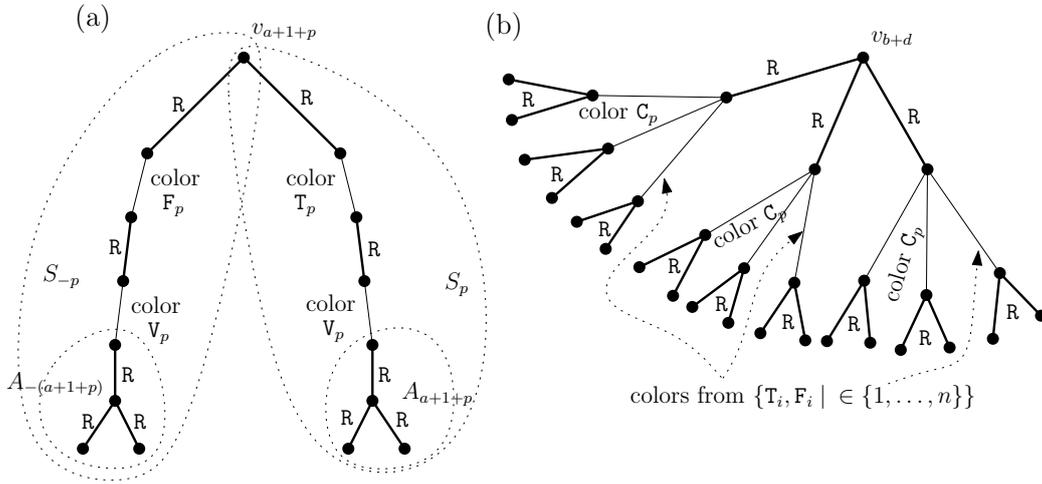


Figure 4: Construction of T : (a) the variable component constructed from S_{-p} and S_p ; (b) the clause component that corresponds to C_d .

and the one in S_{-p} by A_{-i} .

For each clause $C_d, d \in \{1, \dots, m\}$ we attach three subtrees $L_{d,j}, j \in \{1, 2, 3\}$, to the vertex v_{b+d} , where $b = 4n + 2m + 3$, one for each literal $l_{d,j}$ (see Figure 4(b)). Note that the maximal value of $b + d$ is $l - 2$. We construct $L_{d,j}$ by taking an edge e of color R and adding three edges to its endpoint: two of color C_i , and one either of color T_p if $l_{d,j} = x_p$ or of color F_p if $l_{d,j} = \bar{x}_p$. Add two children by the edge of color R to each of these three edges. Then attach the endpoint of degree one of the edge e in $L_{d,j}$

to v_{b+d} . We attach a single edge of color R to v_b . The tree obtained through this construction will be denoted by T_{SAT} .

The area of color R which contains the path P is denoted by A_0 . Notice that all areas A_i of color R have search number two, $s(A_i) = 2$. For a search strategy for T_{SAT} , we denote the index of the first move in which all searchers of color R are in the area A_i as *step* t_i and the index of the last such move as *step* t'_i , $i \in I = \{2, \dots, a\} \cup \{a+2, \dots, b-1\} \cup \{-(b-1), \dots, -(a+2)\} \cup \{0\}$. Let $R = \{a+2, \dots, b-1\} \cup \{-(b-1), \dots, -(a+2)\}$ and $L = I \setminus \{R \cup \{0\}\} = \{2, \dots, a\}$ be the two sets which cover all indices of areas $A_a : a \in I \setminus \{0\}$. Note that by definition $a+1 \notin L$ and $a+1 \notin R$, and the path from v_1 to v_{a+1} contains no vertex $v_j, j \in R$. Similarly, the path from v_{a+1} to v_l contains no vertex $v_i, i \in L$. Informally, we divide the indices in $I \setminus \{0\}$ into two sets: L to the left of v_{a+1} and R to the right.

Lemma 4.1 (Color assignment). *A search \tilde{c} -strategy using $k = 3n + 2m + 2$ searchers has to color them in the following fashion: one searcher for each color in $\{T_p, F_p, V_p \mid p \in \{1, \dots, n\}\}$ and two searchers for each color in $\{R\} \cup \{C_1, \dots, C_m\}$.*

Proof. We first compute the lower bound $\beta(T_{\text{SAT}})$. By Lemma 2.1, at least $3n$ searchers take colors F_p, T_p and $V_p, p \in \{1, \dots, n\}$. Recall that $L'_{x'}$ contains as a subgraph an area T' of color $x' \in \{C_1, \dots, C_m\}$ that is a star with three edges and hence $s(T') = 2$. Since there are m such subtrees $L'_{x'}$, $2m$ searchers receive colors C_1, \dots, C_m . The last two searchers have to be of color R in order to clean areas $A_i, i \in I$. Thus, we have shown that $\beta(T_{\text{SAT}}) \geq 3n + 2m + 2$ and this lower bound is met by the assignment of colors to searchers, as indicated in the lemma. Using Lemma 2.2 we complete the proof. \square

Lemma 4.2. *Let x_1, \dots, x_n and a Boolean formula $C = C_1 \wedge C_2 \dots \wedge C_m$ be an input to 3-SAT. If the answer to 3-SAT is YES, then there exists a search \tilde{c} -strategy using $2 + 3n + 2m$ searchers for T_{SAT} .*

Proof. We first note the main point as to how a Boolean assignment provides the corresponding search strategy. Whether a variable is true or false, this dictates which searcher, either of color F_p or T_p , is placed in the corresponding variable component. The vertices occupied by these searchers form a separator that disconnects A_0 from areas $A_i, i \in R$. The strategy cleans first the latter areas that are protected from recontamination. As a result, all $A_i, i \in R$, become clean. Then, A_0 is cleaned with the clause components along the way: here the fact that the initial Boolean assignment was satisfied ensures that searchers of appropriate colors are available to clean the subsequent clause components.

Suppose that a Boolean assignment to the variables satisfies C . The strategy is described as a sequence of instructions.

1. We start by placing a searcher of color dictated by the Boolean assignment in each variable component. For each S_p (respectively S_{-p}), place a searcher of color T_p (respectively F_p) on the vertex that is incident to the edge of color T_p and does not belong to A_0 if x_p is *false* (respectively *true*).
2. Then, clean A_{a+1+p} (respectively $A_{-(a+1+p)}$) and then the edges in S_p (respectively S_{-p}) that connect this star of color R to the vertex guarded by the searcher of color T_p (respectively F_p). Note that this cleaning uses two searchers of color R and the searcher of color V_p . Then, place the searcher of color V_p on the vertex that belongs to the edge of color V_p in S_{-p} (respectively S_p) and area $A_{-(a+1+p)}$ (respectively A_{a+1+p}). Clean $A_{-(a+1+p)}$ (respectively A_{a+1+p}).

By repeating the above for each index p , we in particular obtain that all areas $A_i, i \in R$ are clean. Note that if $x_p = \textit{false}$ (respectively $x_p = \textit{true}$), then the searcher of color T_p (respectively F_p) stays in S_p (respectively S_{-p}) and the searcher of color F_p (respectively T_p) is available. Let X denote the colors of available searchers among those in colors F_p and T_p .

3. Then we start cleaning A_0 from the vertex v_l and move towards v_b . Clean copies of $L''_d, d \in \{1, \dots, m\}$ attached to v_{l-1} . Consider each approached vertex $v_h, h \in \{b+1, b+2, \dots, l-2\}$, and its three subtrees $L_{d,i}, i \in \{1, 2, 3\}, d \in \{1, \dots, m\}$ separately. We denote the color different than R and C_d in $L_{d,i}$ as $x_{d,i}$. Because C is satisfied, at least one of the literals in each clause is true and for each $d \in \{1, \dots, m\}$ there always exists $L_{d,i}$ such that $x_{d,i}$ matches the color of the searcher not assigned to neither S_p nor S_{-p} , i.e., $x_{d,i} \in X$. Clean each such $L_{d,i}$ by using searchers of color C_d, R and $x_{d,i}$. Hence, at this point each $L_{d,i}$ for which the literal $l_{d,i}$ is satisfied in C is clean. Place the two searchers of color C_d in each remaining contaminated $L_{d,i}$ on vertex belonging to A_0 . Because

at least one subtree $L_{d,i}$ is clean for each $d \in \{1, \dots, m\}$, two searchers of color \mathbf{C}_d are sufficient. Then, continue cleaning A_0 towards the next v_h .

4. We now describe the moves of the search strategy performed once a vertex v_j , $j \in R$, is reached while cleaning A_0 .

Clean all remaining contaminated edges of S_p and S_{-p} rooted in v_j . Remove the searchers of colors $\mathbf{T}_p, \mathbf{F}_p, \mathbf{V}_p$ when they are no longer necessary.

5. Once v_{a+1} has been reached, clean all remaining contaminated subtrees $L_{d,i}$ (it follows directly from previous steps that searchers of appropriate colors are available) and perform moves as in colorless strategy with the addition of necessary switching of searchers on vertices with multiple colors. Repeat this strategy for each $L_x, x \in \{\mathbf{T}_i, \mathbf{F}_i \mid i \in \{1, \dots, n\}\}$, and $L'_{x'}, x' \in \{\mathbf{C}_1, \dots, \mathbf{C}_m\}$.
6. For each color $z \in \mathcal{Q} \setminus \{\mathbf{R}\}$ set a searcher of color z on the common vertex of the subtree H_z and A_0 . Then clean all edges incident to each vertex $v_i, i \in L$, which finishes cleaning A_0 . The finishing touches of our strategy are simple. Once A_0 is clean, the remaining contaminated parts of subtrees H_z , containing $A_i, i \in L$, can be searched in an arbitrary sequence.

□

Now we will give a series of lemmas that allow us to prove the other implication, namely that a successful monotone strategy implies a valid solution to 3-SAT. The next lemma says that between the first and last moves when all searchers of color \mathbf{R} are in an area A_i , no move having all searchers of color \mathbf{R} in a different area A_j is possible. The proof is due to a counting argument.

Lemma 4.3. *No step t_j can occur between any two steps t_i, t'_i :*

$$[t_i, t'_i] \cap [t_j, t'_j] = \emptyset, \quad i \neq j.$$

Proof. The proof is by contradiction. First note that if $t'_i = t_j$ for any $i \neq j$, that would imply that four searchers of color \mathbf{R} are present in a graph at once: two in A_i and two in A_j . Hence, we suppose for a contradiction that there exists $j \neq i$ such that $t_i < t_j < t'_i$ or $t_i < t'_j < t'_i$. Consider a step $t \in [t_i, t'_i]$. At least one searcher of color \mathbf{R} is in A_i because it is partially clean. If $t \in \{t_j, t'_j\}$, then there are two red searchers in A_j , which contradicts color composition imposed by Lemma 4.1. □

We say that a subtree T' is *guarded* by a searcher q on a vertex v if removal of q leads to recontamination of an edge in T' . Note that v does not have to belong to T' , or in other words, T' is any subtree of the entire subgraph that becomes recontaminated once the searcher q is removed. We extend our notation to say that T' is *guarded from T''* if T'' is contaminated and removal of q produces a path that is free of searchers and connects a node of T' with a node of T'' .

Informally, the next lemma states the following. Prior and after the moves that have all searchers of color \mathbf{R} on A_0 , there must be moves having all searchers of color \mathbf{R} on some $A_i, i \neq 0$. The argument is due to the fact that we do not have sufficiently many searchers, in total, to guard A_0 from all other A_i 's (or, conversely, all other A_i 's from A_0).

Lemma 4.4. *The step t_0 cannot be the first one and t'_0 cannot be the last one in a sequence of steps containing each $t_i, i \in I$, i.e., $\min\{t_i \mid i \in I\} < t_0 \leq t'_0 < \max\{t_i \mid i \in I\}$.*

Proof. Suppose for a contradiction that t_0 is the first step, i.e., $t_0 < t_i$, for each $i \in I \setminus \{0\}$. By Lemma 4.3, $t_i > t'_0$ for each $i \in I \setminus \{0\}$. Thus, each area A_i contains contaminated edges in step t_0 . In step t'_0 all edges of the path P are clean. Hence, P is guarded by at least one searcher from A_i for each $i \in I \setminus \{0\}$. A simple counting argument implies that two searchers of color $\mathbf{V}_p, p \in \{1, \dots, n\}$, are used — a contradiction. The second case, when t'_0 is the last step, can be argued analogously. □

We draw attention to the two ways of searching A_0 (to which we refer as a folklore). Since A_0 is a caterpillar, one can assume without loss of generality that it is cleaned by \mathcal{S} by the two searchers of color \mathbf{R} in the following way. Either, the first searcher of color \mathbf{R} is placed, in some move of \mathcal{S} , on v_1 and throughout the search strategy it moves along P from v_1 to v_l — we say that such \mathcal{S} *cleans P from v_1 to v_l* , or the first searcher starts at v_l and moves along P from v_l to v_1 while \mathcal{S} proceeds — we say that

such \mathcal{S} cleans P from v_l to v_1 . In both cases, the second searcher of color R is responsible for cleaning edges incident to v_i , $i \in \{1, \dots, l\}$, when the first searcher is on v_i .

We say that an edge search strategy \mathcal{S}' is a *reversal* of a search strategy \mathcal{S} that consists of l moves if it is constructed as follows: if the move i of \mathcal{S} places (respectively removes) a searcher on a node v , then the move $(l - i + 1)$ of \mathcal{S}' removes (respectively places) the searcher on v , and if the move i of \mathcal{S} slides a searcher from u to v , then the move $(l - i + 1)$ of \mathcal{S}' slides the searcher from v to u . It has been proved in [37] that if \mathcal{S} is a (monotone) edge search strategy, then \mathcal{S}' indeed is a (monotone) edge search strategy. We skip a proof (it is analogous to the one in [37]) that if \mathcal{S} is a monotone search \tilde{c} -strategy, then its reversal is also a monotone search \tilde{c} -strategy. This allows us to assume the following for the search strategy \mathcal{S} for T_{SAT} we consider in this section:

(*) \mathcal{S} cleans P from v_l to v_1 .

Let $R\text{-pr}(v_i)$, $i \in \{1, \dots, l\}$, be the index of the first move such that two searchers of color R are in v_i (either at the start or end of the move). Such moves are well defined because the degree of v_i is greater than two. Note that without loss of generality due to (*), $R\text{-pr}(v_l) = t_0$.

Observe that the removal of edges $\{v_a, v_{a+1}\}$ and $\{v_{a+1}, v_{a+2}\}$ from T_{SAT} gives three connected components and let T_{a+1} be the subtree of T_{SAT} that equals the connected component that contains v_{a+1} . For each subtree T' , let $\text{Clean}(T', t)$ ($\text{Cont}(T', t)$, respectively) denote the set of clean (contaminated, respectively) edges in T' immediately prior to the move t .

Intuitively, Lemma 4.5 says that when we reach the vertices v_a, v_{a+1}, v_{a+2} while moving along A_0 , then the tree T_{a+1} is constructed in such a way that while cleaning it there exists a move in which no searcher is used to guard any A_i with $i \in R$ or any clause component. The configurations of the colors of searchers for the guarding of A_i 's and the clause components are those in the family \mathcal{K} below.

Lemma 4.5. *Let*

$$\mathcal{K} = \{\{\mathbf{R}, \mathbf{C}_1\}, \dots, \{\mathbf{R}, \mathbf{C}_m\}, \{\mathbf{R}, \mathbf{V}_1, \mathbf{T}_1\}, \dots, \{\mathbf{R}, \mathbf{V}_n, \mathbf{T}_n\}, \{\mathbf{R}, \mathbf{V}_1, \mathbf{F}_1\}, \dots, \{\mathbf{R}, \mathbf{V}_n, \mathbf{F}_n\}\}$$

For each $K \in \mathcal{K}$, between moves $R\text{-pr}(v_{a+2})$ and $R\text{-pr}(v_a)$, there exists a move t_K which requires all searchers of colors from the set K to be in T_{a+1} .

Proof. An intuition explaining the proof is as follows. Recall that T_{a+1} consists of multiple copies of subtrees L_x and $L'_{x'}$. Arguments are the same for both L_x and $L'_{x'}$. We consider which edges of T_{a+1} are clean in the move $R\text{-pr}(v_{a+2})$: L_x it is either contaminated or contains a guarding searcher. Then we consider which edges of T_{a+1} are clean in the move $R\text{-pr}(v_a)$: in this case L_x it is either clean or contains a guarding searcher. We count how many guarded L_x 's can exist in the move $R\text{-pr}(v_a)$. A counting argument reveals that at least three L_x 's are clean in the move $R\text{-pr}(v_a)$. From all of the above, these 3 subtrees L_x must have been cleaned after $R\text{-pr}(v_{a+2})$. Among the moves of cleaning 3 subtrees L_x , a move t_K exists.

Consider what can be deduced about $\text{Clean}(T_{a+1}, R\text{-pr}(v_{a+2}))$ and $\text{Cont}(T_{a+1}, R\text{-pr}(v_{a+2}))$ from (*) and the definition of $R\text{-pr}(v_{a+2})$. Recall that by (*), the strategy we consider cleans P from v_l to v_1 . Hence, the edge $\{v_a, v_{a+1}\}$ is contaminated before move $R\text{-pr}(v_{a+2})$. Furthermore, the vertex v_{a+1} cannot be occupied by a searcher during move $R\text{-pr}(v_{a+2})$, because both searchers of color R are on the vertex v_{a+2} , by the definition of $R\text{-pr}(v_{a+2})$. Therefore, no subtree L_x , $x \in X = \{\mathbf{T}_i, \mathbf{F}_i \mid i \in \{1, \dots, n\}\}$, or $L'_{x'}$, $x' \in X' = \{\mathbf{C}_1, \dots, \mathbf{C}_m\}$, can be fully clean and unguarded in the move $R\text{-pr}(v_{a+2})$, because these subtrees are incident to v_{a+1} . For each subtree L_x , there are two possibilities: either $\text{Clean}(L_x, R\text{-pr}(v_{a+2})) = \emptyset$ or $\text{Clean}(L_x, R\text{-pr}(v_{a+2})) \neq \emptyset$ in which case L_x contains at least one guarded vertex. The same holds for any $L'_{x'}$.

Next consider what is known about $\text{Clean}(T_{a+1}, R\text{-pr}(v_a))$ and $\text{Cont}(T_{a+1}, R\text{-pr}(v_a))$. Because v_{a+1} is not guarded in the move $R\text{-pr}(v_a)$ as both searchers of color R are in vertex v_a , the vertex v_{a+1} is not incident to contaminated edges at this point. Otherwise all edges of P connecting v_{a+1} and v_l would be contaminated which contradicts (*). The spread of contamination at move $R\text{-pr}(v_a)$ from each L_x and $L'_{x'}$ through v_{a+1} can be prevented only in the following way: $\text{Clean}(P, R\text{-pr}(v_a))$ is guarded from the contaminated edges in subtrees L_x and $L'_{x'}$ and their copies, and all remaining subtrees in T_{a+1} are clean. Again there are two possibilities: either $\text{Clean}(L_x, R\text{-pr}(v_a)) = E(L_x)$ or if $\text{Clean}(L_x, R\text{-pr}(v_a)) \neq E(L_x)$, then L_x contains at least one guarded vertex. The same holds for any $L'_{x'}$.

By the above paragraphs, each subtree L_x and $L'_{x'}$ at some point between $R\text{-pr}(v_{a+2})$ and $R\text{-pr}(v_a)$ is either being guarded from or is fully searched. Suppose for a contradiction, that three out of five copies of a subtree L_x or $L'_{x'}$ contain a guarding searcher in the move $R\text{-pr}(v_a)$. A simple counting argument suffices to show that they have to be guarded on a common vertex: for L_x , there are two searchers of color \mathbf{R} , and they are placed on the vertex v_a , and one in each of the colors x and V_i where i is selected so that $x \in \{\mathbf{T}_i, \mathbf{F}_i\}$. Similarly for any $L'_{x'}$, there are only two searchers of color x' which can be placed in it. The only common vertex is v_{a+1} — contradiction with the definition of $R\text{-pr}(v_a)$.

Because we eliminated the option of guarding three subtrees L_x and $L'_{x'}$ at the move $R\text{-pr}(v_a)$, the only remaining possibility is that at least three of the subtrees L_x and $L'_{x'}$ are clean before the move $R\text{-pr}(v_a)$. Consider a move after which the first subtree has been cleaned. This subtree is guarded on v_{a+1} until all edges connected to v_{a+1} are clean, so in subsequent moves at least one of the copies of each L_x and $L'_{x'}$ is cleaned while v_{a+1} is guarded by a searcher of color \mathbf{R} . By construction, for each of the following sets: $B \in \mathcal{B} = \{\{\mathbf{R}, \mathbf{V}_1, \mathbf{T}_1\}, \dots, \{\mathbf{R}, \mathbf{V}_n, \mathbf{T}_n\}\}$ and $F \in \mathcal{F} = \{\{\mathbf{R}, \mathbf{V}_1, \mathbf{F}_1\}, \dots, \{\mathbf{R}, \mathbf{V}_n, \mathbf{F}_n\}\}$ there exist a subtree L_x requiring searchers in these colors. Note that $\mathfrak{s}(L_x) = 3$, so all of those searchers will be required simultaneously in at least one move, whose number is denoted by t_B or t_F respectively, when L_x is being searched. Similarly $L'_{x'}$ will require all searchers of colors $G \in \mathcal{G} = \{\{\mathbf{R}, \mathbf{C}_1\}, \dots, \{\mathbf{R}, \mathbf{C}_m\}\}$ to be present in T' in a single move, whose number is denoted by t_G . $\mathcal{B} \cup \mathcal{F} \cup \mathcal{G} = \mathcal{K}$, so t_K exists for each $K \in \mathcal{K}$. \square

Lemma 4.6. *Let*

$$\mathcal{K}'' = \{\{\mathbf{R}, \mathbf{C}_1\}, \dots, \{\mathbf{R}, \mathbf{C}_m\}\}$$

For each $K'' \in \mathcal{K}''$, between moves $R\text{-pr}(v_l)$ and $R\text{-pr}(v_{l-2})$, there exists a move t''_K which requires all searchers of colors from the set K'' to be in one of the subtrees $L''_{C_d}, d \in \{1, \dots, m\}$. \square

We skip the proof because it is analogous to the one of Lemma 4.5.

Let \mathcal{T} be the set of subtrees $H_z, z \in \mathcal{Q} \setminus \{\mathbf{R}\}$, and $S_p, S_{-p}, p \in \{1, \dots, n\}$. For $G \in \mathcal{T}$ let $\tau(G)$ be the index i such that G contains the vertex v_i .

We say in Lemma 4.7, informally, that we need to entirely clean all areas A_i with $i \in R$ prior to the part of the search strategy that uses all searchers of color \mathbf{R} on A_0 . The latter part is the one that cleans A_0 entirely.

Lemma 4.7. *The step t_0 is placed in the search sequence in the following way:*

$$t_j \leq t'_j < t_0 \leq t'_0$$

for each $j \in R$.

Proof. We first summarize the intuitions used in the proof. We start by using Lemma 4.3 and 4.4, which give us that A_0 is not the first nor the last area A_i cleaned. We define two sets of numbers (U^- and U^+ below) corresponding to indices of A_i 's whose cleaning happens before and after cleaning A_0 . During the moves t_0, \dots, t'_0 (i.e., those that clean A_0) the clean subgraph of A_0 has to be guarded from the contaminated A_i 's, and the cleaned A_i 's have to be guarded from the contaminated part of A_0 . Intuitively, once cleaning of A_0 extends past a vertex v_i to which a subtree containing A_i is attached, this A_i needs to be guarded if it's 'status' (being clean or contaminated) is different than that of the area A_0 . Consider a move, which we denote by l_{a+1} below in the proof, in which the vertex v_{a+1} divides the clean and contaminated parts of P . We analyze which A_j 's, $j \in R$, could have been left contaminated and which ones are guarded in the move l_{a+1} . We obtain that there exists a move $t_K, K \in \mathcal{K}$, described in the Lemma 4.5, which can be identified with l_{a+1} . There is not enough searchers to perform t_K while A_j is guarded — a contradiction.

Now we start the formal proof. Define

$$U^- = \{i \in I \mid t_i \leq t'_i < t_0\}$$

and

$$U^+ = \{i \in I \mid t'_0 < t_i \leq t'_i\}.$$

By Lemmas 4.3 and 4.4, $U^- \neq \emptyset$, $U^+ \neq \emptyset$ and $U^- \cup U^+ = I \setminus \{0\}$. Note that $U^- \cap U^+ = \emptyset$ because the strategy is monotone. Given this notation we restate the lemma as U^- contains all indices of steps $t_j, j \in R$, i.e., $R \subseteq U^-$.

Let $u^- \in U^-$ and $u^+ \in U^+$ be selected arbitrarily. Let $g_{|i|}, i \in I$ be the index of the first move in $[t_0, t'_0]$ such that $v_{|i|}$ is incident to a clean edge. There exists $G \in \mathcal{T}$ such that $\tau(G) = |u^+|$ and G has a contaminated edge between moves of numbers $g_{|u^+|}$ and t'_0 because $t'_0 < t_{u^+}$. Thus, $Clean(A_0, t), t \in [g_{|u^-|}, t'_0]$, is guarded from contaminated edges of G .

Let $g'_{|i|}, i \in I$, be the index of the last move in $[t_0, t'_0]$ such that $v_{|i|}$ is incident to a contaminated edge on P . There exists $G' \in \mathcal{T}$ such that $\tau(G') = |u^-|$ and G' has a clean edge between steps t_0 and $g_{|u^-|}$ because $t_{u^-} < t_0$. Thus, $Clean(G', t), t \in [t_0, g'_{|u^-|}]$, is guarded from contaminated edges of P .

Let l_{a+1} be an arbitrary move number such that the edge $\{v_a, v_{a+1}\}$ is contaminated and the edge $\{v_{a+1}, v_{a+2}\}$ is clean. By (*) such a move exists.

Suppose for a contradiction that $R \not\subseteq U^-$, which is equivalent to $R \cap U^+ \neq \emptyset$. During the move l_{a+1} the following subtrees are guarded:

$$Clean(A_0, l_{a+1}) \text{ from } A_j, j \in R \cap U^+, \text{ because } \forall_{j \in R \cap U^+} g_{|j|} < l_{a+1}$$

and

$$Clean(A_i, l_{a+1}) \text{ from } Cont(A_0, l_{a+1}), i \in L \cap U^- \text{ because } \forall_{i \in U^- \cap L} l_{a+1} < g'_{|i|}$$

By Lemma 4.5, there exists a move $t_K, K \in \mathcal{K}$, which requires all searchers of colors belonging to K to be present in T_{a+1} . Every edge incident to v_a cannot be clean before the last move which places two searchers of color R in v_a has occurred, therefore $R-pr(v_a) \leq g'_a$. Two searchers of color R cannot be placed in v_{a+2} before at least one edge incident to it is clean, therefore $g_{a+2} \leq R-pr(v_{a+2})$. This gives us $g_{a+2} \leq R-pr(v_{a+2}) < t_K < R-pr(v_a) \leq g'_a$, and with the fact that in the moves t_K and l_{a+1} the vertex v_{a+1} is occupied, allows us to conclude that for each t_K there exists l_{a+1} such that $t_K = l_{a+1}$.

Let $G \in \mathcal{T}$ be such that $\tau(G) > a + 1$. Recall that if $j \in R$ and $\tau(G) = |j|$, then G is isomorphic to some S_p or S_{-p} . Due to construction of S_p and S_{-p} , searcher used to guard $Clean(A_0, l_{a+1})$ from $Cont(G, l_{a+1})$ is in one of the following colors: R, V_p, T_p if $j > 0$ or R, V_p, F_p if $j < 0$. Let D_G denote a set of colors in G for each $G \in \mathcal{T}$. Note that for each D_G there exists $K \in \mathcal{K}$ such that $D_G \subseteq K$, and therefore there exists a move t_{D_G} such that all searchers in colors D_G are in T' . Consider a move $t_{D_G} = l_{a+1}$, which requires a searcher of one of the colors in D_G to be present in G , such that it contains an area $A_w, w \in R \cap U^+$. Because T' and $G \in \mathcal{T}$ contain no common vertices, such a move cannot exist — a contradiction with the Lemma 4.5. \square

The statement of the following lemma is more specific with respect to the previous one: in Lemma 4.8 we examine those moves in which each area $A_j, j \in R$, is clean and guarded from A_0 . Additionally we are concerned with the colors of searchers guarding these areas. More precisely, between the time when cleaning of A_0 starts and reaches v_b , all subgraphs $A_i, i \in R$, contain clean edges which are guarded from the contaminated edges of A_0 . Only searchers of colors R and either T_p or F_p can be used for the guarding.

Lemma 4.8. *In an arbitrary move $t \in [t_0, R-pr(v_b)]$, each subgraph $Clean(A_j, t)$ for each $j \in R$ is guarded from $Cont(A_0, t)$ by searchers on at least one of these two vertices: a vertex with colors $\{T_p, R\}$ in S_p or $\{F_p, R\}$ in S_{-p}*

Proof. Informally, we analyze the state of the strategy when A_0 is being cleaned but v_b has not been reached, i.e., in a move t from the lemma. In the moves t_0 and $R-pr(v_b)$ each of S_p and S_{-p} has some clean edges that need to be guarded. In the proof, we consider several cases as to which vertices can be guarded to protect those clean edges in moves t_0 and $R-pr(v_b)$. Then, we observe that, due to monotonicity, the set of clean edges when the two searchers of color R are on v_b could not be smaller than in any move prior to it. Thus, if both a vertex with colors $\{T_p, R\}$ and a vertex with colors $\{F_p, R\}$ need not be guarded between these moves, then a recontamination occurs in the move $R-pr(v_b)$ which leads to a contradiction.

From Lemma 4.7 we have $t_j \leq t'_j < t_0 \leq t'_0$ for each $j \in R$. All stars of color R in subtrees S_p and S_{-p} attached to vertices $v_{|j|}, j \in R$, contain clean edges before the move t_0 , so at the move t_0 each such star is guarded from contaminated edges in A_0 .

Consider the moves number t_0 and $R-pr(v_b)$. In these moves both searchers of color R are in A_0 , and $v_{|j|}$ are not attached to any clean edges, so guarding searchers are still necessary at move $R-pr(v_b)$. Recall that by the definition of $R, b \geq |j|$. Because searchers of color R are in A_0 , the vertices with the

following colors are occupied by searchers: F_p or V_p in S_{-p} and V_p or T_p in $S_p, p \in \{1, \dots, n\}$. Because only one searcher of color V_p is available, at least one other searcher is placed on vertex with either F_p or T_p color, denoted by c . In each S_p and S_{-p} there are only two such vertices separated by an edge of color c . It remains to be proven that at least one of them has to be guarded in an arbitrary move $t \in [t_0, R\text{-pr}(v_b)]$.

Assume for a contradiction that both vertices with color F_p and T_p are no longer guarded in a move $t \in [t_0, R\text{-pr}(v_b)]$, thus all edges incident to vertices with color c are clean. By monotonicity, all these edges are clean in the move $R\text{-pr}(v_b)$. Edges of color R incident to the vertex v_i such that $\tau(G) = i$ are contaminated. Since the two searchers of color R are in v_b in the move $R\text{-pr}(v_b)$, they are not in G . Thus, the searcher of color c is the only one that can be used for guarding $Clean(G, R\text{-pr}(v_b))$ from $Cont(A_0, R\text{-pr}(v_b))$. Since all edges incident to the vertex occupied by this searcher are clean, some recontamination occurs. \square

Lemma 4.9. *Let x_1, \dots, x_n and a Boolean formula $C = C_1 \wedge C_2 \dots \wedge C_m$ be an input to 3-SAT. If there exists a search strategy using $2 + 3n + 2m$ searchers for T_{SAT} , then the answer to 3-SAT is YES.*

Proof. The proof revolves around the configuration of searchers in the move $R\text{-pr}(v_b)$. We start by recalling the construction of subtrees based on clauses that must have been cleaned up to this point and the colors of searchers required to clean them. Then, we will use Lemma 4.8 to address the availability of these colors. We define a Boolean assignment as follows: x_p is true if and only if a searcher of color T_p does not guard the area A_{a+1+p} in the move $R\text{-pr}(v_b)$, otherwise x_p is false. Let $X \subset \{T_i, F_i \mid i \in \{1, \dots, n\}\}$ denote the colors of those searchers. By Lemma 4.8, a valid assignment will occur during execution of arbitrary successful search strategy using $2 + 3n + 2m$ searchers. We argue that some literal in each clause $C_d, d \in \{1, \dots, m\}$, is true under the above assignment.

By construction of T_{SAT} , $v_h, h \in \{b+1, b+2, \dots, l-1\}$, is the root of a subtree $L_{d,i}, i \in \{1, 2, 3\}$. $L_{d,i}$ contains an edge of color T_p if and only if the clause C_d contains a variable x_p , and it contains an edge of color F_p if and only if C_d contains a variable's negation, \bar{x}_p . Let us denote this color as $x_{d,i}$.

Consider the step t_0 . It is impossible for any $L_{d,i}$ to be completely clean because all edges incident to $v_h, h \in \{b+1, b+2, \dots, l-1\}$, are contaminated (there are not enough searchers of color R). Consider the move $R\text{-pr}(v_b) > t_0$. Each $L_{d,i}$ is completely clean or $Clean(A_0, R\text{-pr}(v_b))$ is guarded by searchers of colors $x_{d,i}$ and C_d because v_h is connected to clean edges and unguarded (there are not enough searchers of color R). There are only two searchers of color C_d , so for each d one of the subtrees $L_{d,i}$ has a searcher of other color or is clean. Because $s(L_{d,i}) = 3$, three out of five searchers in the following colors can be used: $R, x_{d,i}, C_d$. The only searcher which is present in $L_{d,i}$ at move $R\text{-pr}(v_b)$ has color $x_{d,i}$. Without loss of generality we assume that the strategy cleans a subtree if possible before guarding other subtrees rooted in the same vertex. Consider a move $m_{d,i}$ such that $s(L_{d,i})$ searchers are used in $L_{d,i}$. Due to the way the subtree is colored, a searcher in each color $R, x_{d,i}, C_d$ has to be used in order to clean it. After t_0 a searcher of color R guards clean part of A_0 , so only one searcher out of those five, namely the one of color C_d , can be present outside of $L_{d,i}$. $L_{d,i}$ can be fully cleaned only if searcher of color $x_{d,i}$ is available at this point during $[t_0, R\text{-pr}(v_b)]$, or all edges of color $x_{d,i}$ were clean in the move t_0 . Due to its color this searcher can not be used to replace any searcher of color $x_{d,i}$ outside of $L_{d,i}$.

Let us address what follows if an edge of color $x_{d,i}$ was clean in the move t_0 . By construction, it can be guarded from contaminated edges of P by a searcher of one of the following colors: $x_{d,i}, C_d, R$ in the moves of numbers from the interval $[R\text{-pr}(v_l), R\text{-pr}(v_{l-2})]$. By Lemma 4.6, there exists a move of number in this interval such that all searchers of colors C_d and R are not in $L_{d,i}$. Thus, in order to avoid recontamination, clean edges of color $x_{d,i}$ can be guarded only by a searcher of the same color.

Suppose for contradiction that no literal in a clause $C_i, i \in \{1, \dots, n\}$ is true and a search strategy for T_{SAT} exists. By Lemma 4.8, one of the searchers of color $x \in \{T_p, F_p\}$, or a searcher of color R is placed outside of $L_{d,i}$ during $[t_0, R\text{-pr}(v_b)]$. By the definition of a Boolean assignment $x \in X$. Additionally Lemma 4.8 guarantees that no searcher of color x is used during cleaning any $L_{d,i}$. If $x = x_{d,i}$ then $m_{d,i}$ can not be performed and $L_{d,i}$ is guarded in the move $R\text{-pr}(v_b)$. Because there are only two searchers of color C_d , in order for a strategy to exist at least one subtree $L_{d,1}, L_{d,2}, L_{d,3}$ for each d is cleaned before $R\text{-pr}(v_b)$, or all three have to be guarded, and for that to happen they have to contain edges in at least one color $x_{d,i} \neq x$ corresponding to a true $l_{d,i}$ in the clause C_d . \square

5 NP-hardness of non-monotone searching of trees

This section is devoted to proving the problem remains NP-hard when non-monotone search strategies are allowed:

Theorem 4. *The problem HGS is NP-hard in the class of trees.*

For the proof, we adapt the tree T_{SAT} described in the previous section. The modified tree is denoted by \tilde{T}_{SAT} and it is obtained by performing the following operations on the tree T_{SAT} . In order to preserve the familiar notation we denote each component of \tilde{T}_{SAT} analogous to its counterpart in T_{SAT} with an additional sign \sim above its designation.

We add $4n$ vertices to the path P in the following fashion. Replace the edges $\{v_{a+1}, v_{a+2}\}$ and $\{v_{a+1}, v_a\}$ with paths of color \mathbf{R} of length $2n$ each, denoted by \tilde{P}_R and \tilde{P}_L respectively. Enumerate the vertices of \tilde{P} in \tilde{T}_{SAT} as \tilde{v}_i in such a way that $\tilde{v}_1 = v_1$, $\tilde{v}_{\tilde{a}+1} = v_{a+1+2n}$, $\tilde{v}_{\tilde{b}} = v_{b+1+4n}$, $\tilde{v}_{\tilde{l}} = v_{l+4n}$. Note that this enumeration preserves the informal division of vertices into sets on the left and right of \tilde{v}_{a+1} , and $\tilde{R} = \{\tilde{a} + 2, \dots, \tilde{b} - 1\} \cup \{-(\tilde{b} - 1), \dots, -(\tilde{a} + 2)\}$ is defined accordingly.

We use $2n$ additional colors $O = \{\mathbf{0}_{1,1}, \dots, \mathbf{0}_{n,1}, \mathbf{0}_{1,2}, \dots, \mathbf{0}_{n,2}\}$. For each $o \in O$ create a tree H_o following the construction defined in the previous section and attach one to a unique vertex of the path \tilde{P}_L . We do the same for the path \tilde{P}_R so that $4n$ subtrees are created in total. Let $\tilde{H}_o(\tilde{R})$ denote a subtree containing an edge of color $o \in O$ attached to the vertex $\tilde{v}_i, i \in \tilde{R}$.

Next we modify the construction of each subtree $L_x, x \in \{\mathbf{T}_i, \mathbf{F}_i \mid i \in \{1, \dots, n\}\}$ rooted in v_{a+1} in the following way. Remove 2 leaves of color \mathbf{R} and attach 3 children by the edges of color $\mathbf{0}_{i,1}$ to each leaf. Then attach 3 children by the edges of color $\mathbf{0}_{i,2}$ to each of the new leaves. Finally attach 2 children by the edges of color \mathbf{R} to each of the lastly added leaves. Denote the modified L_x as \tilde{L}_x . Note that $s(\tilde{L}_x) = 5$ and \tilde{L}_x requires searchers of colors $x, \mathbf{R}, \mathbf{V}_i, \mathbf{0}_{i,1}, \mathbf{0}_{i,2}$ to be simultaneously present in some move in \tilde{L}_x in order to search it. In \tilde{T}_{SAT} , eleven copies of \tilde{L}_x are rooted in \tilde{v}_{a+1} in place of five copies of L_x rooted in v_{a+1} in the original T_{SAT} . Whenever an arbitrary copy can be chosen the notation of \tilde{L}_x is used, when the argument requires copies to be distinct they are denoted by $\tilde{L}_{x,i}, i \in \{1, \dots, 11\}$.

Define a star $O_p, p \in \{1, \dots, n\}$ with 3 leaves incident to edges of colors: $\mathbf{0}_{p,1}, \mathbf{0}_{p,1}$ and $\mathbf{0}_{p,2}$. We modify each subtree \tilde{S}_p in \tilde{T}_{SAT} corresponding to S_p constructed according to the definition in the previous section. Recall that each S_p and \tilde{S}_p (respectively S_{-p} and \tilde{S}_{-p}) contains an edge of color \mathbf{T}_p (\mathbf{F}_p respectively). Define a *plugin*(v, u) operation for vertices u and v of a tree as replacement of maximal subtree such that u and v are its leaves with a copy of O_p in such a way that u is identified with a leaf of color $\mathbf{0}_{p,1}$ and v is identified with a leaf of color $\mathbf{0}_{p,2}$. For each $\tilde{T} \in \{\tilde{S}_p, \tilde{S}_{-p} \mid p \in \{1, \dots, n\}\}$ denote the endpoint which belongs to \tilde{A}_0 of the edge of color \mathbf{T}_p or \mathbf{F}_p in \tilde{T} as u_1 , and the other endpoint of this edge as u_2 . Denote the endpoint which belongs to $\tilde{A}_j, j \in \tilde{R}$, of edge of color \mathbf{V}_p as u_3 , and the other endpoint of this edge as u_4 . Perform *plugin*(u_1, u_1), *plugin*(u_2, u_3) and *plugin*(u_4, u_4).

Informally speaking, the described modification prevents using recontamination to switch the searchers used as a basis for Boolean assignment without undoing all the progress made while cleaning subtrees corresponding to the clauses.

The following lemma follows directly from the lower bound $\beta(\tilde{T}_{\text{SAT}})$ and the proof is analogous to Lemma 4.1

Lemma 5.1 (Color assignment). *A \tilde{c} -strategy using $k = 5n + 2m + 2$ searchers has to color them in the following fashion: one searcher for each color in $\{\mathbf{T}_p, \mathbf{F}_p, \mathbf{V}_p, \mathbf{0}_{p,1}, \mathbf{0}_{p,2} \mid p \in \{1, \dots, n\}\}$ and two searchers for each color in $\{\mathbf{R}\} \cup \{\mathbf{C}_1, \dots, \mathbf{C}_m\}$. \square*

Lemma 5.2. *Let x_1, \dots, x_n and a Boolean formula $C = C_1 \wedge C_2 \dots \wedge C_m$ be an input to the 3-SAT. If the answer to 3-SAT is YES, then there exists a search strategy using $5n + 2m + 2$ searchers for \tilde{T}_{SAT} .*

Proof. We propose a modification to the monotone strategy described in Lemma 4.2. Note that the modified strategy is still monotone (we aim to show that recontamination does not help to search \tilde{T}_{SAT}). In the instruction 2 we clean $\tilde{A}_{-(\tilde{a}+2n+1+p)}$ instead of $A_{-(a+1+p)}$ and stars O_p instead of singular edges of color \mathbf{R} replaced by these stars during construction of \tilde{T}_{SAT} . No searcher of color either $\mathbf{0}_{p,1}$ or $\mathbf{0}_{p,2}$ has already been placed on \tilde{T}_{SAT} so it is always possible. We introduce an additional instruction 2' executed after the instruction number 2.

- 2'. For each $\tilde{H}_o(\tilde{R})$ place a searcher of color o on the vertex of color o and belongs to an \tilde{A}_i in $\tilde{H}_o(\tilde{R})$. Then, clean each A_i , where $\tilde{v}_i \in \tilde{P}_R$, and then the edge of color o . The searcher of color o stays in $\tilde{H}_o(\tilde{R})$.

In instruction 4 a searcher of color o is removed from $\tilde{H}_o(\tilde{R})$ when the entire $\tilde{H}_o(\tilde{R})$ becomes clean during cleaning of \tilde{A}_0 in order to ensure that \tilde{L}_x can be searched. \square

5.1 Preliminaries on non-monotone strategies for \tilde{T}_{SAT}

Let G' be a subgraph of G . We define a *successful attempt* $S\text{-Attempt}(G', i) = [t, t']$ as a maximal interval of numbers of moves such that for each $j \in [t, t']$, $\text{Cont}(G', j) \neq G'$, $\text{Cont}(G', j) \neq \emptyset$ and $\text{Clean}(G', t') = G'$, and i is the ordeal number of this attempt among other successful attempts on G' . Analogously define an *unsuccessful attempt* $U\text{-Attempt}(G', i) = [t, t']$ as a maximal interval of numbers of moves such that for each $j \in [t, t']$, $\text{Cont}(G', j) \neq G'$, $\text{Clean}(G', j) \neq G'$ and i is the ordeal number of this attempt among other unsuccessful attempts on G' .

By definition, at least one edge of G' is clean during an attempt on G' . We remove the prefix U or S whenever the success of the attempt is not important at the point of speaking.

Note that any search strategy which cleans a graph G contains the $S\text{-Attempt}(G', 1)$ for any subgraph G' , thus in order to show that cleaning a graph is impossible it suffices to prove that there exists a subgraph for which there can be no successful attempt. By strengthening the previous statement we obtain the following:

Observation 5.1. *If G' is a subgraph of G , then for each $S\text{-Attempt}(G, i)$ there exists $S\text{-Attempt}(G', j)$ such that $S\text{-Attempt}(G', j) \subseteq S\text{-Attempt}(G, i)$.*

We skip the proof of the following lemma as it follows from the folklore of the way to clean a caterpillar graph.

Lemma 5.3. *If in any move of $\text{Attempt}(\tilde{P}, i) = [t, t']$ both edges $\{\tilde{v}_1, \tilde{v}_2\}$ and $\{\tilde{v}_l, \tilde{v}_{l-1}\}$ are contaminated, then the attempt is unsuccessful.* \square

As a consequence we can be sure that a $S\text{-Attempt}(\tilde{P}, i)$ starts cleaning at either \tilde{v}_l or \tilde{v}_1 and ends at \tilde{v}_l or \tilde{v}_1 respectively. If it starts at \tilde{v}_l and if the edge $\{\tilde{v}_{j+1}, \tilde{v}_j\}$ is clean, then the next clean edge of \tilde{P} can be only $\{\tilde{v}_j, \tilde{v}_{j-1}\}$ or the next contaminated set of edges of \tilde{P} has to include all edges $\{\tilde{v}_{j+1}, \tilde{v}_j\}, \dots, \{\tilde{v}_{j+1+x}, \tilde{v}_{j+x}\}$ for some $j+x < l, j-1 > 0$ and no other edges of \tilde{P} . We say that in such attempt a strategy *cleans \tilde{P} from \tilde{v}_l to \tilde{v}_1* . Thanks to the result concerning reversal of strategies established in [37] a symmetrical case does not need to be considered. Thus, we establish an assumption about non-monotone strategies analogous to (*):

(**) \mathcal{S} cleans \tilde{P} in $S\text{-Attempt}(\tilde{P}, 1)$ from \tilde{v}_l to \tilde{v}_1 .

Denote the number of a move when two searchers of color R arrive on the vertex \tilde{v}_i of the path \tilde{P} in $S\text{-Attempt}(\tilde{P}, 1)$ as $R\text{-pr}(\tilde{v}_i, j)$ where j is the ordinal number of the move $R\text{-pr}(\tilde{v}_i, j)$ among other moves $R\text{-pr}(\tilde{v}_i, j)$ of index i . Specifically $R\text{-pr}(\tilde{v}_i, j+1)$ is the number of the first such move after the move $R\text{-pr}(\tilde{v}_i, j)$. Informally speaking, the j in the expression $R\text{-pr}(\tilde{v}_i, j)$ indicates how many times the vertex \tilde{v}_i was reached in the first successful attempt to clean \tilde{P} . Let $\tilde{P}_i, i \in \{1, \dots, l\}$, denote $\tilde{T}_{\text{SAT}}[\{\tilde{v}_i, \dots, \tilde{v}_i\}]$. Let $i(j) = k+j$ denote the ordeal number of the $S\text{-Attempt}(G, i(j))$ such that $i(0)$ is the ordeal number of the first $S\text{-Attempt}(G, k)$ such that $S\text{-Attempt}(G, k) \subseteq S\text{-Attempt}(\tilde{P}, 1)$. Whenever we speak of $S\text{-Attempt}(G, i(0))$, we are concerned with the first attempt to clean G within the first attempt which successfully cleaned \tilde{P} .

5.2 Some technical lemmas

In the next lemma we show that before the vertex \tilde{v}_a is reached in the first successful attempt to clean \tilde{P} , for each of the listed sets of colors there exists a move which requires searchers of those color to be present in \tilde{T}_{a+1} . Analogously to Lemma 4.5, these sets correspond to sets of colors of subtrees attached to the vertices of \tilde{P}_a .

Lemma 5.4. *Let*

$$\mathcal{K} = \{\{\mathbf{R}, \mathbf{C}_i\} \mid i \in \{1, \dots, m\}\} \cup \{\{\mathbf{R}, \mathbf{V}_n, \mathbf{O}_{n,1}, \mathbf{O}_{n,2}, x\} \mid x \in \{\mathbf{T}_1, \dots, \mathbf{T}_n, \mathbf{F}_1, \dots, \mathbf{F}_n\}\}.$$

For each $K \in \mathcal{K}$, in $S\text{-Attempt}(\tilde{P}_{\tilde{a}}, i(0))$, there exists a move $t_K \leq R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$ which requires all searchers of colors from the set K to be in $\tilde{T}_{\tilde{a}+1}$.

Proof. The proof is divided into three parts. First we argue that $\tilde{T}_{\tilde{a}+1}$ could not have been left clean before the move $R\text{-pr}(\tilde{v}_{\tilde{a}+2}, j)$. Then we argue that $\tilde{T}_{\tilde{a}+1}$ has to be clean before move $R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$. Finally we analyze the construction of $\tilde{T}_{\tilde{a}+1}$ to show that cleaning the subtrees of $\tilde{T}_{\tilde{a}+1}$ requires certain sets of searchers. These sets of searchers are listed as a family \mathcal{X} and \mathcal{Y} .

Let us consider moves performed only in the attempt $S\text{-Attempt}(\tilde{P}, 1)$ which by (***) cleans \tilde{P} from $\tilde{v}_{\tilde{i}}$ to \tilde{v}_1 . Choose the minimal j such that $R\text{-pr}(\tilde{v}_{\tilde{a}+2}, j) \in S\text{-Attempt}(\tilde{P}_{\tilde{a}}, i(0))$. Hence, the edge $\{\tilde{v}_{\tilde{a}}, \tilde{v}_{\tilde{a}+1}\}$ is not clean at move $R\text{-pr}(\tilde{v}_{\tilde{a}+2}, j)$ move. The subtree $\tilde{T}_{\tilde{a}+1}$ cannot be completely clean in the move $R\text{-pr}(\tilde{v}_{\tilde{a}+2}, j)$, because it contains the vertex $\tilde{v}_{\tilde{a}+1}$, which is unoccupied (by the definition of $R\text{-pr}(\tilde{v}_{\tilde{a}+2}, 1)$) and incident to the contaminated edge $\{\tilde{v}_{\tilde{a}+1}, \tilde{v}_{\tilde{a}}\}$ (by (***)).

On the other hand each copy of \tilde{L}_x has to be either completely clean or guarded in the move $R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$. Suppose otherwise for a contradiction, then the contamination spreads unobstructed through $\tilde{v}_{\tilde{a}+1}$, which cannot be occupied by a searcher during move $R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$, to $\tilde{v}_{\tilde{i}}$ and, by the Lemma 5.3, the attempt $S\text{-Attempt}(\tilde{P}, 1)$ fails contrary to its definition.

Because the two searchers of color \mathbf{R} are not in a non-leaf vertex of \tilde{L}_x in neither of the moves number $R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$ and $R\text{-pr}(\tilde{v}_{\tilde{a}+2}, j)$ at most four copies of \tilde{L}_x can be guarded at each of these moves. In total at most eight out of eleven copies of \tilde{L}_x can be cleaned only partially between these two moves. Which means that in the attempt $S\text{-Attempt}(\tilde{P}_{\tilde{a}}, i(0))$ there exists a $S\text{-Attempt}(\tilde{L}_{x,1} \cup \tilde{L}_{x,2} \cup \tilde{L}_{x,3}, k)$.

By construction, for each of set:

$$X \in \mathcal{X} = \{\{\mathbf{R}, \mathbf{V}_1, \mathbf{O}_{1,1}, \mathbf{O}_{1,2}, x\} \mid x \in \{\mathbf{T}_1, \dots, \mathbf{T}_n\} \cup \{\mathbf{F}_1, \dots, \mathbf{F}_n\}\}$$

there exists a subtree \tilde{L}_x requiring searchers of these colors. Note that $\mathfrak{s}(\tilde{L}_x) = 5$ and $\mathfrak{s}(\tilde{L}_{x,1} \cup \tilde{L}_{x,2} \cup \tilde{L}_{x,3}) = 6$, thus all searchers of colors contained in X will be present on some vertices of $\tilde{L}_{x,1} \cup \tilde{L}_{x,2} \cup \tilde{L}_{x,3}$ simultaneously, in at least one move, whose number is contained in $S\text{-Attempt}(\tilde{L}_{x,1} \cup \tilde{L}_{x,2} \cup \tilde{L}_{x,3}, k)$. Denote the number of the first such move in this attempt as t_X . Because we consider only moves whose numbers belong to $S\text{-Attempt}(\tilde{P}, 1)$, one searcher of color \mathbf{R} is present on \tilde{P} . In the move t_X this searcher occupies $\tilde{v}_{\tilde{a}}$, therefore $t_K \leq R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$.

The same argument can be repeated for any \tilde{L}'_y and the respective set from $Y \in \mathcal{Y} = \{\{\mathbf{R}, \mathbf{C}_i\} \mid i \in \{1, \dots, m\}\}$ to prove existence of analogously defined t_Y . $\mathcal{K} = \mathcal{X} \cup \mathcal{Y}$ finishes the proof. \square

We skip the proof of the following lemma because it is analogous to the one of Lemma 5.4.

Lemma 5.5. *Let*

$$\mathcal{K}'' = \{\{\mathbf{R}, \mathbf{C}_i\} \mid i \in \{1, \dots, m\}\}.$$

For each $K'' \in \mathcal{K}''$, in $S\text{-Attempt}(\tilde{P}_{\tilde{i}-2}, i(0))$, there exists a move $t_{K''}$ which requires all searchers of colors from the set K'' to be in one of the subtrees $\tilde{L}''_{\mathbf{C}_d}$, $d \in \{1, \dots, m\}$. \square

Let $\tilde{\mathcal{T}}$ be the set of all subtrees $\tilde{S}_p, \tilde{S}_{-p}, \tilde{H}_{0,p,1}(\tilde{R}), \tilde{H}_{0,p,2}(\tilde{R})$, $p \in \{1, \dots, n\}$. Recall that these subtrees are attached to the vertices \tilde{v}_j for $j \in \tilde{R}$.

Lemma 5.6. *In the move $R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$ all subtrees in $\tilde{\mathcal{T}}$ are clean.*

Proof. Each move t_K introduced in Lemma 5.4, where $K \in \mathcal{K}$, happens before the vertex v_a is reached in the first successful attempt to clean \tilde{P} . Each subtree in $\tilde{\mathcal{T}}$ contains only vertices of colors found in some $K \in \mathcal{K}$. Every subtree in $\tilde{\mathcal{T}}$ is connected to vertices which were cleaned before v_a . Because in the move t_K all searchers of colors present in some subtree of $\tilde{\mathcal{T}}$ are in T_{a+1} if this subtree of $\tilde{\mathcal{T}}$ contains a contaminated edge, then the attempt to clean \tilde{P} fails — since we analyze a successful attempt, a contradiction occurs. Finally we show that a recontamination of this subtree of $\tilde{\mathcal{T}}$ cannot happen prior to the move $R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$.

By Lemma 5.4, $R\text{-pr}(\tilde{v}_{\tilde{a}}, 1) \geq t_K$ for each $K \in \mathcal{K}$. By construction, for each subtree $\tilde{G} \in \tilde{\mathcal{T}}$ there exists $K \in \mathcal{K}$ such that the set of colors in vertices of \tilde{G} , denoted by $Q(\tilde{G})$, is a subset of K . Note that

any subtree in $\tilde{\mathcal{T}}$ is connected to the vertex $\tilde{v}_{\tilde{a}+1}$ only by a subpath of \tilde{P} , which may contain a subset of the following vertices $\{\tilde{v}_{\tilde{a}+1}, \dots, \tilde{v}_{\tilde{b}}\}$, and all these vertices are of color R.

Suppose for a contradiction that \tilde{G} contains a contaminated edge in the move t_K such that $Q(\tilde{G}) \subseteq K$. Because all searchers in colors $Q(\tilde{G})$ are in $\tilde{T}_{\tilde{a}+1}$ (one of color R is explicitly on the vertex $\tilde{v}_{\tilde{a}+1}$) and $\tilde{G} \cap \tilde{T}_{\tilde{a}+1} = \emptyset$, \tilde{G} contains no searchers in the move t_K . If it were to contain a contaminated edge at this point, then the contamination would have spread unobstructed along the path \tilde{P} from a vertex by which \tilde{G} is attached (which may be one of the following $\{\tilde{v}_{\tilde{a}+2}, \dots, \tilde{v}_{\tilde{b}}\}$) to the edge $\{\tilde{v}_{\tilde{i}}, \tilde{v}_{\tilde{i}-1}\}$. By Lemma 5.3, the attempt $S\text{-Attempt}(\tilde{P}, 1)$ fails, which contradicts its definition.

If \tilde{G} contained no contaminated edge nor searchers and was adjacent to the $Clean(\tilde{P}, t_K)$, then in the move t_K it was completely clean. By construction, recontamination may be introduced to \tilde{G} only through the vertex of \tilde{P} by which it is attached. Because $t_K \geq R\text{-pr}(\tilde{v}_{\tilde{a}+2}, j)$ there is always a searcher of color R guarding it from $Cont(\tilde{P}, t)$, $t \in [t_K, R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)]$, so it stays clean. \square

Lemma 5.7. *There is at least one clean edge in each \tilde{A}_r , $r \in \tilde{R}$, in each move of $S\text{-Attempt}(\tilde{P}, 1)$.*

Proof. Assume for the sake of a contradiction, that an area \tilde{A}_r is fully contaminated in the move $R\text{-pr}(\tilde{v}_{\tilde{i}}, 1)$. $\mathfrak{s}(\tilde{A}_r) = 2$ so it cannot be cleaned in $S\text{-Attempt}(\tilde{P}, 1)$, because at least one searcher of color R is in \tilde{P} . This contradicts Lemma 5.6, because the move $R\text{-pr}(\tilde{v}_{\tilde{a}}, 1)$, in which all of the subtrees in $\tilde{\mathcal{T}}$ are clean, belongs to $S\text{-Attempt}(\tilde{P}, 1)$. \square

Let \tilde{P}_b^+ denote $\tilde{T}_{\text{SAT}}[\{\tilde{v}_{\tilde{i}}, \dots, \tilde{v}_{\tilde{b}}, v\}]$ where v is a leaf incident $\tilde{v}_{\tilde{b}}$ which does not belong to \tilde{P} . Consider the attempt $S\text{-Attempt}(\tilde{P}_b^+, i(0))$. Note that $R\text{-pr}(\tilde{v}_{\tilde{i}}, 1) \in S\text{-Attempt}(\tilde{P}_b^+, i(0))$, $R\text{-pr}(\tilde{v}_{\tilde{b}}, 1) \in S\text{-Attempt}(\tilde{P}_b^+, i(0))$. Informally speaking, we define the first successful attempt to clean the subtree containing clause components, which are by construction connected to the vertices of the path \tilde{P}_b^+ .

Lemma 5.8. *There is at least one searcher in each $\tilde{G} \in \tilde{\mathcal{T}}$ guarding each $Clean(\tilde{A}_r, t)$ where $r \in \tilde{R}$, $t \in S\text{-Attempt}(\tilde{P}_b^+, i(0))$.*

Proof. Because the move t belongs to $S\text{-Attempt}(\tilde{P}, 1)$, by Lemma 5.7 there is at least one clean edge in \tilde{A}_r which has to be separated from $Cont(\tilde{P}, t)$. By the definition of t , at least one searcher of color R is on \tilde{v}_c , $c > \tilde{b}$ and edges $\{\{\tilde{v}_{\tilde{b}}, \tilde{v}_{\tilde{b}+1}\}, \dots, \{\tilde{v}_{\tilde{i}}, \tilde{v}_{\tilde{i}-1}\}\}$ are contaminated. Therefore there is at least one contaminated edge incident to each \tilde{G} , and a searcher guarding $Clean(\tilde{A}_r, R\text{-pr}(\tilde{v}_{\tilde{b}}, k))$ can only be placed in the subtree $\tilde{G} \in \tilde{\mathcal{T}}$ containing \tilde{A}_r . \square

Note that in \tilde{T}_{SAT} the areas \tilde{A}_r , $r \in \tilde{R}$, are subgraphs of $\tilde{H}_{o_p}(\tilde{R})$, \tilde{S}_p and \tilde{S}_{-p} (while only S_p and S_{-p} were included in T_{SAT}), hence the two separate lemmas below. The lemmas state that in the moves in which two searchers of color R are on the path P within the attempt to clean the clause components, only searchers of colors different than R can be used to guard clean edges of the subtrees $\tilde{H}_{o_p}(\tilde{R})$, \tilde{S}_p and \tilde{S}_{-p} .

Lemma 5.9. *For any i and j , such that $R\text{-pr}(\tilde{v}_i, j) \in S\text{-Attempt}(\tilde{P}_b^+, i(0))$, in a move $R\text{-pr}(\tilde{v}_i, j)$ there is a searcher of color $o_p \in \{0_{p,1}, 0_{p,2}\}$, $p \in \{1, \dots, n\}$, in $\tilde{H}_{o_p}(\tilde{R})$.*

Proof. By the definition of $R\text{-pr}(\tilde{v}_i, j)$, there can be no searcher of color R on any vertex of $\tilde{H}_{o_p}(\tilde{R})$. By Lemma 5.8, each of them contains a searcher, and by colors of vertices in $\tilde{H}_{o_p}(\tilde{R})$, it is of color o_p . \square

Note that Lemmas 5.9 and 5.10 speak of the same moves, therefore the pool of available searchers is shared between them.

Lemma 5.10. *For any i and j , such that $R\text{-pr}(\tilde{v}_i, j) \in S\text{-Attempt}(\tilde{P}_b^+, i(0))$, in a move $R\text{-pr}(\tilde{v}_i, j)$ there is a searcher of color T_p (respectively F_p) or V_p in \tilde{S}_p (\tilde{S}_{-p} respectively).*

Proof. By the definition of $R\text{-pr}(\tilde{v}_i, j)$, there can be no searcher of color R on any vertex of \tilde{S}_p . By Lemma 5.8, each of them contains a searcher, and by colors of vertices in \tilde{S}_p and Lemma 5.9, it is of color T_p or V_p . Proof for \tilde{S}_{-p} is analogical. \square

5.3 Adaptation to non-monotonicity — there is no going back

Because of a possibility of recontamination, the previous lemmas are insufficient to obtain a result analogous to that given by Lemma 4.8. In this section we find a configuration of searchers that cannot be used in \tilde{P}_b^+ in a successful attempt to clean \tilde{P}_b^+ , cf. Lemma 5.13.

Lemma 5.11. *At least one of searchers of color from the following set: $Q = \{0_{p,1}, 0_{p,2}, x_p\}, x_p \in \{T_p, F_p\}$, has to remain in each $\tilde{S}_p \cup \tilde{S}_{-p}$ in each move $t \in [R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, 1)] \subseteq S\text{-Attempt}(\tilde{P}_b^+, i(0))$.*

Proof. The proof revolves around analyzing colors of vertices in \tilde{S}_p and \tilde{S}_{-p} which can be used by guarding searchers in Lemma 5.8. A switch is a change in the guarding searchers of colors different than $0_{p,1}$ or $0_{p,2}$ in \tilde{S}_p . In order to make such a switch, we either have to clean O_p or recontaminate it. We exclude the possibility to clean any star O_p in the moves t listed in the lemma (see Observation 5.2). Thus, we have to consider recontamination of O_p . Then, we use Lemma 5.10 to establish that such a switch can occur once in \tilde{S}_p (or \tilde{S}_{-p} analogically). Finally we look at guarding searchers in both \tilde{S}_p and \tilde{S}_{-p} in the moves $R\text{-pr}(\tilde{v}_i, 1)$ and $R\text{-pr}(\tilde{v}_b, 1)$ and show that a switch can occur in either \tilde{S}_p or \tilde{S}_{-p} , but not both, between these moves.

Pick a vertex, denoted by $u_{p,t}$, such that $u_{p,t} \in V(\tilde{S}_p)$ ($u_{-p,t} \in V(\tilde{S}_{-p})$ respectively) and it is incident to a contaminated and a clean edge in the move $t \in S\text{-Attempt}(\tilde{P}_b^+, i(0))$. Let us focus only on $u_{p,t}$ as the approach is analogous for $u_{-p,t}$. By Lemma 5.8, this vertex exists.

Denote the set of colors other than $0_{p,1}, 0_{p,2}$ in the set of colors of $u_{p,t}$ as $c_{p,t}$, i.e., $c_{p,t} = c(u_{p,t}) \setminus \{0_{p,1}, 0_{p,2}\}$. Recall that \tilde{S}_p and \tilde{S}_{-p} contain copies of the star O_p . Note that if $c_{p,t} = \emptyset$ then $u_{p,t}$ is a central vertex of such a copy of O_p . Otherwise $|c_{p,t}| = 1$. Let $f^+(t)$ denote the number of the first move such that $f^+(t) \geq t$ and $c_{p,f^+(t)} \neq \emptyset$. Let $f^-(t)$ denote the number of the first move such that $f^-(t) \leq t$ and $c_{p,f^-(t)} \neq \emptyset$.

Observation 5.2. *By Lemma 5.8, no $S\text{-Attempt}(O_p, i) \subseteq [R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, 1)]$ exists.*

By construction, any maximal subtree \tilde{T} , such that \tilde{T} is a subgraph of \tilde{S}_p where $u_{p,f^-(t)}$ and $u_{p,f^+(t')}, t < t'$, are leaves and $c_{p,f^-(t)} \neq c_{p,f^+(t')}$, contains a copy of O_p , to which we refer further as O'_p . If $c_{p,h} = \emptyset$, then $h \in \text{Attempt}(O_p, i)$ and $f^-(h)$ corresponds to the beginning of this attempt ($f^+(h)$ corresponds to its end, respectively). Note that both $c_{p,R\text{-pr}(\tilde{v}_i, 1)} \neq \emptyset$ and $c_{p,R\text{-pr}(\tilde{v}_b, k)} \neq \emptyset$. By Observation 5.2, a pair $u_{p,f^-(t)}$ and $u_{p,f^+(t')}$, such that $t, t' \in [R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, k)]$, which satisfies $c_{p,f^-(t)} \neq c_{p,f^+(t')}$, exists only if $O'_p \subseteq \tilde{T}$ has already been clean in the move $f^-(t)$. Similar argument can be repeated for a pair $u_{p,t}$ and $u_{-p,t'}$ (i.e. vertices in \tilde{S}_p and \tilde{S}_{-p}) with the conclusion that a pair which satisfies the above constraints does not exist — there are no clean copies of O_p between them. Informally, we can switch a searcher of color in $c_{p,t} \neq \emptyset$ to a searcher of color in $c_{p,t'} \neq c_{p,t} \neq \emptyset$ only by causing recontamination, and we cannot use the first searcher again. Thus, there is a finite number of switches in $S\text{-Attempt}(\tilde{P}_b^+, i(0))$.

Note that by Lemma 5.10, $c_{p,R\text{-pr}(\tilde{v}_i, 1)}$ ($c_{-p,R\text{-pr}(\tilde{v}_i, 1)}$ respectively) contains either T_p (respectively F_p) or V_p . If $c_{p,t'} = \{R\}$ we contradict Lemma 5.10 in the next move $R\text{-pr}(\tilde{v}_i, j)$ after t' . Therefore, a set $c_{p,t}$ or $c_{p,t'}$ can contain only a one out of these two colors: T_p, V_p , or be empty. By the previous paragraph and the number of different colors, there exists at most one interval of numbers of moves $J = [f^-(j), f^+(j')]$ such that $c_{p,f^-(j)} = T_p$ and $c_{p,f^+(j')} = V_p$ or vice versa. Informally, we can switch the color of required searcher once. The same argument holds for \tilde{S}_{-p} and colors F_p, V_p . Denote the corresponding interval as L .

Because there is only one V_p searcher at least one of the following is true: $c_{p,R\text{-pr}(\tilde{v}_i, 1)} = \{T_p\}$ or $c_{-p,R\text{-pr}(\tilde{v}_i, 1)} = \{F_p\}$, thus at most one edge of color V_p in $\tilde{S}_p \cup \tilde{S}_{-p}$ is clean. For the same reason at most one of the following is true: $c_{p,R\text{-pr}(\tilde{v}_b, k)} = \{V_p\}$ or $c_{-p,R\text{-pr}(\tilde{v}_b, k)} = \{V_p\}$. Thus, in a single strategy at most one of the two intervals J and L exists. If J (L respectively) does not exist, then $c_{p,t} \in \{c_{p,R\text{-pr}(\tilde{v}_i, 1)}, \emptyset\}$ ($c_{-p,t} \in \{c_{-p,R\text{-pr}(\tilde{v}_i, 1)}, \emptyset\}$ respectively) for each move $t \in S\text{-Attempt}(\tilde{P}_b^+, i(0))$. By definition of $u_{p,t}$, only searchers of colors $0_{p,1}, 0_{p,2}$ and those in $c_{p,t}$ can stay in \tilde{S}_p in each move of $[R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, 1)]$. \square

Lemma 5.12. *If the searcher of color $0_{p,z}, z \in \{1, 2\}$ is not in in $\tilde{H}_{0_{p,z}}(\tilde{R})$ in a move $t \in S\text{-Attempt}(\tilde{P}_b^+, i(0))$, then a searcher of color R is in $\tilde{H}_{0_{p,z}}(\tilde{R})$.*

Proof. Because $t \in S\text{-Attempt}(\tilde{P}, 1)$ and by Lemma 5.7 at least one searcher has to remain in $\tilde{H}_{0_{p,z}}(\tilde{R})$. It can be of color R or $\tilde{H}_{0_{p,z}}(\tilde{R})$. \square

Lemma 5.13. *There exists a set of searchers of colors $\{\mathbf{R}, \mathbf{0}_{p,1}, \mathbf{0}_{p,2}, x_p \mid p \in \{1, \dots, n\}\}, x_p \in \{\mathbf{T}_p, \mathbf{F}_p\}$ such that all but one have to remain outside of \tilde{P}_b^+ in each move $t \in [R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, 1)] \subseteq S\text{-Attempt}(\tilde{P}_b^+, i(0))$.*

Proof. By Lemma 5.11, for each $p \in \{1, \dots, n\}$ at least one of searchers of color from the following set: $Q = \{\mathbf{0}_{p,1}, \mathbf{0}_{p,2}, x_p\}, x_p \in \{\mathbf{T}_p, \mathbf{F}_p\}$, has to remain in $\tilde{S}_p \cup \tilde{S}_{-p}$ in each move $t \in [R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, 1)]$. Let s denote such a searcher of color other than x_p . If s exists then by Lemma 5.12 a searcher of color R is in $\tilde{H}_{0_{p,z}}(\tilde{R})$. In a move $t \in [R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, 1)]$ at least one searcher of color R is on the path \tilde{P}_b so s is unique. Then, $(\tilde{S}_p \cup \tilde{S}_{-p}) \cap \tilde{P}_b^+ = \emptyset$ and $\tilde{H}_{0_{p,z}}(\tilde{R}) \cap \tilde{P}_b^+ = \emptyset$ finish the proof. \square

To informally summarize, we show that there exists a set of searchers of colors $\{\mathbf{R}, \mathbf{0}_{p,1}, \mathbf{0}_{p,2}, x_p \mid p \in \{1, \dots, n\}\}, x_p \in \{\mathbf{T}_p, \mathbf{F}_p\}$ of which at most one at a time can take part in cleaning of \tilde{P}_b^+ .

5.4 Conclusion

Lemma 5.14. *Let x_1, \dots, x_n and Boolean formula $C = C_1 \wedge C_2 \dots \wedge C_m$ be an input to the 3-SAT problem. If there exists a search strategy using $2 + 5n + 2m$ searchers for \tilde{T}_{SAT} , then the answer to 3-SAT problem is YES.*

Proof. The proof revolves around the configuration of searchers in the move $R\text{-pr}(\tilde{v}_b, i(0))$. We define a Boolean assignment as follows: x_p is true if and only if a searcher of color \mathbf{T}_p does *not* guard the area $A_{\tilde{a}+1+p}$ in the move $R\text{-pr}(\tilde{v}_b, i(0))$, otherwise x_p is false. Let $X \subset \{\mathbf{T}_i, \mathbf{F}_i \mid i \in \{1, \dots, n\}\}$ denote the colors of those searchers. By Lemma 5.13, a valid assignment will occur during execution of \tilde{c} -search strategy using $2 + 5n + 2m$ searchers. We omit the detailed proof in favor of an analogy to the proof of Lemma 4.9.

Let \tilde{T} denote the maximal subtree containing \tilde{v}_i such that \tilde{v}_b is this subtree's leaf. Note that \tilde{T} is isomorphic to its equivalent in T_{SAT} , and monotone strategies are a subset of strategies available in this version of the problem. We focus on proving that the configuration of searchers in the move $R\text{-pr}(\tilde{v}_b, 1)$ has properties analogous to those of the configuration in the move $R\text{-pr}(v_b)$ of a strategy for T_{SAT} . Regardless of the moves performed by searchers in a \tilde{c} -strategy if a color $x \notin X$, then an edge of color x in $E(\tilde{T})$ which was contaminated before the move $R\text{-pr}(\tilde{v}_i, 1)$ remains contaminated in the moves of numbers from the interval $[R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, 1)]$. Thus, configurations which do not correspond to a valid assignment cannot use the searchers of appropriate colors required to guard them and continue cleaning the tree.

All that remains to be addressed is the possibility of these edges being clean before the move $R\text{-pr}(\tilde{v}_i, 1)$ (recall that in the proof of Lemma 4.9 we used the notion of monotonicity to resolve this issue, here the argument has to be continued). If this was the case they would have to be guarded by at least one searcher in the moves of numbers from the interval $[R\text{-pr}(\tilde{v}_i, 1), R\text{-pr}(\tilde{v}_b, 1)]$. By Lemma 5.5, there exists a move whose number is in this interval such that all searchers of color R and C_d are not in $\tilde{L}_{d,1}, \tilde{L}_{d,2}, \tilde{L}_{d,3}$. Thus, by the colors of vertices of $\tilde{L}_{d,i}$ only the searcher of color x can prevent recontamination of an edge of color x and it cannot be used, by the definition of X . Furthermore, these edges stay contaminated in the move $R\text{-pr}(\tilde{v}_b, i(0))$.

We use only the positions of searchers in a move of a specific number, so we are interested in a result, not the process, of a partial cleaning of T_{SAT} and \tilde{T}_{SAT} . Therefore, most arguments from Lemma 4.9 can be applied to \tilde{T}_{SAT} . Recall the conclusion of the proof of Lemma 4.9. In order for a \tilde{c} -strategy for T_{SAT} to exist at least one subtree $L_{d,1}, L_{d,2}, L_{d,3}$ for each $d \in \{1, \dots, m\}$ is cleaned before the clean part of A_0 reaches v_b , or all three have to be guarded, and for that to happen they have to contain edges in at least one color corresponding to $l_{d,i}$ in clause C_d . The same is true for a \tilde{c} -strategy for \tilde{T}_{SAT} and its respective counterparts of T_{SAT} . \square

6 Polynomially tractable instances

If G is a tree then Lemma 2.2 gives us a lower bound of $\beta(G)$ on the number of searchers. In this section we will look for an upper bound assuming that there is exactly one connected component per color. With this assumption we show a constructive, polynomial-time algorithm both for HGS and HCGS.

Let (E_1, \dots, E_k) be the partition of edges of T so that E_i induces the area of color i in T . Observe that this partition induces a tree structure. More formally, consider a graph in which the set of vertices is $P_E = \{E_1, E_2, \dots, E_k\}$ and $\{E_i, E_j\}$ is an edge if and only if an edge in E_i and an edge in E_j share a common junction in T . Then, let \tilde{T} be the BFS spanning tree with the root E_1 in this graph. We write V_i to denote all vertices of the area with edge set E_i , $i \in \{1, \dots, z\}$.

Our strategy for cleaning T is recursive, starting with the root. The following procedure requires that when it is called, the area that corresponds to the parent of E_i in \tilde{T} has been cleaned, and if $i \neq 1$ (i.e., E_i is not the root of \tilde{T}), then assuming that E_j is the parent of E_i in \tilde{T} , a searcher of color j is present on the junction in $V_i \cap V_j$. With this assumption, the procedure recursively cleans the subtree of \tilde{T} rooted in E_i .

procedure CLEAN(labeled tree T , E_i) \triangleright Clean the subtree of T that corresponds to the subtree of \tilde{T} rooted in E_i

1. For each E_j such that E_j is a child of E_i in \tilde{T} place a searcher of color j on the junction $v \in V_j \cap V_i$.
2. Clean the area of color i using $s(T[V_i])$ searchers. Remove all searchers of color i from vertices in V_i .
3. For each child E_j of E_i in \tilde{T} :
 - (a) place a searcher of color i on the junction $v \in V_j \cap V_i$,
 - (b) remove the searcher of color j from the vertex v ,
 - (c) call *Clean* recursively with input T and E_j ,
 - (d) remove the searcher of color i from the vertex v .

end procedure

Lemma 6.1. *For a given tree $G = (V(G), E(G), c)$, procedure $Clean(G, E_1)$ cleans G using $\beta(G)$ searchers.*

Proof. First, observe that the number of searchers used during the execution of procedure Clean is exactly as specified. Indeed, to clean each of the $T[V_i]$ we use $s(T[V_i])$ searchers of color i and at most one searcher of other colors.

Note that moves (M2) do not cause recontamination. Indeed, the move defined in step 3b of procedure Clean removes a searcher from the node on which another searcher is present, while the move 3d is performed on node v when both subtree and the parent subtree of v are cleaned. This gives the correctness of search strategy produced by procedure Clean. \square

We also immediately obtain.

Lemma 6.2. *If all the strategies used in step 2 of procedure Clean to clean a subtree $T[V_i]$ are monotone, then the resulting \tilde{c} -strategy for G is also monotone.* \square

It is known that there exists an optimal monotone search strategy for any graph [28] and it can be computed in linear time for a tree [31]. An optimal connected search strategy can be also computed in linear time for a tree [2].

Using Lemmas 2.2 and 6.1 we conclude with the following theorem:

Theorem 5. *Let $G = (V(G), E(G), c)$ be a tree such that the subgraph G_j composed by the edges in E_j is connected for each $j \in \{1, 2, \dots, z\}$. Then, there exists a polynomial-time algorithm for solving problems HGS and HCGS.*

7 Conclusions and open problems

Recalling our main motivation standing behind introducing this graph searching model, we note that its properties allow for much easier construction of graphs in which recontaminations need to occur in optimal strategies. Our main open question, following the same unresolved one for connected searching, is whether problems HGS and HCGS belong to NP?

Our more practical motivation for studying the problems is derived from modeling physical environments to whose parts different robots have different access. More complex scenarios than the one considered in this work are those in which either an edge can have multiple colors (allowing it to be traversed by all searchers of those colors), and/or a searcher can have multiple colors, which in turns extends its range of accessible parts of the graph. As a way of modeling mobile entities of different types cooperating to solve various computational tasks (of which searching is just one example), heterogeneous mobile agent computing is receiving a growing interest, including fields like distributed computing. Hence, one may ask for different ways of modeling differences between searchers, which may fit potential practical applications.

References

- [1] O. Amini, D. Coudert, and N. Nisse. Non-deterministic graph searching in trees. *Theor. Comput. Sci.*, 580:101–121, 2015.
- [2] L. Barrière, P. Flocchini, F.V. Fomin, P. Fraigniaud, N. Nisse, N. Santoro, and D.M. Thilikos. Connected graph searching. *Inf. Comput.*, 219:1–16, 2012.
- [3] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *SPAA'02: Proc. of the Fourteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 200–209, New York, NY, USA, 2002. ACM.
- [4] A. Bärtschi, J. Chalopin, S. Das, Y. Disser, D. Graf, J. Hackfeld, and P. Penna. Energy-efficient delivery by heterogeneous mobile agents. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 10:1–10:14, 2017.
- [5] D. Bienstock and P. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991.
- [6] L. Blin, J. Burman, and N. Nisse. Exclusive graph searching. *Algorithmica*, 77(3):942–969, 2017.
- [7] N.E. Clarke and E.L. Connon. Cops, robber, and alarms. *Ars Comb.*, 81:283–296, 2006.
- [8] N.E. Clarke and R.J. Nowakowski. Cops, robber, and photo radar. *Ars Comb.*, 56:97–103, 2000.
- [9] N.E. Clarke and R.J. Nowakowski. Cops, robber and traps. *Utilitas Mathematica*, 60:91–98, 2001.
- [10] J. Czyzowicz, L. Gasieniec, A. Kosowski, and E. Kranakis. Boundary patrolling by mobile agents with distinct maximal speeds. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 701–712, 2011.
- [11] Jurek Czyzowicz, Evangelos Kranakis, Dominik Pajak, and Najmeh Taleb. *Patrolling by Robots Equipped with Visibility*, pages 224–234. Springer International Publishing, Cham, 2014.
- [12] D. Dereniowski. Maximum vertex occupation time and inert fugitive: Recontamination does help. *Inf. Process. Lett.*, 109(9):422–426, 2009.
- [13] D. Dereniowski. Connected searching of weighted trees. *Theor. Comp. Sci.*, 412:5700–5713, 2011.
- [14] D. Dereniowski. Approximate search strategies for weighted trees. *Theor. Comput. Sci.*, 463:96–113, 2012.
- [15] D. Dereniowski. From pathwidth to connected pathwidth. *SIAM J. Discrete Math.*, 26(4):1709–1732, 2012.

- [16] D. Dereniowski, D. Dyer, R.M. Tifenbach, and B. Yang. The complexity of zero-visibility cops and robber. *Theor. Comput. Sci.*, 607:135–148, 2015.
- [17] D. Dereniowski, R. Klasing, A. Kosowski, and Ł. Kuszner. Rendezvous of heterogeneous mobile agents in edge-weighted networks. *Theor. Comput. Sci.*, 608:219–230, 2015.
- [18] D. Dyer, B. Yang, and Ö. Yasar. On the fast searching problem. In *Algorithmic Aspects in Information and Management, 4th International Conference, AAIM 2008, Shanghai, China, June 23-25, 2008. Proceedings*, pages 143–154, 2008.
- [19] A. Farrugia, L. Gasieniec, Ł. Kuszner, and E. Pacheco. Deterministic rendezvous with different maps. In *Journal of Computer and System Sciences*, volume 106, pages 49–59, 2019.
- [20] O. Feinerman, A. Korman, S. Kutten, and Y. Rodeh. Fast rendezvous on a cycle by agents with different speeds. In *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings*, pages 1–13, 2014.
- [21] F.V. Fomin, P. Heggernes, and J.A. Telle. Graph searching, elimination trees, and a generalization of bandwidth. *Algorithmica*, 41(2):73–87, 2004.
- [22] F.V. Fomin and D.M. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, 131(2):323–335, 2003.
- [23] P. Fraigniaud and N. Nisse. Monotony properties of connected visible graph searching. *Inf. Comput.*, 206(12):1383–1393, 2008.
- [24] S. Gaspers, M.-E. Messinger, R.J. Nowakowski, and P. Pralat. Parallel cleaning of a network with brushes. *Discrete Applied Mathematics*, 158(5):467–478, 2010.
- [25] G.A. Hollinger, A. Kehagias, and S. Singh. GSST: anytime guaranteed search. *Auton. Robots*, 29(1):99–118, 2010.
- [26] D. Ilcinkas, N. Nisse, and D. Soguet. The cost of monotonicity in distributed graph searching. *Distributed Computing*, 22(2):117–127, 2009.
- [27] A. Kawamura and Y. Kobayashi. Fence patrolling by mobile agents with distinct speeds. *Distributed Computing*, 28(2):147–154, 2015.
- [28] A.S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.
- [29] G.A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Self-stabilizing meeting in a polygon by anonymous oblivious robots. *CoRR*, abs/1705.00324, 2017.
- [30] E. Markou, N. Nisse, and S. Pérennes. Exclusive graph searching vs. pathwidth. *Inf. Comput.*, 252:243–260, 2017.
- [31] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, and C.H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, 1988.
- [32] R. Mihai and I. Todinca. Pathwidth is NP-hard for weighted trees. In *FAW '09: Proc. of the 3rd Inter. Workshop on Frontiers in Algorithmics*, pages 181–195, Berlin, Heidelberg, 2009. Springer-Verlag.
- [33] T.D. Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs, Lecture Notes in Mathematics*, volume 642, pages 426–441. Springer-Verlag, 1978.
- [34] N.N. Petrov. A problem of pursuit in the absence of information on the pursued. *Differentsial'nye Uravneniya*, 18:1345–1352, 1982.
- [35] Z. Qian, J. Li, X. Li, M. Zhang, and H. Wang. Modeling heterogeneous traffic flow: A pragmatic approach. *Transportation Research Part B: Methodological*, 99:183–204, 2017.

- [36] S. Sundaram, K. Krishnamoorthy, and D.W. Casbeer. Pursuit on a graph under partial information from sensors. *CoRR*, abs/1609.03664, 2016.
- [37] C. Worman and B. Yang. Searching trees with sources and targets. In *FAW '08: Proc. of the 2nd annual international workshop on Frontiers in Algorithmics*, pages 174–185, Berlin, Heidelberg, 2008. Springer-Verlag.
- [38] B. Yang, D. Dyer, and B. Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, 2009.