Perpetual Graph Searching

Lélia Blin, Janna Burman, Nicolas Nisse

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Perpetual Graph Searching

Lélia Blin — Janna Burman — Nicolas Nisse

N° 7897

February 2012

Domaine

ISSN 0249-6399 ISRN INRIA/RR--7897--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

Perpetual Graph Searching *

Lélia Blin† , Janna Burman‡ , Nicolas Nisse‡

Thème :
Équipe-Projet Mascotte

Rapport de recherche n° 7897 — February 2012 — 26 pages

Abstract: In *graph searching*, a team of mobile agents aims at clearing the edges of a contaminated graph. To clear an edge, an agent has to slide along it, however, an edge can be *recontaminated* if there is a path without agents from a contaminated edge to a clear edge. To goal of graph searching is to clear the graph, i.e., all edges are clear simultaneously, using the fewest number of agents. We study this problem in the *minimal CORDA* model of distributed computation. This model has very weak hypothesis: network nodes and agents are anonymous, have no memory of the past, and agents have no common sense of orientation. Moreover, all agents execute the same algorithm in the *Look-Compute-Move* manner and in an asynchronous environment. One interest of this model is that, if the clearing can be done by the agents starting from arbitrary positions (e.g., after faults or recontamination), the lack of memory implies that the clearing is done perpetually and then provides a first approach of fault-tolerant graph searching. Constraints due to the minimal CORDA model lead us to define a new variant of graph searching, called *graph searching without collisions*, where more than one agent cannot occupy the same node.

We show that, in a centralized setting, this variant does not have the same behavior as classical graph searching. For instance, it not monotonous nor close by subgraph. We show that, in a graph with maximum degree Δ, the smallest number of agents required to clear a graph without collisions is at most Δ times the number of searchers required when collisions are allowed. Moreover, this bound is tight up to a constant ratio. Then, we fully characterize graph searching without collisions in trees.

In a distributed setting, i.e., in the minimal CORDA model, the question we ask is the following. Given a graph G, does there exist an algorithm that clears G, whatever be the initial positions of the agents on distinct vertices. In the case of a path network, we show that it is not possible is the number of agents is even in a path of odd order, or if there are at most two agents in a path with at least three vertices. We present an algorithm that clears all paths in all remaining cases. Finally, we propose an algorithm that clears any tree using a sufficient number of agents.

Key-words: Distributed computing, CORDA model, graph searching, self-stabilization

* Partially supported by FP7 STREP EULER (N.N.).
† Univ. Pierre et Marie Curie - Paris 6, LIP6-CNRS UMR 7606, France
‡ MASCOTTE, INRIA, I3S(CNRS/Univ. Nice Sophia Antipolis), France. firstname.lastname@inria.fr

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Nettoyage perpétuel de réseaux

Résumé : Dans le cadre du *nettoyage de graphes contaminés* (*graph searching*), des agents mobiles se déplacent successivement le long des arêtes du graphe afin de les *nettoyer*. Le but général est le nettoyage en utilisant le moins d'agents possible. Nous plaçons notre étude dans le modèle de calcul distribué *CORDA minimaliste*. Ce modèle est muni d'hypothèses très faibles : les nœuds du réseau et les agents sont anonymes, n'ont pas de mémoire du passé ni sens commun de l'orientation et agissent par *cycles Voir-Calculer-Agir* de manière *asynchrone*. Un intérêt de ce modèle vient du fait que si le nettoyage peut être fait à partir de positions arbitraires des agents (par exemple, après pannes ou recontamination), l'absence de mémoire implique un nettoyage perpétuel et donc fournit une première approche de nettoyage de graphe *tolérant aux pannes*. Les contraintes dues au modèle *CORDA minimaliste* nous amènent à définir une nouvelle variante de nettoyage de graphes - le *nettoyage sans collision*, autrement dit, plusieurs agents ne peuvent occuper simultanément un même sommet.

Nous montrons que, dans un contexte *centralisé*, cette variante ne satisfait pas certaines propriétés classiques de nettoyage comme par exemple la monotonie. Nous montrons qu'interdire les "collisions" peut augmenter le nombre d'agents nécessaires d'un facteur au plus Δ le degré maximum du graphe et nous illustrons cette borne. De plus, nous caractérisons complètement le nettoyage sans collision dans les arbres.

Dans le contexte *distribué*, la question qui se pose est la suivante. Existe t'il un algorithme qui, étant donné un ensemble d'agents mobiles arbitrairement répartis sur des sommets distincts d'un réseau, permet aux agents de nettoyer perpétuellement le graphe? Dans le cas des chemins, nous montrons que la réponse est négative si le nombre de agents est pair dans un chemin d'ordre impair, ou si il y a au plus deux agents dans un chemin d'ordre au moins 3. Nous proposons un algorithme qui nettoie les chemins dans tous les cas restants, ainsi qu'un algorithme pour nettoyer les arbres lorsqu'un nombre suffisant d'agents est disponible initialement.

Mots-clés : algorithme distribué, CORDA, nettoyage de graphe, auto-stabilisation

1 Introduction

In the CORDA distributed computing model (a.k.a., *Look-Compute-Move* model) [18, 30], each network entity (e.g., mobile agent on a graph) is *autonomous* and acts by *asynchronous cycles*. A cycle consists of 3 phases: *Look, Compute* and *Move*. Each phase is executed atomically and in an asynchronous way. That is, there is a finite but unbounded time between two phases. An agent first has access to a *snapshot*, or *view*, of the network that comprises the topology of the network, the vertex occupied by the agent and all vertices that are occupied by agents (Look Phase). Note that, by atomicity of a phase, when a snapshot is taken, all agents are occupying a vertex, i.e., no agent is currently crossing an edge. However, depending on the model, it is possible or not to distinguish the number of agents occupying the same vertex. Based on the snapshot, the agent can execute some computation (Compute Phase) for determining its next action: to move to an adjacent vertex or not (Move Phase). Note that the time-complexity of the Compute Phase is not taken into account. The difficulty comes from the fact that the autonomous entities must achieve a common goal being coordinated only thanks to the snapshots. In particular, because of the asynchronous setting, the Compute and Move Phases may be based on out-dated snapshots.

Several works study the impact of various extra constraints in this model (e.g., [9, 16]). In this work, we consider a model that we call *minimalist CORDA model* and that have very weak hypothesis. The graph (nodes and links) is *anonymous* and no local memory is available on nodes. Agents are anonymous, *uniform* (they all execute the same algorithm), *oblivious* (i.e., have no memory of the past) and have no common sense of orientation. Moreover, at every step, at most one agent can occupy some node. This last constraint is called *exclusivity constraint* [5, 6]. All these assumptions make symmetry braking, and hence the algorithm design, harder.

Graph exploration and *Rendez-vous* have been studied in the CORDA model (see below). In both cases, proposed algorithms have some *self-stabilization* property [13]: proposed algorithms perform whatever be the valid starting configuration. Here, "arbitrary valid configuration" means valid according to the model: in particular, in the model with exclusivity property, only starting configurations where all nodes are occupied by at most one agent are considered. Indeed, without such assumption, problems such as exploration would have to be solved with only one agent since, starting with all agents at the same node and because of the asynchronicity, all agents may always occupy the same vertex. However, one oblivious agent cannot explore a graph [].

In this work, we propose the first study of the *graph searching* problem in the CORDA model. In *edge-graph searching* [29], the edges of a graph are initially *contaminated* and a team of agents, called *searchers*, aims at clearing the whole graph by sliding along the edges. A *clearing strategy* is a sequence of sliding of the agents along the edges that results in clearing the graph, i.e., in all edges being simultaneously clear. The goal is to minimize the number of searchers required for such a strategy. It is easy to see that one searcher is sufficient only in the path network. Here, we aim at designing algorithms that resist to *recontamination*, that is, that perpetually clear the graph especially after some edges having been recontaminated. More precisely, we aim at designing algorithms allowing a team of agents to clear a graph whatever be the valid starting configuration. Therefore, it is natural to restrict our investigation to strategies that satisfy the exclusivity property.

We first introduce a new variant of graph searching where no two searchers can occupy the same node, i.e., satisfying the exclusivity property, and we study this variant in a centralized setting. Then, we study this variant in the minimalist CORDA model. As far as we know, it is the first study of graph searching in some CORDA model.

1.1 Exploration and Rendez-vous in the CORDA model

Recently, several mobile agents problems have been considered in the CORDA model. In the *exploration with terminaison* problem, a team of mobile agents is spread in a graph and each node must be visited at least once by an agent, then agents must be able to decide that all vertices have been visited at least once [9, 15–17]. In the *perpetual exploration* problem, each node must be visited infinitely often by each agent [5, 6]. In the problem of *Rendez-vous*, all agents must eventually occupy the same vertex [11, 22, 23].

Table 1: Exploration and rendez vous in the CORDA model.

	Paths	Trees	Rings	Grids	Arbitrary graphs
Exploration with stop	[17]	[16]	[15]		[9]
	+ power of detecting towers of agents				+ local edge labeling
Perpetual Exploration			[5]	[6]	
			exclusivity property		
Rendez-vous			[11, 22]		

In these works, authors aim at identifying the pairs (k, I_k) where k is the number of agents and $I_k \subseteq 2^V$ is the set of valid starting positions for k agents. That is, there is an algorithm that solves the problem with k agents if the set of the starting positions of the k agents is valid.

In the aforementioned works, extra hypothesis are sometimes added compared to the minimalist CORDA model. For instance, it might be possible to occupy the same node with several agents or, even more, agents may detect agent-tower, i.e., the vertices occupied by more than one agent. Sometimes, the anonymous constraint is relaxed to allow a local edge labeling (e.g., port-numbers). These works are summarized in Table 1.

In our work, we consider the same model as [5,6,11], i.e., we are in the minimalist CORDA model, including the exclusivity property.

1.2 Graph Searching Problem

Edge-Search. *Graph Searching* has been introduced by Breisch [7, 8] and then formalized by Parsons [29]. It has mainly be studied for its relationship with *treewidth* and *pathwidth* of graphs [2, 19].

In *edge-graph searching* [29], the edges of a graph are initially *contaminated*. A team of *searchers* aims at clearing the whole graph. For this purpose, a *strategy* for the searchers is a sequence of the following actions or *moves*. The searchers may (1) be placed at vertices; (2) be removed from vertices; (3) slide along the edges of the graph. An edge is *cleared* when a searcher slides along it. On the other hand, an edge is *recontaminated* if there is a path from it to a contaminated edge such that no searchers stand on this path. A strategy is *winning* if all edges of the graph are simultaneously clear after the execution of the strategy. The number of searchers *used* by a strategy is the maximum number of searchers that are simultaneously occupying some nodes of the graph. Finally, the *edge-search number* of a graph G, denoted by $es(G)$, is the least number of searchers used by a winning strategy on G. A winning strategy clearing G with $es(G)$ searchers is called *optimal*.

For instance, $es(P) = 1$ for any path P and a corresponding winning strategy consists in placing the searcher on one end of P and then sliding it toward the other end. Other classical examples are any cycle C or any complete graph K_n with $n \geq 4$ vertices, for which $es(C) = 2$ and $es(K_n) = n$ respectively.

An important property of edge-graph searching is the *monotonicity*. A strategy is *monotone* if an edge is never recontaminated once it has been cleared. For any graph G, there is an optimal winning monotone strategy [2, 24]. By extension, edge-graph seaching is said monotone. In particular, this property allows to prove that the size of an optimal strategy is polynomially bounded by the number of edges. Hence, the problem to decide the edge-search number of a graph belongs to NP.

Let us remark that, during a strategy, two actions (not necessarily consecutive) that consist into removing a searcher from a vertex v and then placing this searcher at a vertex u can be simulated by sliding the searcher along the edges of any path from v to u. Hence, any winning strategy can be easily transformed, without increasing the number of searchers used, into a winning strategy such that searchers are first placed at some vertices of the graph and then only sliding moves are performed. However, such a strategy may not be monotone anymore.

Given a graph G and an integer k as an input, the problem to decide whether $es(G) \leq k$ is NP-complete [25]. However, this problem can be decided in polynomial-time when the input graph is restricted to some graphs' classes, e.g., in trees [10, 14, 32].

Node-Search and Mixed-Search. Two main other variants of graph searching games have been defined. In *node-graph searching* [21], only moves of types (1) and (2) are allowed for searchers, i.e., a searcher cannot slide along an edge. The goal is then to clear all vertices, given that a vertex is cleared when a searcher stands at it and that a vertex is recontaminated if it is not occupied by a searcher and there exists a path without searchers from it to a contaminated vertex. In *mixed-graph searching* [2], all three types of moves are allowed for the searchers and edges must be cleared, however an edge is cleared either when a searcher slides along it or when both its ends are occupied by some searchers. The *node search number* of a graph G, denoted by $ns(G)$, and its *mixed search number*, denoted by $mxs(G)$, are defined as the least number of searchers required to clear G in the corresponding model. In any graph G, $es(G), ns(G)$ and $mxs(G)$ may differ by at most one [2, 21]. For instance, in any path P with at least 2 vertices, $1 = es(P) = mxs(P) < ns(P) = 2$; for any cycle C, $2 = es(C) = mxs(C) < ns(C) = 3$; for any complete graph K_n with $n \geq 4$ vertices, $n = ns(K_n) = es(K_n) > mxs(K_n) = n-1$, finally for any $n > 2$, $es(K_{n,n}) = n+2 > ns(K_{n,n}) = mxs(K_{n,n}) = n+1$.

Node graph searching has been defined because it provides an algorithmic interpretation of the *pathwidth* $pw(G)$ of a graph G, namely $ns(G) = pw(G)+1$ for any graph G [1,2]. Mixed graph searching has been introduced to prove monotonicity (and so the relationship with pathwidth) of graph searching games [2]. Deciding the node search number of a planar graph of maximum degree at most 3 is NP-complete [27].

Distributed graph-searching. Edge-graph searching has also been studied in a distributed setting in [4, 20, 28]. These papers mainly discuss the possible tradeoffs between *connectivity*[1] and monotonicity constraints and the number of required searchers. In these papers, searchers and nodes are supposed to have distinct IDs. This strong hypothesis is not assumed in this work. [10] proposes a unified distributed algorithm to compute any of the edge- node- and mixed-search number in trees. Recently, self-stabilizing algorithms for clearing trees in the node-search model have been designed in [3, 26]. Again, in these papers, searchers have distinct IDs.

1.3 Distributed Model and graph searching with exclusivity property

We consider a team of $k \geq 1$ agents initially spred on the nodes of a network, modeled by a connected undirected graph, and that can move along the links of this graph. The agents are anonymous, i.e., cannot be distinguished from each other. They all execute the same algorithm and they are oblivious, meaning that they have no memory of their past actions. Finally, they cannot communicate with each other but can sense the environment and see the position (nodes) of other agents.

The nodes of the network are anonymous. In particular, in a symmetric network (e.g., a ring), the agents cannot distinguish between two configurations where the relative positions of agents are the same. However, at any node v, a local labeling allows a agent to recognize the edge it has to move along to reach a given neighbor of v.

Following [5], the agents operate in cycles that comprise of three phases: Look, Compute and Move. During the Look phase agents take a snapshot of their environment. The collected information (position of the other agents) are used in the compute phase in which agents decide to move to an adjacent node or to stay idle. Such a move is done in the last phase (move phase). The computational model we consider is the CORDA model in a discrete setting, i.e. when a agent takes a snapshot of the network, it sees the other agents on nodes only. On the other hand, the time between Look, Compute, and Move phases is finite yet unbounded, and is decided by the adversary for each cycle/phase of each agent. Thus, because of such an asynchrony, different agents can execute different phases concurrently (e.g., a agent can perform a look phase while another agent is moving), and a agent can use an outdated snapshot of the network to compute whether to move and where.

[1] A search strategy is connected if the set of clear edges (resp., nodes) induces a connected subgraph at each step.

In such a setting, two agents occupying the same node will execute the same sequence of moves whatever be the algorithm (since the adversary will act for this purpose). Hence, we assume that initially, no node is occupied by more than one agent. Such a configuration is called *valid*. That is, we consider only configurations satisfying the exclusivity property. In particular, our algorithms must ensure that an agent never slides toward a node that is already occupied.

The goal of the team of agents is to clear the edges of the network starting from any valid configuration. Note that, if the agents can only slide along edges of the graph and that no vertex can be occupied by more than one agent, a triangle (cycle with three nodes) cannot be cleared in the edge-graph searching setting (i.e., when an edge is cleared only when a agent slide along it). Hence, we consider that an edge is cleared either when a agent slide along it or when both its ends are occupied.

More precisely, a *perpetual searching algorithm*, or $\mathcal{PSA}$, executed by a agent $\mathcal{R}$ takes a configuration, i.e., the positions of all the agents, as an input and computes the next position of this agent: either it does not move or it goes to a unoccupied neighbor of its current position. A $\mathcal{PSA}$ is *valid* if, starting from any initial valid configuration and whatever be the behavior of the adversarial scheduler, all edges of the graph are cleared after a finite number of moves and the configurations reached are all valid, i.e., the algorithm must ensure that two agents never occupy the same node simultaneously. In particular, permanently applying such a process will ensure a perpetual cleaning of the network.

1.4 Our Results

We first study the impact of the exclusivity property for graph searching in a centralized setting, i.e., when a central entity may control and coordinate all searchers. That is, we consider a new variant of graph searching, called *graph searching without collision*, where searchers can only slide along edges in such a way that at most one searcher occupies a vertex at each step. Moreover, clearing an edge can be done by sliding along it or by occupying both its ends simultaneously. Let $ws(G)$ be the fewest number of searchers required to clear a graph G in this setting. This study in a centralized setting allows us to provide bounds (in terms of number of searchers) that will be useful to evaluate the performances of our distributed algorithms.

The study in a centralized setting is interesting by itself. Indeed, we show that graph searching without collision has a different behavior as edge (resp., node, mixed)-graph searching. First, graph searching without collision does not satisfy the monotonicity property nor the subgraph-closeness, that is, there are graphs such that H is a (induced or not) subgraph of G and $ws(H) > ws(G)$. Then, we show that the difference between ws and ns may be arbitrary large. More precisely, we show that in any star T with $\Delta > 3$ leaves, $ws(T) > \Delta - 2 > 2 = ns(T)$. On the positive side, we show that, for any graph G with maximum degree Δ, $ns(G) - 1 \leq ws(G) \leq (\Delta - 1)ns(G)$. Moreover, if $\Delta \leq 3$, then $ns(G) - 1 \leq ws(G) \leq ns(G)$. Then, we show that ws is close under subgraph in trees, which allows us to fully characterize this parameter in trees. More precisely, given $k \geq 1$, we give a list of necessary and sufficient conditions for a tree T to have $ws(T) = k$. This allows to prove that ws is polynomially computable in the class of trees.

On the other hand, we study graph searching without collision in the minimalist CORDA model with the exclusivity property. In a distributed setting, i.e., in the minimal CORDA model, the question we ask is the following. Given a graph G, does there exist an algorithm that clears G, whatever be the initial positions of the agents on distinct vertices. In the case of a path network, we show that it is not possible is the number of agents is even in a path of odd order, or if there are at most two agents in a path with at least three vertices. We present an algorithm that clears all paths in all remaining cases. Finally, we propose an algorithm that clears any tree using a sufficient number of agents.

2 On graph searching without collisions in a centralized setting

In previous section, we introduced a new variant of graph searching which is a natural model in the distributed setting considered in this paper. This section is devoted to some basic properties of this variant in a centralized setting. These results will be useful in the sequel to provide some lower bounds on the number of agents required in a distributed setting. This also shows that most of the basic properties (monotonicity, closed under taking minors) that are shared by "classical" graph searching variants are not satisfied by this new variant.

In *graph searching without collisions*, a strategy for clearing a graph $G = (V,E)$ is defined as follows. First a set $I \subseteq V$ of vertices are chosen as initial positions and exactly one searcher is placed at each vertex in I. Note that, in particular, all searchers that will be used during a strategy must be present in the graph initially. Then, the strategy consists of a sequence of moves where a move is the sliding of a searcher from its current position to an *unoccupied* neighbor of it. An edge is cleared when a searcher slides along it or when both its ends are occupied, and an edge is recontaminated as usual. Like in other variants, the goal is to reach a state when all edges are simultaneously cleared. Let $ws(G)$ denote the least number of searchers used by a winning strategy on G in this variant.

We need some notations. Let $G = (V,E)$ be a connected graph. A *configuration* is a pair $\mathcal{C} = (I,F)$ where $I \subseteq V$ is the subset of vertices, and $F \subseteq E$ is a set of edges such that, for any two incident edges $e \in F$ and $f \notin F$, their common end is in I. Intuitively, I is the set of vertices occupied by a searcher and F is the set or a subset of clear edges. We say that there is a strategy from $\mathcal{C} = (I,F)$ to $\mathcal{C}' = (I',F')$ if there a sequence of moves (a searcher can slide to any unoccupied neighbor) starting from $\mathcal{C} = (I,F)$ and ending in $\mathcal{C}' = (I',F')$. That is, initially the searchers are in I and the edges of F are cleared, and after the moves, the searchers are in I' and the set of clear edges contains F'. Note that $|I| = |I'|$.

2.1 General properties of graph searching without collision

Let us start with some easy lemmas that especially express the fact that graph searching without collision does not behave as classical variants of graph searching (e.g., node search).

Lemma 1. *Let G be any connected graph with a cut-vertex v and let $cc(v) \geq 2$ be the number of the connected components in $G \setminus \{v\}$, then $ws(G) > cc(v) - 2$.*

Proof. Let v be a cut-vertex of a graph G and consider any strategy using at most $cc(v) - 2$ searchers. Initially, at least two components of $G \setminus \{v\}$, say U and W, are unoccupied and so all edges in these components are contaminated. Now, consider the first step when a searcher occupies a vertex in one of these components, say U. That is, let us consider the first step when a searcher slides from v to a vertex in U. But after this step, no searchers are occupying a vertex in W which is still contaminated, no searcher is occupying v and there is a connected component C of $G \setminus (\{v\} \cup U \cup W)$ that contains no searchers. Hence, C is (re)contaminated because of W. Hence, at any step of the strategy, at least two connected components of $G \setminus \{v\}$ remain contaminated. □

Corollary 1. *In any tree T with maximum degree $\Delta \geq 2$, $ws(G) > \Delta - 2$.*

Lemma 2. *For any graph G, $mxs(G) \leq ws(G)$ and the difference may be arbitrary large.*

Proof. The first statement follows the fact that graph searching without collisions can be defined as the mixed-graph searching variant with the additional constraint that no two searchers may occupy simultaneously a node. The second statement comes from Corollary 1 by noticing that any star with $n+1$ vertices (i.e., with maximum degree n) has mixed-search number 2. □

Lemma 3. *Graph searching without collisions is not monotone.*

Proof. Let G be the graph with vertex set $\{a_1,a_2,a_3,b,c_1,c_2,c_3\}$ such that (a_1,a_2,b,c_2,c_1) is a path and a_3 is adjacent to a_2 and c_3 is adjacent to c_2 (G is a 1-caterpillar).

We first prove that $ws(G) = 2$. A winning strategy using two searchers is defined as follows: choose $\{a_1,a_3\}$ as initial positions and place searchers A and B on these vertices respectively. Then, B slides along the edges of the path from a_3 to c_2, A slides along the edges from a_1 to b, B goes to c_3 (here the edge $\{b,c_2\}$ is recontaminated), and finally, A slides to c_1. The fact that $ws(G) > 1$ is obvious and follows Corollary 1.

Finally, consider any winning strategy using two searchers, we show that there is a step with recontamination. There are two cases to be considered. Either the set of initial positions contains no vertices in $C = \{c_1,c_2,c_3\}$ (or symmetrically in $A = \{a_1,a_2,a_3\}$), or the initially one searcher occupies a vertex in C and the other searcher occupies a vertex in A.

In the first case, i.e., no vertices in C are occupied initially, consider the first step when c_1 or c_3, w.l.o.g., say c_1, is occupied: this must follow a move of a searcher along the edge $\{c_2,c_1\}$ but then, the edge $\{b,c_2\}$ is recontaminated by $\{c_2,c_3\}$ (since c_3 has never been occupied yet).

In the second case, consider the first searcher to reach b, w.l.o.g., it comes from a_2. Then, it is easy to verify that $\{a_2,b\}$ is recontaminated because a single searcher cannot have cleared $\{a_1,a_2\}$ and $\{a_3,a_2\}$ simultaneously. □

Lemma 4. *There are graphs G with (induced) subgraph H such that $ws(H) > ws(G)$.*

Proof. Let H be a $(2n+1)$-node star with leaves $\{a_1,b_1,\cdots,a_n,b_n\}$. By Corollary 1, $ws(H) \geq 2n-1$ and it is easy to check that $ws(H) = 2n-1$. Let G be the graph obtained by adding an edge between a_i and b_i, for any $i \leq n$ (resp., by adding n vertices c_i such that c_i is adjacent to a_i and b_i, for any $i \leq n$). H is a subgraph of G (respectively an induced subgraph of G). Finally, $ws(G) \leq n+1$: place searchers at the center of the star and at any a_i, $i \leq n$, and then, each searcher occupying a vertex a_i slides to b_i. □

In the classical variants of graph searching, it is trivial to show that, if k searchers can clear a graph G, then $k' > k$ searchers can do as well. Indeed, we can simply not use the $k'-k$ extra searchers, or we can place them at some arbitrary vertices and do not use them after that. This is not trivial anymore in the graph searching without collision since, by definition, all searchers must occupy some vertices initially, and an extra searcher could disturb other searchers by being on their way. We show that increasing the number of searchers do not disturb the clearing anyway.

Lemma 5. *Given a graph G, if $ws(G) = k$, then for every $k \leq k' \leq V(G)$, there exists a winning strategy with exclusivity property, for G, using k' searchers.*

Proof. Let $\mathcal{S} = (s_1,\cdots,s_r)$ be a sequence of moves from configuration $\mathcal{C} = (I,F)$ to configuration $\mathcal{C}' = (I',F')$ in G. Let $v \in V(T) \setminus I$. We show, by induction on r, that there is a sequence $\mathcal{S}'$ of moves from configuration $\mathcal{C} = (I \cup \{v\},F)$ to configuration $\mathcal{C}' = (I' \cup \{w\},F^*)$, with $w \in V(G) \setminus I'$ and $F' \subseteq F^*$.

If $r = 1$, then there are two cases. Either s_1 slides a searcher to v or not. In the first case, $\mathcal{S}'$ does nothing (the sequence is empty) and in the latter case, $\mathcal{S}' = \mathcal{S} = (s_1)$. It is easy to check that $\mathcal{S}'$ satisfies the conditions.

Now, assume $r > 1$ and let $\mathcal{C}_1 = (I_1,F_1)$ reached after step s_1 of $\mathcal{S}$. There are two cases according whether $v \in I_1$ or not. If $v \notin I_1$, by the induction hypothesis, there is a sequence $\mathcal{S}'$ of moves from $(I_1 \cup \{v\},F_1)$ to $\mathcal{C}' = (I' \cup \{w\},F^*)$, with $w \in V(G) \setminus I'$ and $F' \subseteq F^*$. In that case, $\mathcal{S}^*$ consists of $(s_1,\mathcal{S}')$. If $v \in I_1$, the step s_1 consists in sliding a searcher from a vertex $x \in I$ to v. Therefore, $(s_2,\cdots,s_r)$ is a sequence of move from (I_1,F_1) to (I',F') and $x \notin I_1$. By the induction hypothesis, there is a sequence $\mathcal{S}'$ of moves from $(I_1 \cup \{x\},F_1)$ to $\mathcal{C}' = (I' \cup \{w\},F^*)$, with $w \in V(G) \setminus I'$ and $F' \subseteq F^*$. In that case, $\mathcal{S}^* = \mathcal{S}'$. □

Theorem 1. *For any connected graph G with maximum degree Δ,*

$$ns(G) - 1 \leq ws(G) \leq (\Delta - 1) \cdot ns(G).$$

Moreover, if $\Delta \leq 3$,

$$ns(G) - 1 \leq ws(G) \leq ns(G).$$

Proof. The fact that $ns(G) - 1 \leq ws(G)$ comes from Lemma 2 and from the fact that $ns(G) \leq mxs(G) + 1$ for any graph G.

Let $\mathcal{S}$ be a winning monotone node-search strategy of G using $k \geq ns(G)$ searchers. Let $(s_1, \cdots, s_r)$ be the corresponding sequence of place and remove steps. Note that we can assume that a searcher is removed as soon as possible, i.e., if after some step s_i, a searcher occupies a vertex incident only to clear edges, there is no place-move before this searcher is removed. We also may assume that there are no useless steps, i.e., no step consists in placing a searcher at a vertex incident to only clear edges nor in placing a searcher at an already occupied vertex. Last remark is that, at each step s_i that consists in placing a searcher at a vertex v, before this step, all edges incident to v are contaminated.

For any i, $1 \leq i \leq r$, let E_i be the set of clear edges after step s_i and let V_i be the set of occupied vertices after step s_i. By remark above, if s_{i+1} consists in placing a new searcher, then any vertex of V_i is incident to at least one contaminated edge after step s_i. We set $V_0 = E_0 = \emptyset$. Because the strategy is monotone, $E_{i-1} \subseteq E_i$ for any $1 < i \leq r$. Moreover, any path from an edge in E_i and an edge in $E \setminus E_i$ must contain a vertex in V_i.

- Let us first assume that $\Delta \leq 3$. To show that $ws(G) \leq ns(G)$, we use $\mathcal{S}$ to define a strategy $\mathcal{S}^*$ without collisions to clear G using k searchers. Initially, the k searchers are placed at any arbitrary k vertices. Then, the strategy consists of r phases described below, where Phase i of $\mathcal{S}^*$ depends on step s_i of $\mathcal{S}$ ($i \leq r$). For any $1 \leq i \leq r$, we show that, if before Phase i, the vertices of V_{i-1} were occupied and the edges of E_{i-1} were clear, then after Phase i, the vertices of V_i are occupied and the edges of E_i are clear. In particular, this implies that after Phase r, the edges of $E_r = E$ are clear. Note that, before Phase 1, the induction hypothesis holds since $V_0 = E_0 = \emptyset$. Now, let $1 \leq i \leq r$ and assume that the induction hypothesis holds before Phase i.

  The easy case is when s_i is a remove-step. In that case, Phase i consists in doing nothing and the induction still holds before Phase $i+1$.

  Let us now assume that step s_i consists in placing a searcher at some vertex $v \in V$. Therefore, $|V_{i-1}| < k$ and there are searchers that are not occupying a vertex in V_{i-1} just before Phase i. We say these searchers are *free*. Let us consider the free searcher that is the closest to v just before Phase i. The Phase i consists in "moving" this searcher from its current position to v. We show how it can be done without recontaminating any edge in E_{i-1} and then we show that the edges of E_i are clear at the end of the Phase i.

  Let w be the vertex occupied by the considered free searcher at the end of Phase $i-1$ and let $P = (w = u_1, \cdots, u_h = v)$ be a shortest path from w to v.

  - First assume that P contains no vertices in V_{i-1}. Then, no vertices in $\{u_2, \cdots, u_h\}$ are occupied because we have considered the free searcher that is closest to v. Therefore, Phase i consists in sliding this searcher from w to v along P. Let F be the set of edges that are incident to some vertex of P. To show that no edges in E_{i-1} is recontaminated during these moves, it is sufficient to point out that either $F \subseteq E_{i-1}$ or $F \subseteq E \setminus E_{i-1}$. Indeed, we already mentioned that any path between an edge in E_{i-1} and an edge in $E \setminus E_{i-1}$ contains a vertex in V_{i-1}. In other words, an edge of E_{i-1} might be recontaminated only by moving a searcher from a vertex in V_{i-1}.
  - Otherwise, there exist $2 \leq j \leq \ell < h$ such that, for any $q < j$, $u_q \notin V_{i-1}$; for any $j \leq q \leq \ell$, $u_q \in V_{i-1}$ and $u_{\ell+1} \notin V_{i-1}$. In that case, the free searcher first slides from w to u_{j-1}. No edges of E_{i-1} are recontaminated for the same reason as in previous item. Then, we simulate the "jump" of the free searcher from u_{j-1} to $u_{\ell+1}$ in the following way. For t from ℓ down to j, the searcher at u_t slides to u_{t+1}. Finally, the free searcher occupying u_{j-1} slides to u_j.

When a searcher moves from u_t to u_{t+1}, the only two edges that might be recontaminated are $\{u_{t-1},u_t\}$ and $\{u_{t+1},u_t\}$ because u_t has degree at most three. However, $\{u_{t+1},u_t\}$ is clear as soon as the next searcher moves from u_{t-1} to u_t.

Therefore, after these $\ell - j + 2$ moves, the edges of E_{i-1} are still clear, the vertices of V_{i-1} are still occupied and a (new) free searcher is occupying the vertex $u_{\ell+1}$, i.e., the free searcher has progressed along P.

The same process is done until a searcher reaches v (always following P).

To conclude, after Phase i, a searcher is on v and all vertices of V_{i-1} are occupied, hence the vertices of $V_i = V_{i-1} \cup \{v\}$ are occupied. Moreover, we showed above that the edges of E_{i-1} are still clear after Phase i. Since $E_i \setminus E_{i-1}$ is the set of edges that are incident to v and to some vertex in V_{i-1}, they are clear as well at the end of this phase.

Hence, the induction hypothesis holds after Phase i and $\mathcal{S}^*$ is a winning strategy without collisions for clearing G with k searchers. Hence, $ws(G) \leq ns(G)$.

- Now, let us consider the case of an arbitrary Δ. We use $\mathcal{S}$ to define a strategy $\mathcal{S}^*$ without collisions to clear G using $(\Delta-1)k$ searchers. As previously, $\mathcal{S}^*$ consists of r phases such that, after Phase i, $i \leq r$, we ensure that the edges in E_i are clear. The difference with previous case is that at each step, we ensure that all vertices of V_i are occupied after Phase i and, moreover, for any vertex of V_i at most two neighbors of it are not occupied.

  As previously, all the searchers are placed on arbitrary different vertices initially. Then, for each step s_i ($1 \leq i \leq r$), if s_i is a remove-step then Phase i consists in doing nothing. Otherwise, if s_i consists in placing a searcher at some vertex v, this means that $|V_{i-1}| < k$ and therefore there are at least $\Delta - 1$ free searchers. Here, each vertex in V_{i-1} is assigned $\leq \Delta - 1$ searchers, the remaining searchers being the free searchers. Phase i consists in moving $\Delta - 1$ free searchers from their current positions to v or some neighbor of v until v is occupied and at most two neighbors of v are not occupied.

  More precisely, while v has not been assigned $\Delta - 1$ searchers and there still is a vertex in $N[v]$ that is unoccupied, we do the following. Let consider the free searcher not occupying a vertex in $N[v]$ that is the closest to a unoccupied vertex x in $N[v]$. This searcher is assigned to v in the following manner. Let w be the vertex occupied by this free searcher at the end of Phase $i-1$, and let $P = (w = u_1, \cdots, u_h = x)$ be a shortest path from w to x. As previously, if no vertices in $\{u_2, \cdots, u_h\}$ are occupied, the free searcher simply slides along the path until it reaches x.

  Otherwise, w.l.o.g., assume that $\{u_2, \cdots, u_j\}$ are occupied and u_{j+1} is not for some $2 \leq j < h$. Then, the strategy is the following. For t from j down to 1, if $u_t \notin V_{i-1}$, the searcher at u_t slides to u_{t+1}. Otherwise, if $u_t \in V_{i-1}$ there are two cases. If there are two neighbors $z \neq u_{t+1}$ and $y \neq u_{t+1}$ of u_t that are not occupied, then the searcher at u_t slides to y. Note that such a case may occur only if $u_{t+1} \notin V_{i-1}$. Otherwise, the searcher at u_t slides to u_{t+1}.

  The fact that, after Phase i, all edges of E_i are clear and that all vertices of V_i and their neighbors (but at most two per vertex in V_i) are occupied follows.

□

Beside it provides an upper bound on *ws*, Theorem 1 implies that, if the node-search number cannot be approximated with constant ratio in the class of bounded degree graphs, then computing *ws* is NP-hard. In particular, if *ws* can be computed in polynomial-time in the class of graph with maximum degree 3, then it gives a $+1$ approximation algorithm to compute the node search number in this class of graphs. As far as we know, no such un-approximability results nor approximation algorithms are known.

It is known that computing the mixed-search number is known to be NP-complete in planar graphs with maximum degree 3 [27], however, it is only known that it cannot be approximated up to an additive constant in general graphs [12]. As far as we know, the best approximation algorithm to compute the search number of planar graphs has constant factor $O(\log n)$, combining the constant factor approximation for treewidth in planar graphs [31] with the basic $O(\log n)$ approximation for computing a path-decomposition from a tree-decomposition.

2.2 Searching a tree without collision

The last part of this section is dedicated to trees. Let T be a tree and $v \in V(T)$. A *branch* of T at v is any connected component of $T \setminus v$.

In the case of trees, we need to strengthen the result of Lemma 5. More precisely, if we have extra searchers, we need to control where they are at the end of the strategy. We prove a slightly stronger result that we will use later.

Proposition 2. *Let T be a tree and assume there is a sequence $\mathcal{S} = (s_1, \cdots, s_r)$ of moves from $\mathcal{C} = (I, F)$ to $\mathcal{C}' = (I', F')$. Let $v \in V(T) \setminus I$. Then, for any $F^* \supseteq F$, there is a sequence $\mathcal{S}'$ of moves in T starting from $(I \cup \{v\}, F^*)$ and ending in configuration $(I' \cup \{w\}, F'')$, where $w \in V(T) \setminus I'$, $F' \subseteq F''$ and, if $v \notin I'$ then $w = v$.*

Proof. If $|V(T)| = 1$, the claim clearly holds. Assume by induction on $n > 1$ that the claim holds for any tree T with $|V(T)| < n$. We now show that it holds if $|V(T)| = n$ by induction on $r \geq 0$. We prove the result for $F^* = F$, if $F \subset F^*$, the result trivially follows.

If the first step s_1 of $\mathcal{S}$ consists of sliding a searcher from $x \in I$ to $y \neq v$ and F^* is the set of clear edges after this step in $\mathcal{S}$, then the first step of $\mathcal{S}'$ is s_1 and then the claim holds by induction on r by considering the sequence of moves $(s_2, \cdots, s_r)$ starting from configuration $((I \cup \{y\}) \setminus x, F^*)$.

Hence, let us assume that the first step s_1 of $\mathcal{S}$ consists in sliding a searcher from $x \in I \cap N(v)$ to v. There are two cases: either v remains occupied during the whole strategy $\mathcal{S}$, or let s_i, $1 < i \leq r$, be the first step such that the searcher at v slides to some vertex $y \in N(v)$. Let $T_1, \cdots, T_p$ be the connected components of $T \setminus \{v\}$ where T_p is the component of x.

- Consider the first case, i.e., from s_1, v is always occupied in $\mathcal{S}$. We may assume (up to reordering the steps in $\mathcal{S}$) that $\mathcal{S}$ applies move s_1, and then applies the moves in $T_1, \cdots, T_p$ sequentially, i.e., any move in T_i is performed before any move in T_{i+1} for any $1 \leq i < p$.

  Consider the restriction of $\mathcal{S}$ to T_p, it starts from $((I \cap V(T_p)) \setminus \{x\}, F^*)$, where F^* is the set of edges of $E(T_p)$ that are cleared after step S_1 of $\mathcal{S}$. Moreover, the restriction of $\mathcal{S}^*$ to T_p terminates in $(I' \cap V(T_p), F' \cap E(T_p))$. Since $|V(T_p)| < |V(T)|$, using induction, there is a sequence of moves $\mathcal{S}^p$ in T_p that starts from $((I \cap V(T_p)) \cup \{x\}, F^*)$ and terminates in $(I' \cap V(T_p) \cup \{w\}, F^p)$ where $F' \cap E(T_p) \subseteq F^p$, $w \notin I' \cap V(T_p)$ and $w = x$ if $x \notin I' \cap V(T_p)$.

  Then, the following sequence of moves proves the claim. Starting from $(I \cup \{v\}, F)$, apply sequentially the moves of $\mathcal{S}$ in $T_1, \cdots, T_{p-1}$ and finally apply the sequence of moves $\mathcal{S}^p$.

- Now, let s_i, $1 < i \leq r$, be the first step of $\mathcal{S}$ such that the searcher at v slides to some vertex $y \in N(v)$. Let $(I^i; F^i)$ be the configuration reached after this step. Note that $v \notin I^i$.

  Therefore, the sequence of moves $(s_{i+1}, \cdots, s_r)$ allows to go from configuration $(I^i; F^i)$ to configuration (I', F'). By induction on r, there is sequence of moves $\mathcal{S}^{**}$ that allows to go from $(I^i \cup \{v\}; F^i)$ to $(I' \cup \{w\}, F'')$, where $w \notin I'$, $F' \subseteq F''$ and $v \notin I'$ then $w = v$. Note that it easily implies that, for any F^* with $F^i \subseteq F^*$, there is sequence of moves that allows to go from $(I^i \cup \{v\}; F^*)$ to $(I' \cup \{w\}, F'')$, where $w \notin I'$, $F' \subseteq F''$ and $v \notin I'$ then $w = v$.

It only remains to prove that there is a sequence $\mathcal{S}^*$ of moves that starts in $(I\cup\{v\},F)$ and terminates in $(I^i\cup\{v\};F^*)$ where $F^i\subseteq F^*$. Let $\mathcal{S}^{i-1}$ be the sequence of moves $(s_1,\cdots,s_{i-1})$ and let $(I^{i-1};F^{i-1})$ be the configuration reached by $\mathcal{S}^{i-1}$.

Let us define $\mathcal{S}^*$ as follows. First, apply the sequentially the moves of $\mathcal{S}^{i-1}$ in $T_1,\cdots,T_{p-1}$. Then, if $y\neq x$, move the searcher at v to y and the searcher at x to v and then apply the moves of $\mathcal{S}^{i-1}$ in T_p. Otherwise, as in previous item, by induction since $|V(T_p)|<|V(T)|$, from the sequence of moves of $\mathcal{S}^{i-1}$ restricted to T_p, we can define a sequence of moves in T_p that starts in the same configuration as in $\mathcal{S}^{i-1}$ (restricted to T_p) plus one extra searcher at x and that reach the same final configuration as in $\mathcal{S}^{i-1}$ (restricted to T_p) with the extra searcher still at x. This sequence of moves is applied to conclude $\mathcal{S}^*$.

The sequence of moves that consists of applying the moves in $\mathcal{S}^*$ and then the moves in $\mathcal{S}^{**}$ satisfies the claim.

□

Proposition 3. *Let T be a tree and assume there is a sequence $\mathcal{S}=(s_1,\cdots,s_r)$ of moves from $\mathcal{C}=(I,F)$ to $\mathcal{C}'=(I',F')$. Assume that there is $v\in I$ such that v is occupied during the first $r-1$ steps of $\mathcal{S}$ and s_r consists in sliding the searcher at v to one of its neighbors. Note that $v\notin I'$. Then, for any $F^*\supseteq F$, there is a sequence $\mathcal{S}'$ of moves in T starting from $(I\cup\{w\},F^*)$ and ending in configuration $(I'\cup\{v\},F'')$, where $w\in V(T)\setminus I$, $F'\subseteq F''$.*

Proof. We prove the result for $F^*=F$, if $F\subset F^*$, the result trivially follows.

We build a strategy $\mathcal{S}'=(s'_1,\cdots,s'_p)$ with an extra searcher (compared with $\mathcal{S}$) that starts in some vertex $w\notin I$ and ends in v such that, at the end of $\mathcal{S}'$ all edges in F are cleared and the set of occupied vertices is $I'\cup\{v\}$.

We need to find an unoccupied initial position w for the extra searcher, such that it ensures that v will be occupied at the end of the sequence of moves. Let $\{v=w_\ell,\cdots,w_1=w\}$, $\ell>1$, be the path defined as follows. $w\in V(R)\setminus I$, i.e., w_1 is not initially occupied in $\mathcal{S}$. For any $1<j\leq\ell$, w_j is initially (in $\mathcal{S}$) occupied by a searcher, i.e., $w_j\in V(R)\cap I$, and the first move of this searcher in $\mathcal{S}$, at step s^*_j, is to slide from w_j to w_{j-1}. Such a path clearly exists and $s^*_j>s^*_{j-1}$ (meaning step s^*_j occurs before step s^*_{j+1} in $\mathcal{S}$) for any $1\leq j<\ell$. In particular, note that $s^*_\ell=s_r$.

Strategy $\mathcal{S}'$ initially places the extra searcher at $w_1=w$ and other searchers are placed at the vertices of I.

Let (I_2,F_2) be the configuration reached by $\mathcal{S}$ just before step s^*_2 (when the searcher at w_2 moves for the first time and goes to $w_1=w$). $w\notin I^2$, therefore, by Proposition 2, there is sequence $\mathcal{S}_2$ of moves that goes from configuration $(I\cup\{w\},F)$ to $(I_2\cup\{w\},F'_2)$ with $F_2\subseteq F'_2$.

Therefore, starting with the extra searcher in w and executing $\mathcal{S}_2$, we arrive in the same configuration as $\mathcal{S}$ before s^*_2 but with the extra searcher still in w (possibly more edges may have been cleared). Then s^*_2 moves a searcher from $w_2\in I_2$ to w_1. Actually, since both w_2 and w_1 are occupied after executing $\mathcal{S}_2$, we have reached the configuration after step s^*_2 in $\mathcal{S}$, having one extra searcher in w_2.

Assume that we have executing the sequence of moves $(\mathcal{S}_2,\cdots,\mathcal{S}_i)$, $i<\ell$, and that we have reached the configuration $(I_i\cup\{w_i\},F_i)$ where (I_i,F_i) is the configuration reached by $\mathcal{S}$ after step s^*_i (in particular, $w_i\notin I_i$). Let (I_{i+1},F_{i+1}) be the configuration reached by $\mathcal{S}$ just before step s^*_{i+1}. Note that $w_{i+1}\notin I_{i+1}$. By Proposition 2, there is sequence $\mathcal{S}_{i+1}$ of moves that goes from configuration $(I_i\cup\{w_i\},F_i)$ to $(I_{i+1}\cup\{w_i\},F'_{i+1})$ with $F_{i+1}\subseteq F'_{i+1}$. Then s^*_{i+1} moves a searcher from $w_{i+1}\in I_{i+1}$ to w_i. Actually, since both w_{i+1} and w_i are occupied after executing $\mathcal{S}_{i+1}$, we have reached the configuration after step s^*_{i+1} in $\mathcal{S}$, having one extra searcher in w_{i+1}.

Going on in that way, the sequence of moves $(\mathcal{S}_2,\cdots,\mathcal{S}_\ell)$ satisfies the desired requirements. □

Lemma 6. *Let T be any tree and R be any subtree of T, then $ws(R)\leq ws(T)$.*

Proof. We only need to prove the result for $V(T)\setminus V(R)$ is a leaf of T and then the result follows by induction on $|V(T)\setminus V(R)|$. Hence, let v be the leaf of T such that $R=T\setminus\{v\}$ and let $u\in V(R)$ its neighbor.

If $|V(R)|\leq ws(T)$, the statement clearly holds. Hence, we assume that $|V(R)|>ws(T)$.

Let $\mathcal{S} = (s_1, \cdots, s_r)$ be a strategy without collisions that clears T with $k \geq ws(T)$ searchers. Let us set $\mathcal{S} = (\mathcal{S}_1, \cdots, \mathcal{S}_p)$ where $\mathcal{S}_i$ is a maximal sub-sequence of $\mathcal{S}$ where the number of searchers occupying $V(R)$ is constant. That is, for any $1 \leq i \leq p$, either

- $k-1$ searchers occupy the vertices in R during $\mathcal{S}_i$ and, if $i > 1$, the last move of $\mathcal{S}_{i-1}$ is to slide a searcher from u to v, and if $i < p$, the first move of $\mathcal{S}_{i+1}$ is to slide a searcher from v to u. In particular, during $\mathcal{S}_i$, a searcher always occupies v;
- or, k searchers occupy the vertices in R during $\mathcal{S}_i$ and, if $i > 1$, the first move of $\mathcal{S}_i$ is to slide a searcher from v to u, and if $i < p$, the last move of $\mathcal{S}_i$ is to slide a searcher from u to v; In particular, during $\mathcal{S}_i$, a searcher never occupies v (but possibly after the last step);

We now describe a strategy $\mathcal{S}^*$ using at most k searchers to clear R without collision. Let I be the set of vertices initially occupied by the searchers in $\mathcal{S}$.

If $p = 1$, it means that, in $\mathcal{S}$, a searcher always stays at v. In that case, we omit this searcher, i.e., $\mathcal{S}^*$ uses only $k-1$ searchers and directly follows from $\mathcal{S}$. That is, $\mathcal{S}^*$ starts by placing searchers at the vertices of $I \cap V(R) = I \setminus \{v\}$, and for each step of $\mathcal{S}$ that slides some searcher along an edge, Strategy $\mathcal{S}^*$ performs the same move. Such a strategy clearly clears R using $k-1$ searchers. Therefore, the result holds.

Now, assume $p > 1$, we build $\mathcal{S}^* = (\mathcal{S}_1^*, \cdots, \mathcal{S}_p^*)$ using k searchers. For any $i \leq p$, let (I_i, F_i) be the configuration reached after the last step of $\mathcal{S}_i$.

If $\mathcal{S}_1$ uses k searchers in R, then $\mathcal{S}_1^*$ exactly follows $\mathcal{S}_1$, i.e., the k searchers are placed at $I \subseteq V(R)$ and for each step of $\mathcal{S}_1$ that slides some searcher along an edge, Strategy $\mathcal{S}_1^*$ performs the same move. Clearly, configuration $(I_1 \cap V(R), F_1 \cap E(R))$ is reached after the last step of $\mathcal{S}_1^*$

If $\mathcal{S}_1$ uses $k-1$ searchers in R, then it is a sequence of moves that starts from configuration $(I \cap V(R), \emptyset)$ and reaches configuration $(I_1 \cap V(R), F_1 \cap E(R))$ with $u \notin I_1 \cap V(R)$. By Proposition 3, there is a vertex $w \in V(R)$ and a sequence $\mathcal{S}_1^*$ of moves that starts from configuration $((I \cap V(R)) \cup \{w\}, \emptyset)$ and reaches $((I_1 \cap V(R)) \cup \{u\}, F_1')$ with $F_1 \cap E(R) \subseteq F_1'$.

Let $1 \leq i < p$ and assume we have built the sequence $(\mathcal{S}_1^*, \cdots, \mathcal{S}_i^*)$ that uses k searchers and reaches the configuration $((I_i \cap V(R)) \cup \{u\}, F_i')$ where $F_i \cap E(R) \subseteq F_i'$.

If $\mathcal{S}_{i+1}$ uses k searchers in R, in particular, its first move is to slide a searcher from v to u and its last move (if $i+1 < p$) is to slide back a searcher from u to v. Then $\mathcal{S}_{i+1}^*$ exactly follows $\mathcal{S}_{i+1}$ but the first and the last move. Clearly, configuration $((I_{i+1} \cap V(R)) \cup \{u\}, F_{i+1}')$ where $F_{i+1} \cap E(R) \subseteq F_{i+1}'$ is reached after the last step of $\mathcal{S}_{i+1}^*$.

If $\mathcal{S}_{i+1}$ uses $k-1$ searchers in R, then it is a sequence of moves that starts from configuration $(I_i \cap V(R), F_i \cap E(R))$ and reaches configuration $(I_{i+1} \cap V(R), F_{i+1} \cap E(R))$ with $u \notin I_i \cap V(R)$ and, if $i+1 < p$, $u \notin I_{i+1} \cap V(R)$. By Proposition 2, for any $F_i' \supseteq F_i$, there is a sequence $\mathcal{S}_{i+1}^*$ of moves that starts from configuration $(I_i \cap V(R) \cup \{u\}, F_i')$ and reaches $((I_{i+1} \cap V(R)) \cup \{u\}, F_{i+1}')$ where $F_{i+1} \cap E(R) \subseteq F_{i+1}'$.

Going on by induction on p, the sequence $\mathcal{S}^* = (\mathcal{S}_1^*, \cdots, \mathcal{S}_p^*)$ built in that way clears R without collision. □

Lemma 7. *Let T be any tree and R be a branch at $v \in V(T)$. Let $\mathcal{S}' = (s_1', \cdots, s_r')$ be a strategy without collision for T and let k be the maximum number of searchers occupying simultaneously (at the same step) some vertices of R. Then $ws(R) \leq k$.*

Proof. Let u be the vertex of R that is adjacent to v and let $e = \{u, v\} \in E(T)$. Note that, for all $i \leq r$, s_i' consists of the sliding of a searcher along an edge of $E(T)$.

We consider the restriction of $\mathcal{S}'$ to R which is the sequence $\mathcal{S}$ of moves built as follows. For any step s_i' of $\mathcal{S}'$, if s_i' consists of sliding a searcher along an edge $\{x, y\} \in E(R)$, we keep this move ; if s_i' consists of sliding a searcher along an edge $\{x, y\} \in E(T) \setminus (E(R) \cup \{e\})$ (i.e., $v \notin \{x, y\}$), we remove s_i' ; if s_i' consists in sliding a searcher from v to u, we add a *particular move* that consists in placing a searcher at u ; and if s_i' consists in sliding a searcher from u to v, we add a *particular move* that consists in removing a searcher at u.

Therefore, $\mathcal{S} = (s_1, \cdots, s_p)$ is a sequence of moves that are either sliding a searcher along an edge of R, or a particular step which is either placing a searcher at u or removing a searcher at u. This sequence results in the clearing of all edges in R and no more than k vertices are occupied at some step.

We prove by induction on the number of particular steps of $\mathcal{S}$ (removing or placing a searcher) that $ws(R) \leq k$.

FIrst assume that $\mathcal{S}$ contains no particular steps. Clearly, it is a strategy without collision, using k searchers and clearing all edges of R. Therefore, the result holds.

If $\mathcal{S}$ contains some particular steps, there are three cases to be considered.

- Assume that the last particular step s_i ($i \leq p$) of $\mathcal{S}$ is a removal step. Let (I,F) be the configuration after this step and let (I',F') be the configuration at the end of $\mathcal{S}$. Note that $u \notin I$, that $|I| < k$ and that $(I \cup \{u\}, F^*)$, $F \subseteq F^*$, is the configuration just before s_i.

  By Proposition 2, there is a sequence $\mathcal{S}^*$of moves (slidings) from $(I \cup \{u\}, F^*)$ to $(I' \cup \{w\}, F'')$ with $w \in V(R) \setminus I'$ and $F' \subseteq F''$. Hence, the sequence of moves $(s_1, \cdots, s_{i-1}, \mathcal{S}^*)$ clears all edges in R and no more than k vertices are occupied at some step. Moreover, it has less particular moves than $\mathcal{S}$, therefore the result holds by induction.

- Assume that the first particular step s_i ($i \leq p$) of $\mathcal{S}$ is a placing step. Let (I,F) be the configuration at the beginning of $\mathcal{S}$ and let (I',F') be the configuration before this step s_i. Note that $u \notin I'$, that $|I| < k$ and that $(I' \cup \{u\}, F')$ is the configuration just after s_i.

  By Proposition 3 and Proposition 2, there is a sequence $\mathcal{S}^*$of moves (slidings) from $(I \cup \{w\}, F)$ to $(I' \cup \{u\}, F'')$ with $w \in V(R) \setminus I$ and $F' \subseteq F''$. Hence, the sequence of moves $(\mathcal{S}^*, s_{i+1}, \cdots, s_p)$ clears all edges in R and no more than k vertices are occupied at some step. Moreover, it has less particular moves than $\mathcal{S}$, therefore the result holds by induction.

- If none of the previous cases hold, then there are $1 \leq i < j \leq p$ such that s_i is a removal step, s_j is a placing step and s_ℓ is not a particular step for all $i < \ell < j$. Let (I,F) be the configuration after step s_i and let (I',F') be the configuration before step s_j. Note that $u \notin I \cup I'$, that $|I| = |I'| < k$ and that $(I \cup \{u\}, F^*)$, $F' \subseteq F^*$, is the configuration just before s_i and $(I' \cup \{u\}, F')$ is the configuration just after s_j.

  From Proposition 2, there is a sequence $\mathcal{S}^*$of moves (slidings) from $(I \cup \{u\}, F^*)$ to $(I' \cup \{u\}, F'')$ with $F' \subseteq F''$. Hence, the sequence of moves $(s_1, \cdots, s_{i-1}, \mathcal{S}^*, s_{j+1}, \cdots, s_p)$ clears all edges in R and no more than k vertices are occupied at some step. Moreover, it has less particular moves than $\mathcal{S}$, therefore the result holds by induction.

□

Note that the previous lemma will fail if R is not restricted to be a branch of T. To see this, consider a star like tree T with a "central" node v and n branches at v s.t. each branch is a path of length 1. Now, consider a subgraph R of T which is an $n+1$-node star induced by $v \bigcup N(v)$. Note that $ws(R) = ws(T) = n-1$. However, there is a wining strategy for T where at any moment, only 2 searchers are simultaneously in R.

All previous lemmas allow us to prove a Parson-like characterization of trees T with $ws(T) \leq k$. We first recall the Parson's Theorem.

Theorem 4. *[29] For any tree T and $k \geq 1$, $es(T) > k$ if and only if there is a vertex $v \in V(T)$ and at least three branches T_i of $T \setminus \{v\}$ with $es(T_i \cup \{v\}) \geq k$.*

The above theorem is not true anymore when considering graph searching without collision as the following example shows. Consider an example of a tree T with node v that has 3 branches T_1, T_2, T_3 and for $i = 1,2,3$, $T_i \cup \{v\}$ is (m+1)-node star. Thus, $ws(T_i \cup \{v\}) = m-1$. Note that there is a winning strategy for T using only $m-1$ searchers.

However, we show that a characterization similar to the one of Parson for edge-search holds for graph searching without collision.

Lemma 8. *For any tree T and integer $k \geq 1$, if $\exists v \in V(T)$ with at least 3 branches $\{T_i | i = 1,2,3\}$ s.t. $ws(T_i) \geq k$, then $ws(T) > k$.*

Proof. By Lemma 6, $ws(T) \geq k$. Assume by contradiction that $ws(T) = k$ and let $\mathcal{S}$ a corresponding strategy for T. Then, by Lemma 6, $ws(T_i) = k$, for $i = 1,2,3$. By lemma 7, for any $i \leq 3$, there is a step of the strategy $\mathcal{S}$ such that k searchers occupy simultaneously vertices in T_i. Let s_i be the last such step of $\mathcal{S}$ that occurs in T_i. W.l.o.g. assume that $s_1 < s_2 < s_3$ and we may assume that, before step s_i, T_i is not completely clear. Then, at step s_2, no vertices of T_1 or T_3 are occupied. However, T_3 being not cleared yet, it causes the recontamination of all vertices of T_1. Since $ws(T_1) = k$ and it is completely contaminated after step s_2, by Lemma 7, there must be a step after s_2 where k searchers occupy simultaneously the vertices of T_1. A contradiction. □

Lemma 9. *For any tree T and integer $k \geq 1$, if $\exists v \in V(T)$ and an even integer $i > 3$ such that there is a set $B = \{T_j : ws(T_j) \geq k - i/2 + 1\}$ of branches at v and $|B| > i$, then $ws(T) \geq k$.*

Proof. Let $\mathcal{S}$ be any sequence of moves that clears T without collision. By lemma 7, for any $j \leq |B|$, there is a step of the strategy $\mathcal{S}$ such that k searchers occupy simultaneously vertices in T_j. Let s_j be the last such step of $\mathcal{S}$ that occurs in T_j. W.l.o.g. assume that $s_{j-1} < s_j$, for any $1 < j \leq i$, and we may assume that, before step s_j, T_j is not completely clear (this means that $\mathcal{S}$ uses k searchers in T_j only if it is really needed). Then, at step $s_{i/2}$, $k - i/2 + 1$ searchers are in $T_{i/2}$, some vertices have been cleared in T_j for any $j < i/2$, and T_j cannot become fully contaminated anymore until the end of the strategy (otherwise there would another step after s_j where k searchers are in T_j).

For purpose of contradiction, let us assume that $\mathcal{S}$ uses at most $k-1$ searchers. Then, at step $s_{i/2}$, at least $k - i/2 + 1$ searchers are in $T_{i/2}$ and there are at most $i/2 - 2$ searchers outside $T_{i/2}$. That is, at that moment, there is at least one branch $X \in \{T_{i/2+1}, \ldots, T_{|B|}\}$ at v with still contaminated edges and one branch $Y \in \{T_1, \ldots, T_{i/2-1}\}$ at v with some clear edges that must not be recontaminated and no searchers occupy nodes in both these branches. If there is no searcher at v (in particular, it is the case for $i = 4$), Y is fully recontaminated - a contradiction.

Otherwise, there is a searcher in v. However, since there is at least one non cleared yet branch without any searcher in it, it has to be cleared by moving there at least one searcher. For that, the searcher from v have to move. However, if this searcher moves (no matter where), there will be still at most $i/2 - 2$ searchers outside $T_{i/2}$ and hence, at least one cleared and one uncleared branch without any searcher, and no searcher in v. The cleared branch will be recontaminated - a contradiction. □

Lemma 10. *Let T be a tree and $k \geq 1$. If, for any $v \in V(T)$*

1. *v has degree at most $k+1$ and,*
2. *for any branch S at v, $ws(S) \leq k$, and*
3. *at most two branches R and R' at v are such that $ws(R) = ws(R') = k$, and*
4. *for any even $i > 3$, $|\{S \text{ branch at } v \ : \ ws(S) \geq k - i/2 + 1\}| \leq i$,*

then $ws(T) \leq k$.

Proof. As in [29], we first build a particular path A, called *avenue*, in T, such that all connected components S of $T \setminus A$ are such that $ws(S) < k$.

By the assumption, for any $v \in V(T)$, there is no branch R at v with $ws(R) > k$ and there are at most 2 branches R at v with $ws(R) = k$.

Let $v \in V(T)$ that has the maximum number of branches S with $ws(S) = k$.

- If $ws(S) < k$ for any branch S at v, then let $A = \{v\}$.
- If there is only one branch S at v with $ws(S) = k$, let u be the neighbor of v in S. Then, either all connected components S of $T \setminus \{u,v\}$ are such that $ws(S) < k$ in which case $A = \{u,v\}$, or there is a branch S' at u, $v \notin S'$ with $ws(S') = k$. In that case, let w be the neighbor of u in S'. Go on this process with u instead of v and w instead of u until two adjacent vertices x and y are found such that all connected components S of $T \setminus A$, $A = \{x,y\}$, are such that $ws(S) < k$.
- Finally, if two branches S and S' at v are such that $ws(S) = ws(S') = k$. In that case, let u be the neighbor of v in S and let w be the neighbor of v in S'. Starts with $A = \{u,v,w\}$. If u has two branches R and R' with $ws(R) = ws(R') = k$, then by Lemma 6, one of these branches, say R', is the one containing v. Let x be the neighbor of u in R and add it to A.

  Let us continue this process iteratively while possible : assume that, at some step of the process, A is the path $(v_1, \cdots, v_i)$. If there is an end of A, say v_i, such that two branches R and R' at v_i satisfy with $ws(R) = ws(R') = k$, then by construction and by Lemma 6, one of these branches, say R', is the one containing v_{i-1}. Let v_{i+1} be the neighbor of v_i in R and add it to A. The process clearly terminates since, at each step, there are at most two components of $T \setminus A$ with $ws = k$ and their size strictly decreases.

  This process ends with A is a path such that any connected component S of $T \setminus A$ are such that $ws(S) < k$.

Let $A = \{u_1, \cdots, u_p\}$ be the avenue. Let R^1 be the branch at u_1, not containing u_2 (if $p > 1$), and with maximum search number. Let x be the neighbor of u_1 in R^1. Let S^p be the branch at u_p, not containing u_{p-1} (if $p > 1$) and different from R^1 if $p = 1$, and with maximum search number. Let y be the neighbor of u_p in S^p.

For any $i \leq p$, let $v^i_1, \cdots, v^i_{d_i}$ be the neighbors of u_i not in $A \cup \{x,y\}$ ($d_i \leq k-1$ for any $1 \leq i \leq p$) and let T^i_j be the branch at u_i containing v^i_j, for any $j \leq d_i$. Let R^i be the branch at u_i containing u_{i-1} for any $i > 1$ and let S^i be the branch at u_i containing u_{i+1} for any $i < p$.

We assume that, for any $i \leq p$, the vertices $v^i_1, \cdots, v^i_{d_i}$ are ordered according to their search number in the following way: $ws(T^i_1) \geq ws(T^i_{d_i}) \geq ws(T^i_2) \geq ws(T^i_{d_i-1}) \geq \cdots \geq ws(T^i_j) \geq ws(T^i_{d_i-j+1}) \geq \cdots$. Note that, by definition of A, for any $i \leq p$ and any $j \leq d_i$, $ws(T^i_j) \leq k-1$.

We built a strategy that clears T without collision and using k searchers as follows.

First place one searcher at u_1, place $\ell = \min\{k-1, |V(R^1)|\}$ searchers at the set J of vertices of R^1 such that it is possible to clear R^1 starting from J using ℓ searchers without collision (such a J exists because $ws(R^1) \leq k-1$ and by Lemma 5) and place the $k-1-\ell$ remaining searchers (if any) at any unoccupied vertices.

The strategy starts by clearing R^1 using the ℓ searchers defined above.

Assume that after some steps, we have reached the following configuration. Let i, $1 \leq i \leq p$ and let $1 \leq j_0 \leq d_i + 1$ such that:

- R^i is clear;
- for any $j < j_0$, T^i_j is clear;
- u_i is occupied;
- if $1 < j_0 \leq \lceil d_i/2 \rceil$, the vertices in $\{v^i_1, \cdots, v^i_{j_0-1}\}$ are occupied, and if $\lceil d_i/2 \rceil < j_0$, the vertices in $\{v^i_{j_0}, \cdots, v^i_{d_i}\}$ are occupied;
- all other searchers occupy distinct vertices and are called *free searchers*.

We call such a configuration a (i, j_0)-*situation*. Note that after having cleared R^1, we reach the (i, j_0)-situation where $i = 1$ and $j_0 = 2$.

There are several cases to be considered

- if $j_0 = d_i + 1$, then, if $i < p$, then the strategy consists in sliding the searcher at u_i to u_{i+1}. Then, we reach the $(i+1,1)$-situation.

  Otherwise, if $i = p$, then $T \setminus S^p$ is clear. To clear S^p, we first gather $ws(S^p) \leq k-1$ searchers in S^p using the following process. First, the searcher at u_p goes to its neighbor y in S^p (unless y is already occupied by a free searcher). Then, while there are less than $ws(S^p)$ searchers in S^p, one searcher not in S^p goes to u_p, the searcher at y goes further in S^p and then the searcher at u_p goes to y. When $ws(S^p)$ searchers are in S^p (one of them occupies y), a searcher not in S^p (it exists since $ws(S^p) \leq k-1$) goes to u_p. Finally, the searchers in S^p clear this branch and T is clear, which terminates the strategy.

- if $j_0 = 1$. By definition of A, $ws(T^i_{j_0}) \leq k-1$. In this case, R is cleared and one searcher occupies u_i and there are $k-1$ free searchers. We show that we can bring $ws(T^i_{j_0})$ searchers in $T^i_{j_0}$ without recontamination of R^i. Let x' be the neighbor of u_i in R^i (x' is either u_{i-1} or x). Consider a free searcher not in $T^i_{j_0}$ with no searcher between its current position and u_i and slide this searcher until he reaches a neighbor y' of u_i. Either $y' \neq x'$, in which case, the searcher at u_i goes at x', then the free searcher goes at u_i and then enters in $T^i_{j_0}$ and finally, the searcher at x' goes back to u_i. Or $y' = x'$, in which case the searcher at u_i enters in $T^i_{j_0}$ and the searcher at x' goes to u_i.

  Once $ws(T^i_{j_0})$ searchers have reached $T^i_{j_0}$ and another searcher occupies u_i, we clear $T^i_{j_0}$ using the searchers in it. Then, one searcher in $T^i_{j_0}$ goes to $v^i_{j_0}$.

  Finally, if $j_0 = \lceil d_i/2 \rceil < d_i$, then, the free searchers goes to the vertices in $\{v^i_{j_0+1}, \cdots, v^i_{d_i}\}$ while $\{v^i_1, \cdots, v^i_{j_0}\}$ and u_i remain occupied to avoid recontamination (we use the same process as above to simulate the fact that a searcher "crosses" u_i). This is possible since u_i has degree at most $k+1$ (by assumption (1)) and therefore $d_i \leq k-1$.

  In any case, we have reached the (i, j_0+1)-situation.

- if $j_0 \leq \lceil d_i/2 \rceil$ and not the previous cases. Let $k' = ws(T^i_{j_0})$. Because of the ordering of the subtrees T^i_j, there are at least $2j_0+1$ branches at u_i with search number k' (these branches are $R^i, S^i, T^i_1, T^i_{d_i}, T^i_2, T^i_{d_i-1}, \cdots, T^i_{j_0}$). Because $j_0 > 1$, by assumption (4), $|\{S \text{ branch at } u_i \ : \ ws(S) \geq k - j_0 + 1\}| \leq 2j_0$. Therefore $k' < k - j_0 + 1$. Since exactly $k - j_0$ searchers are free (those that are not at u_i nor at $\{v^i_1, \cdots, v^i_{j_0-1}\}$), it is possible to bring the free searchers in $T^i_{j_0}$ by the same process as in previous case. Then $T^i_{j_0}$ can be cleared and a searcher in it can reach $v^i_{j_0}$.

  Finally, as in the previous case, if $j_0 = \lceil d_i/2 \rceil$, it is possible to reach the configuration where all vertices in $\{v^i_1, \cdots, v^i_{d_i}\}$ and u_i without recontaminating $R^i, T^i_1, \cdots, T^i_{j_0}$.

  In any case, we have reached the (i, j_0+1)-situation.

- if $j_0 > \lceil d_i/2 \rceil$ and not the previous cases. Let $k' = ws(T^i_{j_0})$. Because of the ordering of the subtrees T^i_j, there are at least $2(d_i - j_0 + 2)$ branches at u_i with search number k'. Because $j_0 \leq d_i$, by assumption (4), $k' < k - \lceil (2(d_i - j_0 + 2))/2 \rceil + 1 = k - (d_i - j_0 + 2) + 1$. Since exactly $k - (d_i - j_0 + 2)$ searchers are free (those that are not at u_i nor at $\{v^i_{j_0}, \cdots, v^i_{d_i}\}$), it is possible to bring the free searchers in $T^i_{j_0}$. To do so, there is a small difference with before: to let a searcher pass through u_i, the searcher at u_i goes to its neighbor in S^i (while before it was to its neighbor in R^i) and then goes back to u_i.

  Then $T^i_{j_0}$ can be cleared and a searcher in it can reach $v^i_{j_0}$ and we have reached the (i, j_0+1)-situation.

The above strategy clears T using k searchers which proves the lemma. □

Theorem 5. *For any tree T and $k \geq 1$, $ws(T) \leq k$ if and only if for any $v \in V(T)$*

1. *v has degree at most $k+1$ and,*

2. *for any branch S at v, $ws(S) \leq k$, and*

3. *at most two branches have search number (ws) exactly k, and*

4. *for any even $i > 3$, $|\{S \text{ branch at } v \ : \ ws(S) \geq k - i/2 + 1\}| \leq i$.*

Proof. The "if" part directly follows from Lemma 10. For the "only if" part, item (1) follows Corollary 1, item (2) follows Lemma 6, item (3) follows from Lemma 8 and item (4) follows from Lemma 9. □

Theorem 6. *ws can be computed in polynomial time in the class of trees*

Proof. By Theorem 5, $ws(T)$ can be computed by dynamic programming for any tree T. □

3 Perpetual Graph Searching

In this section, we present algorithms to perpetually clear trees. For any path P, we fully characterize the set of integers k such that there is an algorithm for permanently clearing P with k searchers. In the case of trees or grids, we design algorithms for perpetual graph searching using some particular number of searchers. First, let us do some remarks.

Contrary to graph searching in a centralized setting, there are graphs G and $k < k'$ such that k searchers can perpetually clear G while k' searchers cannot. This is proved in next subsections and it is mainly due to the fact that increasing the number of searchers may make the size of the team even, which creates symmetrical configurations that cannot be "broken". Hence, in perpetual graph searching, the smallest number of searchers required to clear a graph is not sufficient anymore to characterize whether the graph can be cleared or not. Our goal is to (try to) describe the set of integers k such that k searchers can perpetually clear a graph in a given graphs' class. For any graph G, let $ps(G)$ be the set of integers k such that there is an algorithm for k searchers to perpetually clear G.

First note that $ps(G) \subseteq \{1, \cdots, |V(G)|\}$ and $|V(G)| \in ps(G)$ for any graph G.

3.1 Perpetual search in paths

As a start-up, we fully characterize $ps(P)$ for any path P.

Theorem 7. *Let P be any n-node path, then*

- $ps(P) = \{1\}$ *if* $n = 1$ *and* $ps(P) = \{1,2\}$ *if* $n = 2$*;*
- $ps(P) = \{3, \cdots, n\}$ *for any* $n \geq 3$ *even;*
- $ps(P) = \{3, 5, \cdots, 2k+1, \cdots, n\}$ *for any* $n \geq 3$ *odd.*

Proof. It is easy to see that at most 2 searchers cannot perpetually clear a path with at least 3 vertices.

Let $P = (v_1, \cdots, v_{2p+1})$ be an n-node path for some odd $2p+1 = n \geq 3$ and let $3 \leq 2r < n$. We show that $2r$ searchers cannot perpetually clear P. Indeed, the searchers are initially placed at $\{v_1, \cdots, v_r, v_{n-r+1}, \cdots, v_n\}$. Then, v_{p+1} can never be occupied by a searcher. Indeed, the adversary can schedule the moves such that, at any step when v_i is occupied for $i \leq p$, then v_{n-i+1} is also occupied. In particular, if a searcher occupying v_p moves to v_{p+1}, then the searcher occupying v_{p+2} will do the same and two searchers will occupy simultaneously v_{p+1}, a contradiction. Hence, $2r \notin ps(P)$ for any $2r < n$ in any n-node path P with n odd.

Let $n \geq 3$ and let $P = (v_1, \cdots, v_n)$ be a path. Let $3 \leq r < n$ be any integer if n is even and r is odd otherwise.

We show that, from any initial configuration, r searchers can reach the following configuration I. If $r = 2r'$, then $I = \{v_1, \cdots, v_{r'-1}, v_{r'+1}, v_{n-r'}, v_{n-r'+2}, \cdots, v_n\}$. If $r = 2r'+1$, then $I = \{v_1, \cdots, v_{r'}, x, v_{n-r'+1}, \cdots, v_n\}$ with $x \in \{v_{r'+2}, v_{n-r'-2}\}$. We show that I can be reached from any configuration in the case $r = 2r'+1$, the other cases

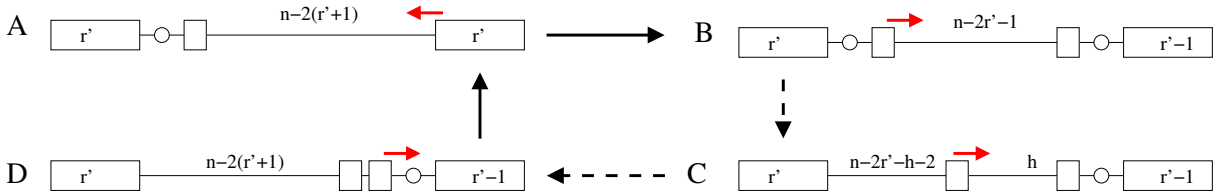

Figure 1: Perpetual strategy for $2r'+1$ agents in a n-node path, $n > 2r'+3$. An empty square represents one agent, an empty circle represents one node without agent, a rectangle with label ℓ represents ℓ searchers on consecutive vertices and a line with label h represents h consecutive vertices without searchers. In Configuration C, $h \geq 1$. Note that $n-2r'-h-1 \geq n-2(r'+1) > 1$ in Configurations A, C and D.

are similar. If v_i, $i \leq r'$ (resp., $i \geq n-r'+1$) is the smallest (resp., greatest) unoccupied vertex, then the searcher at v_j moves to v_{j-1} (resp., to v_{j+1}), where j smallest (resp., greatest) integer greater (resp., smaller) than v_i such that v_j is occupied. If $n = 2p+1$ and at some step, the vertices in $\{v_1, \cdots, v_{r'}, v_{p+1}, v_{n-r'+1}, \cdots, v_n\}$ are occupied, then the searcher at v_{p+1} chooses to move to one of its neighbors.

From the configuration I, it is possible to perpetually clear the path in the following way.

First, let us consider the case when $r < n \leq r-2$. If $n = 2p+1$ and $r = 2p$, then one searcher will oscillate between between v_p and v_{p+1} while all other vertices are occupied permanently. If $n = 2p+1$ and $r = 2p+1$ (resp., $n = 2p$ and $r = 2(p-1)$), then the searcher initially at v_{p-1} oscillates v_p and v_{p-1}, and the searcher initially occupying v_{p+3} (resp., v_{p+2}) oscillates v_{p+2} and v_{p+3} (resp., between v_{p+2} and v_{p+1}) while all other vertices are occupied permanently.

From now on, we assume that $n > r-2$. The algorithm for an odd number of searchers is described in Figure 1, and the algorithm for an even number of searchers are described in Figure 2. For instance, in the configuration C in the right of Figure 2, the $2(r'+1)$ searchers are occupying vertices $\{v_1, \cdots, v_{r'-1}, v_{r'+1}, v_{r'+h+1}, v_{n-r'-h-1}, v_{n-r'}, v_{n-r'+2}, \cdots, v_n\}$. In that configuration, the searcher at $v_{r'+h+1}$ must slide to $v_{r'+h+2}$ which is indicated by the red arrow above the empty square representing this searcher. Note that the configuration is not symmetric, so no ambiguity occurs to identify the agent that must move. Moreover, no two configurations are identical, so there are no ambiguity for identifying the configuration. Asynchronicity may occur in some configurations where two agents are allowed to move, e.g., in configurations B', C', D, E, F'. That is why it is possible to reach configuration B directly from configuration A without passing through configuration A' as indicated by the black arrows.

It is easy to check that the sequence of moves obtained by following these algorithms actually clear all edges of the path and that no two agents will reach the same vertex. □

3.2 Cases of impossibility

In this section, we give a characterization of some cases when k searchers cannot perpetually clear a tree. This characterization is based on a structural property of some nodes in a tree.

Basically, assume that a graph has two one-degree vertices u and v that are adjacent to the same vertex w. Then, if at some step u and v are occupied, then these two searchers are somehow "lost" to clear the remaining part of the graph. Indeed, the searcher at v and the one at u have the same vision of the graph, and if one decide to move to w, the other searcher must take the same decision. Hence, the adversary can schedule the moves such that both searchers occupy simultaneously w which is forbidden. In what follows, we generalize this fact.

Let G be a graph and $v \in V(G)$. We say that v is a *bad node* if there exists a node $u \in V(G)$, $u \neq v$, such that, any strategy in the minimalist CORDA model using two agents starting in u and v cannot visit all nodes of the graph without collision. That is, the adversary can schedule the moves such that both searchers occupy the same

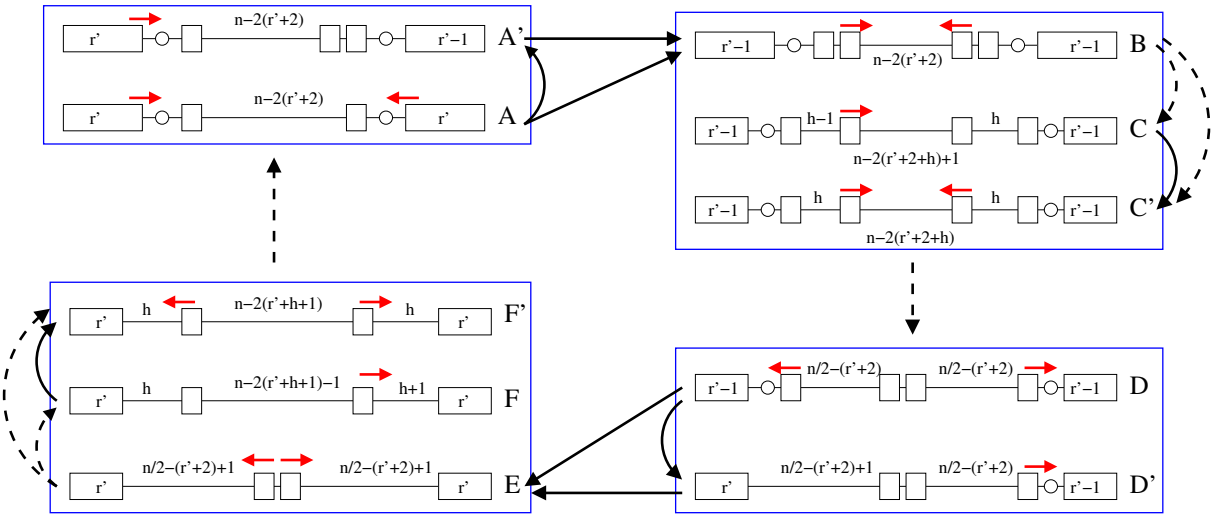

Figure 2: Perpetual strategy for $2(r'+1)$ agents in a n-node path, with n even and $n > 2(r'+2)$. An empty square represents one searcher, an empty circle represents one node without searcher, a rectangle with label ℓ represents ℓ searchers on consecutive vertices, and a line with label h represents h consecutive vertices without searchers. In configurations C and C', $h \geq 1$, and in Configurations F and F', $h \geq 2$. Blue rectangles only try to help the reading by dividing the algorithm in Phases.

node before all nodes are visited (possibly, the collision may occur in the last unvisited node). We show that the number of bad nodes of a graph G is closely related to $ps(G)$. We first need some definitions.

Two nodes $u, v \in V(G)$, $u \neq v$, are *twins* if there is an isomorphism $i : V(G) \to V(G)$ such that $i(u) = v$ and that preserves adjacency. Informally, this means that $G \setminus \{u\}$ is isomorphic to $G \setminus \{v\}$ and that this isomorphism maps the neighbors of u to the neighbors of v. For instance, any node has a twin in vertex-transitive graphs such as Cayley graphs []. In other words, in the case of a tree T, let $u, v \in V(T)$, $u \neq v$, and let $T_1, \cdots, T_\ell$ be the connected components of $T \setminus \{u\}$ with $v \in V(T_1)$ and let $S_1, \cdots, S_{\ell'}$ be the connected components of $T \setminus \{v\}$ with $u \in V(S_1)$, then u and v are twins if and only if, $\ell = \ell'$, and, for any $i \leq \ell$, there is an isomorphism f_i (i.e., preserving adjacency) from S_i to T_i that maps the neighbor of u in T_i to the neighbor of v in S_i and $f_1(u) = v$. In the context of the CORDA model, it means that a single agent cannot distinguish whether it stands at u or v when no other agents stand in the graph to brake symmetries. u and v are *odd twins* if there are twins and that any path from u to v has even length (containing an odd number of nodes). By definition of odd-tins vertices:

Lemma 11. *Let T be a tree and $u, v \in V(T)$ be two odd-twins. Let $u = v_1, v_2, \cdots, v_{k+1}, \cdots, v_{2k+1} = v$ be the path between u and v. Then, for any $i \leq k$, v_i and $v_{2k+1-i+1}$ are odd-twins.*

Lemma 12. *Let T_0 be the set of vertices of a tree T without odd-twin. Then T_0 induces a non-empty subtree (i.e., connected).*

Proof. The fact that $T_0 \neq \emptyset$ is easy.

Assume that $T_0 \subset V(T)$, otherwise the result holds. Let u and v be odd-twins in T and let P be the path connected them. Assume that u is not a leaf, hence, v is not a leaf. Moreover, let x be a neighbor of u in $T \setminus P$. By definition, the subtree S of $T \setminus \{u\}$ containing x is isomorphic to a subtree T of $T \setminus \{v\}$ not containing u. Let y be the neighbor of v in S. Therefore, x and y are odd-twins. This implies that, for any $v \in V(T)$ with an odd-twin in T, at most one component of $T \setminus \{v\}$ contains nodes without odd-twins which proves the lemma. □

Lemma 13. *Let T be a tree and $v \in V(T)$, v is a bad node if there is $u \in V(T)$, $u \neq v$, such that u and v are odd-twins.*

Proof. Let $u, v \in V(T)$ be odd-twins and let $u = v_1, v_2, \cdots, v_{k+1}, \cdots, v_{2k+1} = v$ be the path between u and v. Assume a searcher occupying u goes to its neighbor x, therefore, since the searcher at v executes the same algorithm, has a symmetric view and since the adversary schedules the moves of the two searchers following the worst case, it goes to a neighbor y of v such that x and y are odd-twins. In particular, for any $i \leq k$, when a searcher occupies v_i, then the other searcher occupies $v_{2k+1-i+1}$. By Lemma 11, since this two nodes are odd-twins, repeating the application of the algorithm will pursue this fact. Finally, when a searcher goes to v_{k+1}, the other searcher proceeds as well. □

Theorem 8. *Let T be a tree and let $B \subset V(T)$ be the set of vertices with an odd-twin in T. Let T_0 be the subtree induced by $V(T) \setminus B$ and let $i \geq 1$. If $i < ws(T)$, or i even and $i \leq |B|$, or $|B| \leq i < |B| + ws(T_0)$, then $i \notin ps(T)$.*

Proof. If $i < ws(T)$, it is clear that $i \notin ps(T)$ since any algorithm in the CORDA model with exclusivity property defines a sequence of sliding that clears the graph, i.e., defines a strategy without collision in a centralized setting.

Similarly, if $|B| \leq i < |B| + ws(T_0)$, consider the starting configuration where all nodes in B are occupied and the $i - |B|$ remaining searchers are spread arbitrarily. By definition of B, no searcher occupying a vertex of B can move. Indeed, either $v \in B$ has all its neighbors in B and therefore the searcher at v cannot move because its neighbors are occupied, or $v \in B$ has a unique neighbor, say w, not in B (by Lemma 12). In the latter case, w has at least one neighbor $u \in B$ that is an odd-twin of v. Therefore, if the searcher at v moves at w, the searcher at u does as well and two searchers occupy the same node which is forbidden. Therefore, there are at most $i - |B| < ws(T_0)$ remaining to clear T_0 which is not possible.

Now consider the case when $i \leq |B|$ and i is even. Therefore, the starting configuration consists in placing the searchers in such a way that, for any $v \in B$ occupied by a searcher there is an odd-twin u of v that is also occupied. This is clearly possible since i is even. Moreover, to avoid breaking the symmetries by the position of the searchers, we proceed as follows: first, the leaves of $V(T) \cap B$ are occupied, then their parents and so on (Nico : TO BE DETAILLED). Finally, each time a searcher moves from v to u, the at least one other searcher occupying an odd-twin of v will perform (because of the schedule of the adversary) the symmetric move by moving to an odd-neighbor of u (or to u itself). In particular, starting from such a configuration, no vertex in $T_0 = T \setminus B$ (not empty by Lemma 12) can be reached without collision. □

Note that above theorem generalizes the fact that, for any even $i < |P|$, $i \notin ps(P)$ for any path P with an odd number of vertices, as in Theorem 7.

The next technical lemma will be useful in the next subsection. We say that a tree T is *symmetric* if there is an edge $e \in E(T)$ such that the two components of $T \setminus \{e\}$, called the *half* of T, are isomorphic. Note that, if T is symmetric, the corresponding edge is unique.

Lemma 14. *Let T be a tree and let T_0 be its subtree without odd-twin nodes. If T_0 is not symmetric, there is an algorithm that assigns labels to any $v \in V(T_0)$ such that no two vertices receive the same label. If T_0 is symmetric, there is an algorithm that assigns labels to any vertex in T_0 such that no two vertices in the same half receive the same label.*

Proof. We use the fact that rooted trees can be labelled in such a way that two rooted trees receive the same label if and only if they are isomorphic. Indeed, a single node (which is the root) is labelled $\ell(\{r\}) = (1)$. Let T be a tree rooted in r, and $T_1, \cdots, T_h$ be the components of $T \setminus \{r\}$ such that, for any $i \leq h$, T_i is rooted in its vertex which is the neighbor of r in T. Then, $\ell(T) = (\ell(T_1), \cdots, \ell(T_h))$. Clearly, by induction on the height of T, ℓ defines uniquely rooted trees (up to isomorphism).

Recall that the centroid of a tree S is a vertex v such that any component of $S \setminus \{v\}$ has at most $|V(S)|/2$ nodes. Moreover, any tree has at most two centroids and if it has two centroids they are adjacent.

Let r be the centroid of T_0 if it has a unique centroid and, otherwise, let r_1, R_2 be its two centroids and let $e = \{r_1, r_2\} \in E(T_0)$ and S_1, resp., S_2 be the component of $T_0 \setminus \{e\}$ containing r_1, resp., r_2.

In the first case (T_0 has a unique centroid), each vertex $v \in V(T_0)$ receives a label $\ell(v)$ as defined previously with T rooted in r. Moreover, let $v = v_1, v_2, \cdots, r$ be the path from v to r. The final label of v is $\ell^*(v) = (\ell(v), (\ell(v_2), \cdots, \ell(r))$. We show that two distinct vertices v and u in $V(T_0)$ cannot have the same label, i.e., $\ell^*(v) \neq \ell^*(u)$. For purpose of contradiction, assume that u and v have the same label and let w be their nearest common ancestor in T rooted in r. Note that w must exist and must be different from both u and v, for otherwise $\ell^*(v) \neq \ell^*(u)$. Now, let $u' \in V(T_0)$, resp. $v' \in V(T_0)$, be the neighbor of w on the path from w to u, resp., from w to v. Therefore, $\ell^*(v') = \ell^*(u')$. Hence, the subtree rooted in v' is isomorphic to the subtree rooted in u' by definition of ℓ. Moreover, they have a common neighbor w. Therefore, u' and v' are odd-twins, contradicting the fact that $u', v' \in V(T_0)$.

In the second case (T_0 has two adjacent centroids r_1 and r_2), each vertex $v \in S_1$, resp., $v \in S_2$, receives a label $\ell(v)$ as defined previously with T rooted in r_1, resp., r_2. Moreover, let $v = v_1, v_2, \cdots, v_h = r_i$ be the path from v to r_i, where $r_i = r_1$ if $v \in S_1$ and $r_i = r_2$ otherwise. The final label of v is $\ell^*(v) = (\ell(v), (\ell(v_2), \cdots, \ell(r_i))$. As in the previous case, two distinct vertices in S_1, resp., in S_2, have received distinct labels. Finally, if $\ell^*(r_1) > \ell^*(r_2)$ (e.g., in lexicographical order), we take r_1^* as the root of T to redefine the labels of any vertex in $V(T_0)$. Again, no two vertices in $V(T_0)$ receive the same label.

The only case where the root cannot be determine deterministically occurs if $\ell^*(r_1) = \ell^*(r_2)$. However, in that case, it means that T (and T_0) is symmetric with respect to the edge $e = \{r_1, r_2\}$. Therefore, the lemma is proved. □

3.3 Perpetual search in trees

Let T be a tree, let B be the subset of its nodes that admit odd-twins and let T_0 be the subtree obtained by removing the vertices in B (see Lemma 12). Theorem 8 describes a subset of $\{1, \cdots, |V(T)|\}$, related to $|B|$ and $ws(T_0)$, that is not contained in $ps(T)$. In this section, we aim at determining some non-trivial subset of integers (not reduced to $\{|V(T)|\}$) that is contained in $ps(T)$. More precisely, we describe an algorithm that perpetually clear a tree when given a sufficient (but not too large) number of searchers.

In the CORDA model, nodes are anonymous, however, thanks to Lemma 14 it is possible for the agents, only using the topology of the tree given by the snapshot, to determine one unique or two adjacent roots in $V(T_0)$, and labels for the nodes of T_0. Moreover, if there is a unique root, all vertices of T_0 receive different labels, and otherwise, T_0 is symmetric with respect to the edge between the two roots and vertices in the same half of T_0 receive different labels. The key point here is that there is no ambiguity for defining root(s) and labels, i.e., all agents define it in the same way without coordination, just by looking at the topology. Therefore, in this section, we assume that the vertices of T_0 are labelled in that way.

Lemma 15. *There is an algorithm $\mathcal{A}$, in the minimalist CORDA model with exclusivity property, with the following properties. Let T be a tree and let B and T_0 defined as above. Let k be any integer with $|B| \leq k \leq |V(T)|$. Starting with k searchers in any configuration with exclusivity property, $\mathcal{A}$ allows to reach a configuration where all nodes in B are occupied.*

Proof. Recall that, while the nodes are anonymous, it is possible for the searchers to agree on one or two adjacent roots for T. The algorithm $\mathcal{A}$ proceeds as follows.

If one searcher has an unoccupied child in B, he goes to it. Clearly, no two searchers can arrive at the same node since one vertex cannot have two parents. Moreover, following $\mathcal{A}$, no searcher at some node in B goes to its parent. Therefore, no collision may occur during this phase.

If no searcher has an empty child in B and if some vertices in B are still not occupied, $\mathcal{A}$ proceeds as follows. Consider the set C of vertices of V_0 that are adjacent to some vertices of B. The vertices in C are unoccupied because otherwise, we would be in the previous case. Consider the set of searchers that minimize the distance to

some vertex of C and, among these searchers, take the one on the vertex with smallest label (recall that vertices of T_0 receive labels that are different, or, if T_0 is symmetric, one label may be shared by two nodes in different half of T_0). The chosen searchers go toward a vertex of C that is closest to them. There may be two chosen searchers, but in different halves of T_0, and therefore they cannot go toward the same vertex. Again, no collision may occur during this phase. □

Lemma 16. *There is an algorithm $\mathcal{A}$, in the minimalist CORDA model with exclusivity property but assuming identifiers for nodes, with the following properties. Let T be a tree whose node are labelled such that, either all vertices receive distinct labels, or T is symmetric and all nodes in the same half receive distinct labels. Let k be any integer such that $ws(T)+1 \leq k \leq |V(T)|$, and moreover, if T is symmetric, then either k is odd, or $2(ws(T)+1) \leq k \leq |V(T)|$. Starting with k searchers in any configuration with exclusivity property, $\mathcal{A}$ allows to perpetually clear T.*

Proof. NICO: TO BE WRITTEN WITH MUCH MORE DETAILS

First, let us assume that all vertices of T have distinct labels.

- Let $f \in V(T)$ be a leaf. First, we show that there is a uniquely defined strategy for clearing T using $ws(T)$ searchers, that finally place a searcher at f, and such that all configurations are distinct. We prove it by induction on $n = ws(T)$. It is clearly true if $n = 1$ since in that case, the tree is a path: the searcher starts from the leaf that is not f and goes to f. Assume the induction hypothesis for any $1 \leq n' < n$ and let T with $ws(T) = n$.

  Recall that, by Lemma 10, there is a particular path A in T called the avenue and that is uniquely defined (ties in its definition are broken using the labels of nodes). Similarly, each node of the avenue orders the branches according to ws, and ties are broken using labels. Therefore, the "decomposition" used in Lemma 10 can be uniquely defined (such that all searchers have the same definition of it). Let $A = (u_1, \cdots, u_p)$ be the avenue and let R^1 be the first subtree to be clear (as in the strategy proposed in Lemma 10). Note that $ws(R^1) = ws(T) - 1$ and let $x \in V(R^1)$ be the neighbor of u_1. By induction, there is a strategy that clears R^1 and finally moves a searcher to x and all configurations (in R^1) are different. Let T_1^1 be the next branch at u_1 to be cleared. Note that $ws(T_1^1) < ws(T)$, therefore the induction hypothesis applies. Let I_1^1 be the set of initial positions in T_1^1 of the strategy for T_1^1.

  Initially, $ws(T) - 1$ searchers are placed at their initial positions as in the strategy for R^1 and the last searcher occupies u_1. Then, the searchers in R^1 execute the strategy until R^1 is cleared and x is occupied. Until now, all configurations have been different. Then, the searcher at u_1 goes to the furthest vertex of I_1^1, and then the searcher at x goes to u_1. Then, among the searchers in R^1 that are closest to x, the one on the vertex with smallest label goes to x. Note that now, only $ws(T) - 2$ searchers occupy the vertices in R^1, therefore, again, all configurations are different from previous configuration. Then, the searcher at u_1 will go to the second furthest vertex in I_1^1, and so on until all vertices in I_1^1 are occupied and finally u_1 is occupied which is the "signal" to clear T_1^1.

  At some step, assume the searchers have achieved a (i, j_0)-situation (see Lemma 10) and moreover, searchers are occupying $I_{j_0}^i$, the initial positions of the strategy for $T_{j_0}^i$ (which exists by induction hypothesis) and finally u_i is occupied, which is the signal to clear $T_{j_0}^i$. Note that, no searchers occupy vertices in S^i, and, for $j > j_0$, no vertices in T_j^i are occupied but, if $j_0 > \lceil d_i/2 \rceil$, their single vertex v_j^i that is adjacent to u_i. Until now, all configurations have been distinct.

  Then, the strategy of $T_{j_0}^i$ is done until $T_{j_0}^i$ is clear and $v_{j_0}^i$ is occupied. This can be done because such a strategy exists by induction, moreover, there is no ambiguity because it is the first time u_i is occupied while $ws(T_{j_0}^i)$ searchers are in $T_{j_0}^i$.

  Then, if $j_0 = \lceil d_i/2 \rceil$, the vertices in $X = \{v_{j_0+1}^i, \cdots, v_{d_i}^i\}$ must be occupied by the searchers not in $T_{j_0}^i$. This can be done as in Lemma 10. To avoid ambiguity, the vertices of X are occupied by order of their

labels. Moreover, to decide which searcher must occupy such a vertex, we brake ties by distances and the labels of vertices.

Then, if $j_0 < d_i$, $ws(T^i_{j_0+1}$ searchers must go to their initial positions in $T^i_{j_0+1}$. This proceeds as when the searchers went from R^1 to T^1_1 above. Otherwise, if $j_0 = d_i$, the searcher at u_i goes to u_{i+1}. This case can be determined without ambiguity since it is the first time that u_i was occupied and the final positions of the strategy in $T^i_{j_0}$. In this case, once u_{i+1} is occupied, the searchers go to T^{i+1}_1 as previously.

The process goes on until S^p (the last branch at u_p) is clear. At this step, u_p is occupied and all other searchers are in S^p since $ws(S^p) = ws(T) - 1$. Then, either f, the leaf we want to reach at the end, belongs to S^p in which case it is reached at the end of the strategy in S^p (by induction), or, once the final configuration in S^p has been reached, the searcher at u_p goes to f.

- Then, using an extra searcher (compared with $ws(T)$), we show how to perpetually clear T. To do so, consider the first leaf v of R^1 that must be occupied by the previous strategy. The extra searcher oscillates between this leaf v and its neighbor. When the searcher is at v, the $ws(T)$ other searchers must execute the above strategy, in particular, they must be in one configuration corresponding to this strategy.

  If the searchers are not in such a configuration, a searcher reaches the neighbor of v, this is the "signal" for the other searchers to reach the starting configuration of the above strategy. Once, this configuration is achieved, the searcher at the neighbor of v goes to v.

  Also, after the last configuration of the clearing strategy is achieve, the searcher at v goes back to its neighbor to signal that the other searchers must go back to the starting configuration of the strategy.

- The last thing to mention is what to do with more searchers. If there are too many searchers, first the searchers will occupy the leaves with smallest label, recursively. Each time a searcher occupy such a leaf, the searcher is said *fixed* and the leaf is virtually removed from the tree. We go on until, the number of not fixed searchers corresponds to the $ws+1$ of the remaining tree. This is possible since each time we "remove" a leaf, the ws may decrease at most by one.

The problem when the tree is symmetric and labels are not all distinct is to brake the initial symmetry. If the number of searchers is odd, this initial symmetry is broken considering the half that contains the maximum number of searchers. That is, the half containing u_1 is decided in that way, and the avenue and the strategy may be decided uniformly by all searchers.

Otherwise, we use separately, two teams of $ws(T)+1$ searchers, each in one half of T. □

Theorem 9. *There is an algorithm $\mathcal{A}$, in the minimalist CORDA model with exclusivity property, with the following properties. Let T be a tree, let B be the subset of its nodes that admit odd-twins and let T_0 be the subtree $T \setminus B$. Let k be any integer such that $|B| + ws(T) + 1 \leq k \leq |V(T)|$, and moreover, if T_0 is symmetric, then either $k - |B|$ is odd, or $|B| + 2(ws(T)+1) \leq k \leq |V(T)|$. Starting with k searchers in any configuration with exclusivity property, $\mathcal{A}$ allows to perpetually clear T.*

Proof. The algorithm directly follows the two algorithms presented in Lemmas 15 and 16. While it remains some nodes in B that are not occupied, the searchers follows the first algorithm. Then, the remaining $k - |B|$ searchers clear T_0 using the second algorithm since T_0 can be viewed as a labelled tree. It is clear that the two algorithms cannot interfere whatever be the starting configuration. □

4 Conclusion and further work

In this work, we first propose a new variant of graph searching, called without collisions, and show it behaves differently from classical variants such as node-search. A remaining question is the one concerning the time-complexity of computing $ws(G)$ for any graph G. According to some of our results, ws is a $+1$-approximation

of *ns* in graphs with maximum degree 3, we conjecture that it is NP-complete (because the node search number is NP-complete in graphs with maximum degree 3 and not approximable up to an additive constant in general graphs). However, the polynomiality of the computation of *ws* would provide an additive constant approximation to the pathwidth in bounded degree graphs.

We also propose the first study of graph searching in the CORDA model. In this model with very weak hypothesis (minimalist CORDA model with exclusivity property), we give a partial characterization in the case of trees. More precisely, for any tree T with B its nodes with odd-twins and $T_0 = T \setminus B$, let $X = \{k \leq |V(T)| \ : \ k \leq ws(T),\ or\ k \leq |B|\ and\ even,\ or\ |B| \leq k < |B| + ws(T_0)\}$ and $Y = \{|B| + ws(T_0) < k \leq |V(T)| \ : \ if\ T_0\ symmetric\ either\ k\ odd\ or\ |B| + 2ws(T_0) + 1 < k\}$. Then, we prove that $Y \subseteq ps(T) \subseteq \{1, \cdots, |V(T)|\} \setminus Y$. In the case when T is a path with at least 3 nodes, then $ps(T) = \{3, \cdots, |V(T)|\}$ if T has an even number of vertices and $ps(T) = \{3, \cdots, |V(T)|\} \setminus Y$ otherwise. It would be interesting to characterize the remaining cases in trees and to study other graph classes.

References

[1] D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science*, 5:33–49, 1991.

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

[3] J. R. S. Blair, F. Manne, and R. Mihai. Efficient self-stabilizing graph searching in tree networks. In *Proceedings of 12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6366 of *LNCS*, pages 111–125. Springer, 2010.

[4] L. Blin, P. Fraigniaud, N. Nisse, and S. Vial. Distributed chasing of network intruders. *Theor. Comput. Sci.*, 399(1-2): 12–37, 2008.

[5] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. In *24th International Symposium on Distributed Computing (DISC)*, volume 6343 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2010.

[6] François Bonnet, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Asynchronous exclusive perpetual grid exploration without sense of direction. In *OPODIS*, pages 251–265, 2011.

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

[8] R.L. Breisch. *Lost in a Cave-applying graph theory to cave exploration*. 2012.

[9] J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. Network exploration by silent and oblivious robots. In *36th International Workshop on Graph Theoretic Concepts in Computer Science (WG)*, volume 6410 of *Lecture Notes in Computer Science*, pages 208–219, 2010.

[10] D. Coudert, F. Huc, and D. Mazauric. A distributed algorithm for computing and updating the process number of a forest. *Algorithmica*, 2011, to appear. URL `http://dx.doi.org/10.1007/s00453-011-9524-3`.

[11] G. D'Angelo, G. Di Stefano, and A. Navarra. Gathering of six robots on anonymous symmetric rings. In *SIROCCO*, volume 6796 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2011.

[12] N. Deo, M. S. Krishnamoorthy, and M. A. Langston. Exact and approximate solutions for the gate matrix layout problem. *IEEE Trans. on CAD of Integrated Circuits and Systems*, pages 79–84, 1987.

[13] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[14] J. A. Ellis, I. H. Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Inf. Comput.*, 113(1): 50–79, 1994.

[15] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Computing without communicating: Ring exploration by asynchronous oblivious robots. In *11th Int. Conf. on Princ. of Dist. Syst. (OPODIS)*, volume 4878 of *Lecture Notes in Computer Science*, pages 105–118. Springer, 2007.

[16] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theor. Comput. Sci.*, 411(14-15):1583–1598, 2010.

[17] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. How many oblivious robots can explore a line. *Inf. Process. Lett.*, 111 (20):1027–1031, 2011.

[18] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. In *Intelligent Vehicles Symposium*, pages 480–485, 2000.

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

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

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

[22] R. Klasing, A. Kosowski, and A. Navarra. Taking advantage of symmetries: Gathering of many asynchronous oblivious robots on a ring. *Theor. Comput. Sci.*, 411(34-36):3235–3246, 2010.

[23] R. Klasing, E. Markou, and A. Pelc. Gathering asynchronous oblivious mobile robots in a ring. *Theor. Comput. Sci.*, 390 (1):27–39, 2008.

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

[25] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The complexity of searching a graph. *J. Assoc. Comput. Mach.*, 35(1):18–44, 1988.

[26] R. Mihai and M. Mjelde. A self-stabilizing algorithm for graph searching in trees. In *Proceedings of 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 5873 of *LNCS*, pages 563–577. Springer, 2009.

[27] B. Monien and I. H. Sudborough. Min cut is np-complete for edge weighted treees. *Theor. Comput. Sci.*, 58:209–229, 1988.

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

[29] T. D. Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, volume 642, pages 426–441. 1978.

[30] G. Prencipe. Instantaneous actions vs. full asynchronicity : Controlling and coordinating a set of autonomous mobile robots. In *ICTCS*, pages 154–171, 2001.

[31] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[32] K. Skodinis. Computing optimal linear layouts of trees in linear time. *J. Algorithms*, 47(1):40–59, 2003.

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399