Theoretical Computer Science 607 (2015) 135–148

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

The complexity of zero-visibility cops and robber

Dariusz Dereniowski [a,1], Danny Dyer [b], Ryan M. Tifenbach [c,*,2], Boting Yang [d]

[a] Department of Algorithms and System Modeling, Gdańsk University of Technology, Gdańsk, Poland
[b] Department of Mathematics and Statistics, Memorial University of Newfoundland, St. John's, Newfoundland and Labrador, Canada
[c] Department of Mathematics and Statistics, University of Regina, Regina, Saskatchewan, Canada
[d] Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada

ARTICLE INFO

Article history:
Received 26 August 2014
Received in revised form 11 February 2015
Accepted 13 March 2015
Available online 18 March 2015

Keywords:
Cops & robber
Graph searching
Tree
Pursuit evasion

ABSTRACT

We consider the zero-visibility cops & robber game restricted to trees. We produce a characterisation of trees of copnumber k and We consider the computational complexity of the zero-visibility Cops and Robber game. We present a heavily modified version of an already-existing algorithm that computes the zero-visibility copnumber of a tree in linear time and we show that the corresponding decision problem is NP-complete on a nontrivial class of graphs.

1. Introduction

There are many pursuit-evasion models in graph theory where a collection of agents, known as cops, move through a graph attempting to capture an evader, known as the robber. In the classic cops and robber model of Nowakowski/Winkler/Quillot [15,17], the cops and robber have full information about each other's location and the structure of the graph, then alternate turns moving from a vertex to an adjacent vertex. In this model, the cops win if a strategy exists whereby they can guarantee they occupy that same vertex as the robber in a finite number of moves. Otherwise, the robber wins. For a thorough survey of this model, see [2].

The *zero-visibility cops and robber game* is a variant of this game in which the robber is invisible; that is, the cops have no information at any time about the location of the robber. The game for the cops becomes to guarantee that after some finite time they must have occupied the same position as the robber. Moreover, the goal becomes to do so with the fewest cops possible. This model was introduced by Tošić [19], who characterised those graphs with copnumber one, and computed the copnumber of paths, cycles, complete graphs and complete bipartite graphs.

Due to the invisibility of the robber, this pursuit-evasion game bears similarities to the edge-searching model of Parsons [16], which employs an "arbitrarily fast" invisible robber. Here, the minimum number of cops needed to capture a robber is essentially the pathwidth of the graph [5,9], and allowing recontamination (a robber being allowed to return to an edge that has previously been cleared) does not reduce the number of cops needed [1]. The authors have similarly bounded

* Corresponding author.
E-mail address: ryan.tifenbach@uregina.ca (R.M. Tifenbach).

[1] D. Dereniowski has been partially supported by Narodowe Centrum Nauki under contract DEC-2011/02/A/ST6/00201, and by a scholarship for outstanding young researchers founded by the Polish Ministry of Science and Higher Education.
[2] R.M. Tifenbach has been supported by a postdoctoral fellowship from the Atlantic Association for Research in the Mathematical Sciences.

http://dx.doi.org/10.1016/j.tcs.2015.03.022
0304-3975/

the zero-visibility case involving pathwidth, but have shown the surprising result that recontamination does help; that is, zero-visibility cops and robber is not monotonic [4].

Determining if the number of cops needed to catch the robber in the edge-searching model is less than k is NP-complete for arbitrary graphs and linear-time solvable for trees [13]. In the classic cops and robber model, since one cop is sufficient to catch a robber on any tree, the decision problem is trivial. However, it is a much more recent result that the corresponding decision problem for arbitrary graphs is actually EXPTIME-complete [10]. For zero-visibility cops and robber, a quadratic time algorithm for trees is known [18]. We improve on this result by finding a linear time algorithm.

The results presented herein are a sequel to those found in [4]. None of the proofs found in [4] are presented here; the reader is encouraged to refer the previous paper for further results on the subject of zero-visibility cops & robbers and for a much more extensive introduction. In Section 2 we introduce the notation used in this work. Our algorithmic result is divided into two parts. The first part, given in Section 3, is a constructive characterisation of trees having a given copnumber. This characterisation is then directly used in Section 4 to obtain our algorithm for computing the zero-visibility copnumber of a tree. In Section 5 we analyse a computationally hard case by proving that computing a zero-visibility copnumber of a starlike graph is NP-hard in general. The proofs in Section 5 are based on those found in [7].

2. Preliminaries

2.1. Graph-theoretic notation

The vertex and edge sets of a graph G are denoted V_G and E_G, respectively. We consider only *simple* graphs – graphs which contain no loops or multiple edges.

Let G be a graph. We use xy to denote the edge with endpoints x and y; when $xy \in E_G$ we say that x and y are *adjacent*. We use $x \sim y$ ($x \sim_G y$ if G needs to be specified) to denote the fact that x and y are distinct adjacent vertices and $x \simeq y$ to denote the fact that either $x \sim y$ or $x = y$. If $x \sim y$, then the vertices x and y are referred to as *neighbours*. The *degree* of a vertex is the number of neighbours it possesses. A *leaf* is a vertex that has degree equal to one; that is, a leaf has a single unique neighbour. For $W \subseteq V_G$, the *closed neighbourhood* of W in G, denoted $N_G[W]$, is the collection of vertices that are either contained in W or have a neighbour contained in W:

$$N_G[W] = \left\{ x \in V_G \middle| \exists y \in W \text{ such that } y \simeq x \right\}.$$

A *walk* of length r in G is a sequence of vertices $\omega = (\omega(0), \ldots, \omega(r))$ such that $\omega(s) \sim \omega(s+1)$ for $s = 0, \ldots, r-1$; we refer to such a walk as a walk from $\omega(0)$ to $\omega(r)$, and we refer to $\omega(0)$ and $\omega(r)$ as the *endpoints* of ω. For any two vertices $x, y \in V_G$, the *distance* between x and y in G, denoted $d_G(x, y)$, is the smallest length of a walk from x to y; if there are no walks from x to y, then $d_G(x, y) = \infty$. A *path* is a walk which repeats no vertices, a *closed walk* is a walk whose endpoints are identical and a *cycle* is a closed walk which repeats no vertices other than its endpoints.

A *subgraph* of a graph G is a graph H whose vertex and edge sets are subsets of the vertex and edge sets of G. Evidently, if H is a subgraph of G, then for all $x, y \in V_H$, $d_H(x, y) \geq d_G(x, y)$. If H is a subgraph of G such that for all $x, y \in V_H$, $d_H(x, y) = d_G(x, y)$, we refer to H as an *isometric subgraph* of G. Let $x \in V_G \setminus V_H$. We say that x is *adjacent* to a subgraph H if x is adjacent to at least one vertex $y \in V_H$. The distance from a vertex x to a subgraph H in G is the minimum distance from x to a vertex in H:

$$d_G(x, H) = \min \left\{ d_G(x, y) \middle| y \in V_H \right\}.$$

For a nonempty subset $W \subseteq V_G$, the *induced subgraph* on W, denoted $G(W)$, is the subgraph of G whose vertex set is W and which contains every edge of G with both endpoints in W:

$$E_{G(W)} = \left\{ xy \in E_G \middle| x, y \in W \right\}.$$

Given a set of vertices and/or edges X, the subgraph of G obtained by deleting X from G is the graph with vertex set $V_G \setminus X$ and edge set $E_G \setminus X$ and is labelled $G - X$. When X contains a single vertex or edge, say $X = \{v\}$ or $X = \{e\}$, we use $G - v$ or $G - e$ in place of $G - X$.

2.2. Zero-visibility cops & robber

We consider the pursuit game referred to as *zero-visibility cops & robber* previously examined in [4,18,19].

The game is played by two players, the *cop* and the *robber*, on a graph G; we refer to both players and their respective pieces as cops and robbers. The game begins with the cop player placing one or more cop pieces on vertices of G followed by the robber placing a single robber piece on a vertex of G unknown to the cop player. Beginning with the cop, the players then alternate turns; on a player's turn, he may move each of his pieces along an edge to a vertex adjacent to its current position, or leave any pieces where they are. The game ends with victory of the cop player if the robber piece and a cop piece ever simultaneously occupy the same vertex; the robber's goal is to avoid this situation indefinitely. The position of

the robber piece is kept secret from the cop player until the game ends, although the cop may at times be able to deduce (by examining the history of the movements of cop pieces) the possible locations of the robber.

We use the following terminology, much of which was introduced in [4], to analyse the game. By a *copwalk* we mean a sequence of vertices $(\omega(0), \ldots, \omega(r))$ that describes a movement of a piece in the cops and robber game; i.e., $\omega(s)$ is the position of the piece after its controller has taken s turns, and so for $s = 0, \ldots, r-1$, $\omega(s+1) \simeq \omega(s)$. Let G be a connected graph. A *strategy* on G of length T and order k is a collection $\mathcal{L} = \{\omega_i\}_{i=1}^{k}$ of k copwalks of length T. We also denote

$$\mathcal{L}_s = \{\omega_1(s), \ldots, \omega_k(s)\} \text{ for } s \geq 0.$$

A strategy of order k and length T gives us a sequence of prescribed moves for a cop player utilising k cops; we might imagine that the cop player following such a strategy forfeits if he has not won after T turns. We refer to a strategy as *successful* if it guarantees victory for the cop player.

Evidently, a strategy $\mathcal{L}$ is successful if and only if for every copwalk α of length $T-1$ in G, there is $\omega_i \in \mathcal{L}$ and $s \leq T-1$ such that $\alpha(s) \in \{\omega_i(s), \omega_i(s+1)\}$ (the robber that follows α may be caught by either moving onto a cop, which occurs when $\alpha(s) = \omega_i(s)$, or by having a cop move onto it, which occurs when $\alpha(s) = \omega_i(s+1)$).

Rather than tracking individual moves by a robber piece, we will view this game as an exercise in graph cleaning, played by a single player:

1. At the beginning of the game, a number of cop pieces are placed on vertices in the graph. Every occupied vertex is marked as "clean" and every unoccupied vertex is marked as "dirty".
2. The cop player makes a series of turns. During each turn, the player may move each piece along an edge to an adjacent vertex or leave it where it is.
3. Every time a dirty vertex is occupied by a cop it becomes clean.
4. In between each of the cop player's turns, every clean vertex that is unoccupied and is adjacent to a dirty vertex becomes dirty.

At each point during the game, the dirty vertices are those vertices that could contain the robber, given that he has not yet been caught. We refer to the point in between the cop's turns where clean vertices may become dirty as *recontamination*. We note Proposition 2.1, which follows directly from the relevant definitions.

Proposition 2.1. *For any graph G, a strategy $\mathcal{L}$ on G of length T is successful if and only if following $\mathcal{L}$ results in every vertex being clean after T turns.* □

Let G be a graph and let $\mathcal{L} = \{\omega_i\}_{i=1}^{k}$ be a strategy on G of length T. We define two sequences of sets of vertices that track the dirty vertices in G throughout the game: for each $s \in \{1, \ldots, T\}$,

1. let $\mathcal{R}_s$ be the set of vertices of G that are dirty at the beginning of the cop's s-th turn (before he makes any moves), and
2. let $\mathcal{S}_s$ be the set of vertices of G that are dirty immediately after the cop takes his s-th turn (before possible recontamination).

It is clear that $\mathcal{R}_1 = V_G \setminus \mathcal{L}_0$. The relevant rules of the game imply that for $1 \leq s \leq T$,

$$\mathcal{S}_s = \mathcal{R}_s \setminus \mathcal{L}_s \text{ and } \mathcal{R}_{s+1} = N_G[\mathcal{S}_s] \setminus \mathcal{L}_s.$$

Evidently, a strategy of length T is successful if and only if $\mathcal{S}_T$ is empty. We will further assume that the strategies we consider contain no superfluous moves; thus, if $\mathcal{L}$ is a successful strategy of length T, we assume that $\mathcal{S}_T$ is empty and that $\mathcal{R}_T$ is nonempty.

The *zero-visibility copnumber* of a graph G, denoted by $c_0(G)$, is the minimum number of cops required to guarantee capture of the robber in a finite number of turns. Thus, $c_0(G) = k$ if there is a successful strategy of order k on G and there are no successful strategies of order $k-1$ on G.

In [4], the problem of allowing infinite strategies was considered – strategies made up of copwalks of infinite length. It was also shown in [4] that allowing such strategies does not change the zero-visibility copnumber of a graph; that is, if there is a successful strategy of order k and infinite length on G then there is a successful strategy of order k and finite length on G. For this reason, each strategy we consider in this work is finite.

3. A constructive characterisation of the zero-visibility copnumber of a tree

We start by recalling the following result:

Lemma 3.1. *(See [18].) Let G be a graph and let H be an isometric subgraph of G; then $c_0(H) \leq c_0(G)$.*

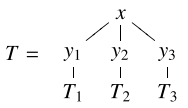


Fig. 1. The tree T derived from T_1, T_2, T_3.

We provide a brief sketch of a proof. A strategy is deterministic, rather than dynamic – the cop player makes no decisions or observations, he simply follows a collection of walks. We imagine that H is an isometric subgraph of G and we modify a successful strategy on G in the following manner. Any cops which never enter H during the strategy on G are simply discarded. Whenever a cop leaves H, we simply replace the copwalk he follows in $G - H$ with one in H of equal length. At the point that the successful strategy on G catches the robber who has not left H, it has been with a cop currently in H, which is still there under this modified strategy.

Let G be a connected graph. If $G - e$ is disconnected, where $e \in E_G$, then e is called a *cut edge* in G. Not every graph contains cut edges. Let G be a connected graph that contains a cut edge e and let H_1 and H_2 be the two connected components of $G - e$. It is straightforward to show that H_1 and H_2 are isometric subgraphs of G and so Lemma 3.1 implies that both $c_0(H_1)$ and $c_0(H_2)$ are less than or equal to $c_0(G)$. Lemma 3.2, proved in [4], further examines this situation.

Lemma 3.2. *(See [4].) Let G be a connected graph with a cut edge e and let H be one of the two connected components of $G - e$. Let $\mathcal{L}$ be a successful strategy of length T on G. Then, at some point in the strategy, at least $c_0(H)$ cops are simultaneously present in H; that is, there is $s \leq T$ such that*

$$|\mathcal{L}_s \cap V_H| \geq c_0(H).$$

In Lemma 3.3, we show that the robber territory can never get too far distant from vertices occupied by cops.

Lemma 3.3. *Let G be a connected graph and let $\mathcal{L}$ be a strategy on G. Let X be the set of vertices containing cop pieces and let Y be the set of dirty vertices after any number of moves by the cop player. Then, for every connected component H of $G(Y)$ there is a vertex in $x \in X$ at distance at most 2 from H.*

Proof. At the start of the cop player's s-th turn, we have $X = \mathcal{L}_{s-1}$ and $Y = \mathcal{R}_s$; at the end of the cop's s-th turn we have $X = \mathcal{L}_s$ and $Y = \mathcal{S}_s$. We will show, via induction on s, that if the set of dirty vertices in question is nonempty,

1. for every connected component H of $G(\mathcal{R}_s)$, there is $x \in \mathcal{L}_{s-1}$ adjacent to H; and
2. for every connected component H of $G(\mathcal{S}_s)$, there is $x \in \mathcal{L}_s$ at distance at most 2 from H.

For $s = 1$ the first statement is clear, as $\mathcal{R}_1 = V_G \setminus \mathcal{L}_0$ implies that V_G is the disjoint union of $\mathcal{R}_1$ and $\mathcal{L}_0$.

So, suppose that $s \geq 1$, that $\mathcal{S}_s$ is nonempty and that the first statement holds. Let H be a connected component of $G(\mathcal{S}_s)$. Since $\mathcal{S}_s = \mathcal{R}_s \setminus \mathcal{L}_s$, H is an induced subgraph of a connected component H' of $G(\mathcal{R}_s)$. If one or more cops moved onto H' during the s-th turn, then H is adjacent to at least one of these cops. If not, we in fact have $H = H'$ and, via the first statement, some cop piece was adjacent to H before the cop's s-th turn. This cop piece must be at a distance of at most 2 from H after the cop's s-th turn.

Thus, for each value of $s \geq 1$, if the first statement is true so is the second.

Now, suppose that the second statement holds true for $s \geq 1$ and that $\mathcal{R}_{s+1}$ is nonempty. Let H be a connected component of $G(\mathcal{R}_{s+1})$. Since $\mathcal{S}_s \subseteq \mathcal{R}_{s+1} = N[\mathcal{S}_s] \setminus \mathcal{L}_s$, there is a connected component H' of $G(\mathcal{S}_s)$ such that H' and $N[V_{H'}] \setminus \mathcal{L}_s$ are both subgraphs of H. Let $x \in \mathcal{L}_s$ have minimal distance from H'. Via the inductive hypothesis, $d(x, H') \leq 2$. If x is adjacent to H' it is also adjacent to H. If x is at distance 2 from H', let x' be adjacent to both x and H'. We then have $x' \notin \mathcal{L}_s$ and $x' \in N[V_{H'}]$, so $x' \in N[V_{H'}] \setminus \mathcal{L}_s \subseteq V_H$. Thus, x is adjacent to H (since $x \sim x'$).

Therefore, when the second statement holds true for some value of $s \geq 1$, the first statement holds true for $s' = s + 1$. □

Let $\mathcal{L}$ be a strategy on G and let $xy \in E_G$. We say that a cop *vibrates on xy* in steps $s_1, \ldots, s_2$, where $0 \leq s_1 < s_2$, if $\omega(i) \in \{x, y\}$ for each $i \in \{s_1, \ldots, s_2\}$ and $\omega(i) \neq \omega(i+1)$ for each $i \in \{s_1, \ldots, s_2 - 1\}$, where ω is the copwalk of the cop. Informally speaking, the cop moves back and forth between x and y in steps $s_1, \ldots, s_2$.

We now define a tree construction that plays a crucial role in our arguments. Let $k \geq 1$ and let T_1, T_2, T_3 be trees such that $c_0(T_i) = k$ for each $i \in \{1, 2, 3\}$. A tree T is obtained in the manner shown in Fig. 1, where the edge that joins each T_i to the remainder of T can be incident to any vertex in T_i. The vertex x will be called the *central vertex* of T and each of the new vertices y_i is the *connecting vertex* of T_i, $i \in \{1, 2, 3\}$. We then say that T is *derived* from T_1, T_2 and T_3.

Let T be a tree and let $x \in V_T$. We define $F_T(x) = T - N_T[x]$ to be the forest obtained by deleting x and each of its neighbours from T. We note that every connected component of $F_T(x)$ is a subtree of T that is adjacent to a unique neighbour of x.

We now describe a cleaning procedure, given in Algorithm 1, that we often use. The input to the procedure is a tree T, a vertex v of T and the number of cops k. We refer to the procedure as a *standard cleaning* of T from v.

Algorithm 1 Standard cleaning of T from v with k cops.

```
Initially place all k cops on v.
Label the neighbours of v as v_1, ..., v_ℓ.
for i := 1, ..., ℓ do
    Let the k-th cop start vibrating on v v_i.
    for each tree H in F_T(v) having a vertex adjacent to v_i do
        Clean H using the first k − 1 cops.
    end for
    Let the first k − 1 cops return to v.
    Wait (at most one turn) so that all cops are on v.
end for
```

Note that the procedure does not prescribe how the subtrees H are cleaned. We will refer to this procedure usually to obtain some upper bounds on the number of cops some trees require.

We have the following observation.

Observation 3.4. *Let T be a tree, let $v \in V_T$ and let $k \geq 1$. If $c_0(H) \leq k-1$ for each connected component H of $F_T(v)$, then a standard cleaning of T from v with k cops produces a successful strategy on T. Moreover, this strategy is such that v is visited at least every second turn and the strategy ends with every cop on v.* □

Lemma 3.5. *Let $k \geq 1$. If T is derived from T_1, T_2, T_3, where $c_0(T_i) = k$ for each $i \in \{1,2,3\}$, then $c_0(T) = k+1$.*

Proof. Let x be the central vertex of T and let y_i be the connecting vertex of T_i for each $i \in \{1,2,3\}$. Observation 3.4 immediately implies that $c_0(T) \leq k+1$.

So, we need to show that T cannot be cleaned using only k cops. We proceed by contradiction; suppose that $\mathcal{L}$ is a successful strategy of order less than or equal to k on T. By Lemma 3.2, a subtree T_i, $i \in \{1,2,3\}$, can be cleaned only when k cops are simultaneously present in T_i in some step s, i.e., $\mathcal{L}_s \subseteq V_{T_i}$. Take s to be the last step after which all k cops are present in the same subtree T_{i_1} and a vertex of some other subtree T_{i_2} is dirty; by Lemma 3.2, such an s must exist.

By Lemma 3.3, there is a (connected) subtree T' of T such that immediately after the cop's s-th turn,

(i) $V_{T_{i_2}} \cap V_{T'}$ $(i_2 \neq i_1)$ is nonempty;
(ii) every vertex in $V_{T'}$ is dirty; and
(iii) there is a cop at distance at most 2 from T'.

The subtree T' contains a path joining a vertex in T_{i_2} to a vertex of distance at most 2 from T_{i_1}; any such path contains the central vertex x and so $x \in V_{T'}$. Therefore, after the cop's s-th turn, the central vertex x is dirty and every cop is in T_{i_1}. It is clear that every vertex v in the two subtrees T_j with $j \neq i_1$ will be recontaminated (or, is already dirty) before a cop can visit v. In order to clean T, all k cops must enter each of the two subtrees T_j with $j \neq i_1$, after the s-th step, and clean them each anew. The first time they do so, there will be dirty vertices in the other such subtree. This contradicts the assumption that step s was the last step at which this occurred. □

Theorem 3.6. *Let T be a tree and let $k \geq 1$. Then, $c_0(T) \geq k+1$ if and only if there is $x \in V_T$ with at least three distinct neighbours that are each adjacent to a connected component H of $F_T(x)$ with $c_0(H) \geq k$.*

Proof. In effect, we claim that $c_0(T) \geq k+1$ if and only if T has an induced subgraph that is derived from T_1, T_2 and T_3 (that is, of the form shown in Fig. 1), where each T_i has $c_0(T_i) \geq k$.

First, suppose that there is $x \in V_T$ with three distinct neighbours y_1, y_2 and y_3 that are each adjacent to a connected component of $F_T(x)$ with zero-visibility copnumber greater than or equal to k. Let T_1, T_2 and T_3 be connected components of $F_T(x)$ such that for each $i \in \{1,2,3\}$, y_i is adjacent to T_i and $c_0(T_i) \geq k$.

It is simple to show that for any tree H with $c_0(H) \geq k+1$, we can obtain a subtree H' with $c_0(H') = k$ by deleting leaves, one at a time, from H. Any tree with copnumber greater than or equal to $k+1 \geq 2$ must contain at least two leaves. Thus, for each $i \in \{1,2,3\}$, we can form a subtree T_i' of T_i that has $c_0(T_i') = k$ by successively deleting leaves and we can do so without deleting the vertex adjacent to y_i. (If the neighbour of y_i in T_i is a leaf in T_i, we simply delete one of the other leaves.) Let

$$T' = T\left(\{x, y_1, y_2, y_3\} \cup V_{T_1'} \cup V_{T_2'} \cup V_{T_3'}\right).$$

The subtree T' is derived from T_1', T_2' and T_3'; by Lemma 3.5, $c_0(T') = k+1$. As T' is an isometric subgraph of T, Lemma 3.1 implies $c_0(T) \geq k+1$.

We now prove the converse statement via contraposition. Suppose that $k \geq 1$ and that T is a tree such that every $x \in V_T$ has at most two distinct neighbours that are adjacent to connected components H of $F_T(x)$ with $c_0(H) \geq k$. We describe a successful strategy on T that utilises k cops.

If there exists $x \in V_T$ such that every connected component H of $F_T(x)$ has $c_0(H) \leq k-1$, then Observation 3.4 implies that T can be cleaned using k cops, i.e., $c_0(T) \leq k$.

Otherwise, suppose that there is $x \in V_T$ and a connected component H of $F_T(x)$ with $c_0(H) \geq k$. Let x_1 and x_2 be neighbouring vertices in T such that

(a) there is at least one connected component H of $F_T(x_2)$ that has $c_0(H) \geq k$ and x_1 is the neighbour of x_2 that is adjacent to H; and
(b) over all ordered pairs of neighbours (x_1, x_2) which satisfy (a), the number of vertices in the connected component of $T - x_2$ which contains x_1 is minimal.

We first claim that every connected component H' of $F_T(x_1)$ that is not adjacent to x_2 has $c_0(H') \leq k-1$. Suppose we can find a connected component H' of $F_T(x_1)$ that is not adjacent to x_2 and has $c_0(H') \geq k$; let x_0 be the neighbour of x_1 that is adjacent to H'. Then, $(x_1', x_2') = (x_0, x_1)$ satisfies (a), above. However, the connected component of $T - x_2'$ which contains x_1' is a subtree of the connected component of $T - x_2$ which contains x_1; this contradicts the fact that x_1 and x_2 satisfy (b), above.

So, let y be a neighbour of x_1 that is not equal to x_2. The connected components H of $T - y$ which do not contain x_1 are connected components of $F_T(x_1)$ that are not adjacent to x_2 and so each such connected component has $c_0(H) \leq k-1$. Thus, our strategy begins in the following way: we let T' be a connected component of $T - x_2$ that contains x_1 and we apply Algorithm 1 with input T' and $v = x_1$. When Algorithm 1 terminates, T' is clean and all k cops are on x_1; we then move all k cops from x_1 to x_2.

So, at this point, we have a path $x_1 \cdots x_r$ (initially $r = 2$) such that

(i) all k cops are on x_r;
(ii) there is at least one connected component H of $F_T(x_r)$ which is adjacent to x_{r-1} and has $c_0(H) \geq k$; and
(iii) the subtree of $T - x_r$ which contains x_{r-1} is clean.

We will now show that either it is straightforward to clean the remainder of the graph T, or that we can extend our path by one vertex while maintaining conditions (i) through (iii) above. The vertex set of T being finite, such a process must eventually clean all of T.

Let Y be the set of neighbours of x_r that are *not* adjacent to a connected component H of $F_T(x_r)$ with $c_0(H) \geq k$. Let T'' be the subtree of T containing x_r, Y and every subtree of $F_T(x_r)$ that is adjacent to a member of Y. We apply Algorithm 1 with inputs T'' and x_r.

By assumption, every vertex $x \in V_T$ has at most two neighbours that are adjacent to connected components H of $F_T(x)$ with $c_0(H) \geq k$. We have already identified one such neighbour of x_r, namely, x_{r-1}. If x_r has no other such neighbours, then we have $Y \cup \{x_{r-1}\} = N[x_r]$ and we can see that the graph is now clean. If there is a second such neighbour of x_r, we label it x_{r+1} and then move all k cops from x_r to x_{r+1}. After this move, we have extended the length of the path above by one while maintaining conditions (i) through (iii).

This completes the proof of the theorem. □

3.1. Tree minors and critically k-copwin trees

Let G be a graph. An *edge contraction* is a manner in which a new graph is constructed from G. By contracting an edge xy of G, we form a new graph H by removing x and y and adding a new vertex x' adjacent to every neighbour of x or y:

$$V_H = \{x'\} \cup (V_G \setminus \{x, y\}) \quad \text{and} \quad E_H = \left\{x'v \;\middle|\; xv \in E_G \text{ or } yv \in E_G\right\} \cup (E_G \setminus \{xy\}).$$

In other words, an edge contraction of the edge xy is obtained by equating the vertices x and y.

A *minor* of a graph G is a graph H that can be obtained from G via some sequence of edge contractions, edge deletions and/or vertex deletions. Minors are a generalisation of the concept of a subgraph that preserve certain topological properties of a given graph.

When G is a tree, any connected minor of G is, itself, a tree and can be obtained by a sequence of edge contractions (if the end result is to be connected, any edge or vertex deletions can be replaced with appropriate subsequences of contractions).

Theorem 3.7. *Let T be a tree and let H be a connected minor of T. Then, $c_0(H) \leq c_0(T)$.*

Proof. We will prove, by induction on $n = |V_T|$, that if H is formed by at most one edge contraction from T, then $c_0(H) \leq c_0(T)$; this, together with Lemma 3.1, is sufficient to show the claim.

It can be shown that trees on 6 or fewer vertices have $c_0(T) = 1$ implying that the statement holds for T with $|V_T| \in \{1, \ldots, 6\}$. So, suppose that $n \geq 6$ and that the claim holds for trees on n or fewer vertices. Let T be a tree on $n+1$ vertices and let H be formed from T by a single edge contraction.

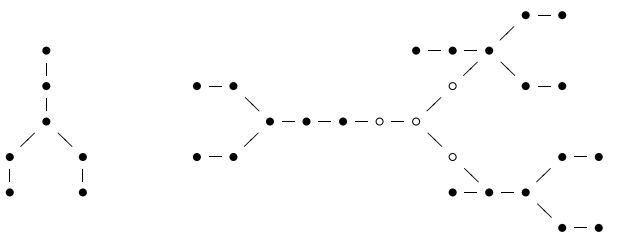
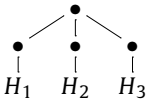


Fig. 2. The single tree contained in $\mathcal{T}_2$ (left) together with one of the 10 trees contained in $\mathcal{T}_3$ (right). In the tree on the right, the 3 copies of the tree in $\mathcal{T}_2$ are shown with dots; the root and its children are shown with circles.

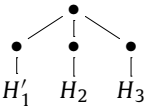
If $c_0(H) = 1$, we have $c_0(T) \geq c_0(H)$; so, suppose that $c_0(H) = k + 1$, where $k \geq 1$. Then, via the discussion in the third paragraph of the proof of Theorem 3.6, there exists a subtree H' of H, with the following form, where each H_i has $c_0(H_i) = k$.

$H' =$ 

It must be that either H' is a subtree of T, or that there is a subtree T' of T such that H' is formed from T' via a single edge contraction.

If H' is a subtree of T, then, due to Lemma 3.1, $c_0(T) \geq c_0(H') = k + 1 = c_0(H)$.

If H' is not a subtree of T, then, possibly by relabelling the subtrees H_i, T contains a subtree T' of the following form, where the subtree H'_1 has H_1 as a minor:

$T' =$ 

By the inductive hypothesis, $c_0(H'_1) \geq c_0(H_1) = k$. Theorem 3.6, together with Lemma 3.1, then implies that

$$c_0(T) \geq c_0(T') \geq k + 1 = c_0(H).$$

This completes the proof. □

Let T be tree. We say that T is *copwin-critical* if the only connected minor of T that has zero-visibility copnumber equal $c_0(T)$ is T itself.

We define a sequence of families of trees that will characterise copwin-critical trees. The family $\mathcal{T}_1$ consists of the tree on one vertex. For $k \geq 1$, the family $\mathcal{T}_{k+1}$ consists of all trees T derived from $T_1, T_2, T_3 \in \mathcal{T}_k$ (not necessarily distinct).

The family $\mathcal{T}_2$ consists of a single tree, given in Fig. 2.

There are three distinct (nonisomorphic) ways in which the tree in $\mathcal{T}_2$ could be connected to another tree by a single edge – this edge could be joined to one of the 3 leaves, to one of the 3 vertices adjacent to the leaves, or to the central vertex. Thus, there are 10 distinct trees contained in $\mathcal{T}_3$. We show one of the members of $\mathcal{T}_3$ in Fig. 2; in this tree, we see each of the 3 possible ways of joining the tree in $\mathcal{T}_2$.

The number of trees in $\mathcal{T}_k$ grows very rapidly with k – it is clear that

$$|\mathcal{T}_{k+1}| > \binom{|\mathcal{T}_k| + 2}{3}.$$

(The number of ways of choosing s elements, with repetition allowed, from a collection of r elements is $\binom{r+s-1}{s}$.) This is, in fact, a very weak lower bound, as it ignores the distinct manners in which three trees from $\mathcal{T}_k$ can form a tree from $\mathcal{T}_{k+1}$. A somewhat involved counting argument shows that $\mathcal{T}_4$ consists of 204156 distinct trees.

Theorem 3.8 says that the zero-visibility number of a tree T is the largest k such that $\mathcal{T}_k$ contains a minor of T. The proof is a simple induction utilising Theorems 3.6 and 3.7, and is omitted.

Theorem 3.8. *Let T be a tree; then, $c_0(T) \geq k$ if and only if there is $T' \in \mathcal{T}_k$ such that T' is a minor of T.* □

4. An algorithm for calculating the zero-visibility copnumber of a tree

We give a modified version of the algorithm first presented in [5]. There, a graph parameter referred to as vertex separation is investigated. The vertex separation of a graph is equal to its pathwidth; see [5,9].

It is shown in [5] that for $k \geq 1$, the vertex separation of a tree T is greater than or equal to $k+1$ if and only if there is a vertex $v \in V_T$ such that at least three connected components of $T - v$ have vertex separation greater than or equal to k. Our Theorem 3.6 closely mirrors this result. The similarity of our result allows us to utilise a very similar methodology in calculating the zero-visibility copnumber of a tree.

The first classification of pursuit parameters on trees appears in [16]. The characterisation there is, again, very similar to our own and that found in [5].

We will utilise, extensively, the concept of a rooted tree in this section. A *rooted tree* is a tree T where some vertex $u \in V_T$ has been marked as the *root* of T. The root of a tree T is denoted by $\mathtt{r}(T)$. For every vertex $v \in V_T \setminus \{\mathtt{r}(T)\}$, there is a unique path joining v and u; the *parent* of v is the sole neighbour of v in this unique path. If v' is the parent of v, we refer to v as a *child* of v'.

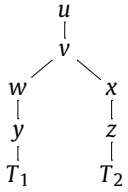
Let T be a rooted tree. For $x, y \in V_T$, we say that $x \prec y$ if there is a path in T

$$x = x_0 \sim x_1 \sim \ldots \sim x_l = y$$

where x_k is a child of x_{k+1} for each $k \in \{0, \ldots, l-1\}$. We say that $x \preceq y$ if $x \prec y$ or $x = y$. The relation $\preceq$ is clearly a partial order.

For each $v \in V_T$, $T[v]$ is the rooted subtree of T with root v on the vertex set $\{x \mid x \preceq v\}$; we refer to $T[v]$ as the *descendant subtree* at v. If $X \subseteq V_T$ for a rooted tree T, then each connected component T' of $T - X$ is treated as a rooted tree with the root $v \in V_{T'}$ such that $x \preceq v$ for each $x \in V_{T'}$, where $\preceq$ is the relation on the vertex set in T. We denote for brevity $\overline{T} = T - \mathtt{r}(T)$ for a rooted tree T. Thus, each tree in $\overline{T}$ is rooted at the vertex adjacent to $\mathtt{r}(T)$ in T.

A vertex v in a rooted tree T is *k-pre-branching* if $\mathtt{c}_0(T[v]) = k$ and v has a child u with $\mathtt{c}_0(T[u]) = k$. We refer to a vertex v as *weakly k-branching* if $\mathtt{c}_0(T[v]) = k$ and v has at least two children that are k-pre-branching. We refer to a vertex v as *k-branching* if $\mathtt{c}_0(T[v]) = k$ and v has a weakly k-branching child.



In the diagram above, if $\mathtt{c}_0(T[u]) = \mathtt{c}_0(T[y]) = \mathtt{c}_0(T[z]) = k$, then w and x (and also u and v) are k-pre-branching, v is weakly k-branching and u is k-branching.

Let T' be a graph whose each connected component is a rooted tree. If T' has a k-branching vertex, then we define $\mathtt{b}^k(T')$ to be one of its k-branching vertices; otherwise we write $\mathtt{b}^k(T') = \perp$ for brevity. We define the following counters for k-branching and k-pre-branching vertices of T':

$$\#^k_{\mathrm{b}}(T') = \left|\left\{T \;\middle|\; T \text{ is a connected component of } T' \text{ and } \mathtt{b}^k(T) \neq \perp\right\}\right|$$

and

$$\#^k_{\mathrm{pb}}(T') = \left|\left\{T \;\middle|\; T \text{ is a connected component of } T' \text{ and } T \text{ contains a } k\text{-pre-branching vertex}\right\}\right|.$$

Thus, $\#^k_{\mathrm{b}}(T')$ and $\#^k_{\mathrm{pb}}(T')$ count the number of connected components of T' having k-branchings and k-pre-branching vertices, respectively. Also, we define

$$\mathbb{1}^k_{\mathrm{wb}}(T') = \begin{cases} 1, & \text{if } T' \text{ has a weakly } k\text{-branching vertex,} \\ 0, & \text{otherwise} \end{cases}$$

and

$$\#^k_{\mathtt{c}_0}(T') = \left|\left\{T \;\middle|\; T \text{ is a connected component of } T' \text{ and } \mathtt{c}_0(T) = k\right\}\right|.$$

Note that all four above functions give 0 or 1, i.e., they act as 'indicator functions', if T' is a tree. For an integer $p \geq 0$, we define $\mathbb{1}(p) = 0$ if $p = 0$ and $\mathbb{1}(p) = 1$ if $p > 0$.

Lemma 4.1 given below follows in a very direct manner from Theorems 3.6 and 3.7, together with the relevant definitions – we omit its proof. This lemma is crucial for our algorithm because it provides a constructive method for the computation of the label of a rooted tree T, assuming that:

- the labels of the children of the root of T are given, and
- if $\mathtt{r}(T)$ has exactly one child u such that $T[u]$ has k-branching v, then the label of $T - V_{T[u]}$ is given.

In the algorithm stated later, this scheme is used in a bottom-up fashion, which allows us to compute the labels of all vertices of the input tree, starting at the leaves and ending at the root. The label of the root gives us the desired zero-visibility copnumber.

Lemma 4.1. *Let T be a rooted tree on two or more vertices. Let $k = \max\{\mathrm{c}_0(T') \mid T'$ is a connected component of $\overline{T}\}$ and $v = \mathrm{b}^k(\overline{T})$. Then:*

(i) *If $\#^k_{\mathrm{b}}(\overline{T}) > 1$, then $\mathrm{c}_0(T) = k+1$ and $\#^{k+1}_{\mathrm{b}}(T) = \mathbb{1}^{k+1}_{\mathrm{wb}}(T) = \#^{k+1}_{\mathrm{pb}}(T) = 0$.*

(ii) *If $\#^k_{\mathrm{b}}(\overline{T}) = 1$ and $\mathrm{c}_0(T - V_{T[v]}) \geq k$, then $\mathrm{c}_0(T) = k+1$ and $\#^{k+1}_{\mathrm{pb}}(T) = \mathbb{1}^{k+1}_{\mathrm{wb}}(T) = \#^{k+1}_{\mathrm{b}}(T) = 0$.*

(iii) *If $\#^k_{\mathrm{b}}(\overline{T}) = 1$ and $\mathrm{c}_0(T - V_{T[v]}) < k$, then $\mathrm{c}_0(T) = k$, $\mathrm{b}^k(T) = v$ and $\#^k_{\mathrm{b}}(T) = \mathbb{1}^k_{\mathrm{wb}}(T) = \#^k_{\mathrm{pb}}(T) = 1$.*

(iv) *If $\#^k_{\mathrm{b}}(\overline{T}) = 0$, then:*

(1) *If $\mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = 1$ and $\#^k_{\mathrm{c}_0}(\overline{T}) > 1$, then $\mathrm{c}_0(T) = k+1$ and $\#^{k+1}_{\mathrm{b}}(T) = \mathbb{1}^{k+1}_{\mathrm{wb}}(T) = \#^{k+1}_{\mathrm{pb}}(T) = 0$.*

(2) *If $\mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = 1$ and $\#^k_{\mathrm{c}_0}(\overline{T}) \leq 1$, then $\mathrm{c}_0(T) = k$, $\mathrm{b}^k(T) = \mathrm{r}(T)$ and $\#^k_{\mathrm{b}}(T) = \mathbb{1}^k_{\mathrm{wb}}(T) = \#^k_{\mathrm{pb}}(T) = 1$.*

(3) *If $\mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = 0$, then:*

(3a) *If $\#^k_{\mathrm{pb}}(\overline{T}) > 2$, then $\mathrm{c}_0(T) = k+1$ and $\#^{k+1}_{\mathrm{b}}(T) = \mathbb{1}^{k+1}_{\mathrm{wb}}(T) = \#^{k+1}_{\mathrm{pb}}(T) = 0$.*

(3b) *If $\#^k_{\mathrm{pb}}(\overline{T}) = 2$, then $\mathrm{c}_0(T) = k$, $\#^k_{\mathrm{b}}(T) = 0$ and $\mathbb{1}^k_{\mathrm{wb}}(T) = \#^k_{\mathrm{pb}}(T) = 1$. ($\mathrm{r}(T)$ is a weakly k-branching vertex.)*

(3c) *If $\#^k_{\mathrm{pb}}(\overline{T}) < 2$, then $\mathrm{c}_0(T) = k$, $\#^k_{\mathrm{b}}(T) = \mathbb{1}^k_{\mathrm{wb}}(T) = 0$ and $\#^k_{\mathrm{pb}}(T) = 1$ ($\mathrm{r}(T)$ is k-pre-branching).*

We will use Lemma 4.1 to directly obtain a dynamic programming algorithm that computes the zero-visibility copnumber of a given tree T. Roughly speaking, the algorithm roots T at any vertex (we refer by T to the rooted tree in the following), and then T is 'processed' in a bottom-up fashion. In particular, for each vertex v, a label of $T[v]$ (and possibly a label of some subtree of $T[v]$) is computed. (See below for the definition of a label.) The label of $T[v]$ gives the zero-visibility cop number of $T[v]$, and it can be computed based on the labels of the children of v. For this reason the label contains also some other entries besides $\mathrm{c}_0(T[v])$. Once the labels of all vertices are computed, the label of $\mathrm{r}(T)$ gives us the desired $\mathrm{c}_0(T)$.

Formally, we define a *label* of any rooted tree T, as

$$L(T) = \left(k, \#^k_{\mathrm{b}}(T), \mathbb{1}^k_{\mathrm{wb}}(T), \#^k_{\mathrm{pb}}(T), v\right)$$

where $k = \mathrm{c}_0(T)$, and v is a k-branching vertex if T has one, or v is undefined otherwise. We use for brevity the symbol $\perp$ in the last entry when v is undefined. Then, for any $k \geq \mathrm{c}_0(T)$, a *k-label* of T is $L_k(T) = L(T)$ if $k = \mathrm{c}_0(T)$, and $L_k(T) = (k, 0, 0, 0, \perp)$ otherwise.

We start with a claim that allows us to determine a k-label of a tree T in case when $\#^k_{\mathrm{b}}(\overline{T}) \neq 1$.

Claim 4.2. *Let T be a rooted tree. Denote the trees in $\overline{T}$ by $T_1, \ldots, T_l$. Let $k = \max\{\mathrm{c}_0(T_j) \mid j \in \{1, \ldots, l\}\}$. If the k-labels of $T_1, \ldots, T_l$ are given and $\#^k_{\mathrm{b}}(\overline{T}) \neq 1$, then the k'-label of T can be computed in time $O(l)$, where $k' \geq \mathrm{c}_0(T)$.*

Proof. We use Lemma 4.1 to state the formulæ for the particular entries of the label of T,

$$L(T) = \left(m, \#^m_{\mathrm{b}}(T), \mathbb{1}^m_{\mathrm{wb}}(T), \#^m_{\mathrm{pb}}(T), v\right).$$

Since $\#^k_{\mathrm{b}}(\overline{T}) \neq 1$, we have:

$$m = \mathrm{c}_0(T) = \begin{cases} k+1, & \text{if } \#^k_{\mathrm{b}}(\overline{T}) > 1 \vee \left(\mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = 1 \wedge \#^k_{\mathrm{c}_0}(\overline{T}) > 1\right) \vee \left(\mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = 0 \wedge \#^k_{\mathrm{pb}}(\overline{T}) > 2\right), \\ k, & \text{otherwise.} \end{cases}$$

Note that the following formulas allow us to find $\#^k_{\mathrm{b}}(\overline{T})$, $\#^k_{\mathrm{pb}}(\overline{T})$, $\mathbb{1}^k_{\mathrm{wb}}(\overline{T})$ and $\#^k_{\mathrm{c}_0}(\overline{T})$:

$$\#^k_{\mathrm{b}}(\overline{T}) = \sum_{j=1}^{l} \#^k_{\mathrm{b}}(T_j), \quad \#^k_{\mathrm{pb}}(\overline{T}) = \sum_{j=1}^{l} \#^k_{\mathrm{pb}}(T_j),$$

$$\#^k_{\mathrm{c}_0}(\overline{T}) = \left|\left\{j \mid \mathrm{c}_0(T_j) = k\right\}\right|, \quad \mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = \mathbb{1}\left(\sum_{j=1}^{l} \mathbb{1}^k_{\mathrm{wb}}(T_j)\right).$$

The values of $\#^k_{\mathrm{b}}(T_j)$, $\#^k_{\mathrm{pb}}(T_j)$, $\mathrm{c}_0(T_j)$ and $\mathbb{1}^k_{\mathrm{wb}}(T_j)$ are taken directly from the k-label of T_j for each $j \in \{1, \ldots, l\}$.

If $m = k+1$, then the label of T is $(m, 0, 0, 0, \perp)$. Thus, the k'-label of T is $L_{k'}(T) = (k', 0, 0, 0, \perp)$ and the proof is completed. Hence, let $m = k$ in the following.

Algorithm 2 Computing the zero-visibility copnumber of an input tree T.

1: Root T at any vertex.
2: Take a permutation $u_1, \ldots, u_n$ of vertices of T such that $u_i \prec u_j$ implies $i < j$ for all $i \neq j$.
3: **for** $i := 1$ **to** n **do**
4: Let $v_1, \ldots, v_\ell$ be the children of u_i in T.
5: $k := \max\{\mathrm{c}_0(T[v_j]) \mid j \in \{1, \ldots, \ell\}\}$
6: Compute $\#^k_{\mathrm{b}}(\overline{T[u_i]})$.
7: **if** $\#^k_{\mathrm{b}}(\overline{T[u_i]}) = 1$ **then**
8: Find the index $s \in \{1, \ldots, \ell\}$ such that $\#^k_{\mathrm{b}}(T[v_s]) = 1$.
9: Compute the k-label of $T[u_i] - V_{T[x_i]}$, where $x_i = \mathrm{b}^k(T[v_s])$.
10: Compute the label of $T[u_i]$.
11: **else**
12: Compute the label of $T[u_i]$.
13: **end if**
14: **end for**
15: **return** $\mathrm{c}_0(T)$ that is stored in its label.

We have $\#^k_{\mathrm{b}}(T) = \mathbb{1}(q + \#^k_{\mathrm{b}}(\overline{T}))$, where $q = 1$ if $\mathrm{r}(T)$ is k-branching and $q = 0$ otherwise. Lemma 4.1, $m = k$ and $\#^k_{\mathrm{b}}(\overline{T}) \neq 1$ imply that $q = 1$ if and only if: $\#^k_{\mathrm{b}}(\overline{T}) = 0$, $\mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = 1$ and $\#^k_{\mathrm{c}_0}(\overline{T}) \leq 1$.

Then, $\mathbb{1}^k_{\mathrm{wb}}(T) = \mathbb{1}(q' + \mathbb{1}^k_{\mathrm{wb}}(\overline{T}))$, where $q' = 1$ if $\mathrm{r}(T)$ is weakly k-branching and $q = 0$ otherwise. Lemma 4.1, $m = k$ and $\#^k_{\mathrm{b}}(\overline{T}) \neq 1$ imply that $q' = 1$ if and only if: $\#^k_{\mathrm{b}}(\overline{T}) = 0$, $\mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = 0$ and $\#^k_{\mathrm{pb}}(\overline{T}) = 2$.

Similarly, $\#^m_{\mathrm{pb}}(T) = \mathbb{1}(q'' + \#^k_{\mathrm{pb}}(\overline{T}))$, where $q'' = 1$ if $\mathrm{r}(T)$ is k-pre-branching and $q = 0$ otherwise. This follows from the fact that if the root of T is k-branching or weakly k-branching, then one of the subtrees T_j's contains a k-pre-branching vertex. Again, Lemma 4.1, $m = k$ and $\#^k_{\mathrm{b}}(\overline{T}) \neq 1$ imply that $q'' = 1$ if and only if: $\#^k_{\mathrm{b}}(\overline{T}) = 0$, $\mathbb{1}^k_{\mathrm{wb}}(\overline{T}) = 0$ and $\#^k_{\mathrm{pb}}(\overline{T}) < 2$.

It remains to determine the vertex v in $L(T)$ in case when $m = k$. Thus, $\#^k_{\mathrm{b}}(\overline{T}) = 0$ and hence $v = \mathrm{r}(T)$ if $q = 1$ and $v = \perp$ otherwise. Finally, we obtain that $L_{k'}(T) = L(T)$ when $k' = k$ and $L_{k'}(T) = (k', 0, 0, 0, \perp)$ otherwise. □

Theorem 4.3. *There exists a linear time algorithm that computes the zero-visibility cop number of any tree.*

Proof. Our proof is constructive, i.e., we describe an algorithm computing the zero-visibility copnumber of an input tree T. We start with its informal description that points out the main ideas. First, T is rooted at any vertex. The vertices of T are ordered as $v_1, \ldots, v_n$ so that each vertex appears in it prior to its parent. The vertices are then processed according to this order. The processing of a vertex u leads to finding the label of the subtree $T[v]$ and, if this subtree has a $\mathrm{c}_0(T[u])$-branching v, then the $\mathrm{c}_0(T)$-label of the subtree $T[u] - V_{T[v]}$. Denote by k the maximum zero-visibility copnumber among the subtrees descendant from the children of u, and denote by $v_1, \ldots, v_l$ the children of u. Note that the number of k-branching vertices in the subtrees $T[v_1], \ldots, T[v_l]$, i.e., $\#^k_{\mathrm{b}}(\overline{T[u]})$, can be computed on the basis of the labels of those subtrees. Then, two cases are considered. In the first case $\#^k_{\mathrm{b}}(\overline{T[u]}) = 1$ where in order to compute the label of $T[u]$ we need to know the zero-visibility copnumber of the subtree $T[u] - V_{T[v]}$, where $v = \mathrm{b}^k(T[u])$. (See Lemma 4.1(ii) and 4.1(iii).) In order to compute the desired $\mathrm{c}_0(T[u] - V_{T[v]})$, we compute the k-label of this subtree. Note that, for the computation of the latter label, we do not use 4.1(ii) and 4.1(iii) since, by assumption, there is no k-branching vertex in $T[u] - V_{T[v]}$. In the second case $\#^k_{\mathrm{b}}(\overline{T[u]}) \neq 1$ where we use Lemma 4.1(i) and 4.1(iv) for the label computation.

We now give a sketch of the algorithm in the form of a pseudo-code. This sketch gives the order of computations of labels of selected subtrees. The details on how the instruction from lines 6, 9, 10 and 12, can be implemented are given below.

We now give the details of Algorithm 2 by arguing that its subsequent calculations can be performed. To that end we argue by induction on $i \in \{1, \ldots, n\}$ that at the end of the i-th iteration of the 'for' loop the following invariant is satisfied:

(i1) for each $j \in \{1, \ldots, i\}$, the k_j-label $L_{k_j}(T[u_j])$ is computed, where $k_j = \mathrm{c}_0(T[u_j])$, and

(i2) for each $j \in \{1, \ldots, i\}$, if $\#^{k_j}_{\mathrm{b}}(T[u_j]) = 1$, then the k_j-label $L_{k_j}(T[u_j] - V_{T[x_j]})$ is computed, where $x_j = \mathrm{b}^{k_j}(T[u_j])$.

Note that when u_i has no children, then in the i-th iteration we have: $k = 0$, $\#^k_{\mathrm{b}}(\overline{T[u_i]}) = 0$. Hence, in such case the label $L(T[u_i]) = (1, 0, 0, 0, \perp)$ can be computed in line 12. Thus, in particular, the claim holds for $i = 1$.

Suppose that (i1) and (i2) hold for some $i - 1 \geq 1$ and we consider the i-th iteration of the 'for' loop of Algorithm 2.

First note that the integer k, computed in line 5, can be obtained directly from the labels of the subtrees $T[v_i]$, $i \in \{1, \ldots, l\}$; each of those labels holds the zero-visibility number of the corresponding subtree. By the inductive hypothesis (i1), the latter labels are computed. Moreover, in line 6, we can use the following formula:

$$\#^k_{\mathrm{b}}(\overline{T[u_i]}) = \mathbb{1}\Big(\sum_{j=1}^{l} \#^k_{\mathrm{b}}(T[v_i])\Big).$$

Denote $T' = T[u_i] - V_{T[x_i]}$ and we now consider the computation of the k-label $L_k(T')$ in line 9. Denote $T_j = T[v_j]$ for $j \in \{1, \ldots, l\} \setminus \{s\}$ and $T_s = T[v_s] - V_{T[x_s]}$. By (i1), the label of T_j is computed at the end of the $(i-1)$-th iteration of Algorithm 2 for each $j \in \{1, \ldots, l\} \setminus \{s\}$. By (i2), the label of T_s is also computed. Note that $\#_{\mathrm{b}}^k(\overline{T[u_i]}) = 1$ implies $\#_{\mathrm{b}}^k(T') = 0$. Thus, by Claim 4.2, the k-label of T' can be indeed computed in line 9.

We now consider the computation of $L(T[u_i])$ in line 10. Note that $\#_{\mathrm{b}}^k(\overline{T[u_i]}) = 1$. Thus, by Lemma 4.1, if $\mathtt{c}_0(T') \geq k$, then $L(T[u_i]) = (k+1, 0, 0, 0, \perp)$; and if $\mathtt{c}_0(T') < k$, then $L(T[u_i]) = (k, 1, 1, 1, x_s)$. Note that $\mathtt{c}_0(T')$ can be derived from its label computed in line 9.

Finally, Claim 4.2 implies that the label of $T[u_i]$ can be computed in line 12. This completes the proof of (i1) and (i2).

We finish by estimating the running time of Algorithm 2. By Claim 4.2, computations in lines 9 and 12 can be done in time $O(l)$. Lines 5, 6 and 8 clearly require linear time in l. By the arguments above, the computation of label in line 10 can be done in time $O(1)$. Thus, the running time of the i-th iteration of the 'for' loop is linear in the number of children of u_i. This implies that the running time of Algorithm 2 is $O(n)$. □

5. NP-hardness of zero-visibility cops and robber

Let G be an n-vertex graph with vertex set $V_G = \{v_1, \ldots, v_n\}$. Let C be a set and let $\mathcal{V}_G = \{V_{ij} \mid v_i v_j \in E_G\}$ be a collection of sets where $\mathcal{V}_G$ together with C form a pairwise disjoint collection of sets of order n. Define $\xi(G)$ to be the graph obtained as follows. The vertex set of $\xi(G)$ is

$$V_{\xi(G)} = C \cup \bigcup_{v_i v_j \in E_G} V_{ij},$$

where C, and V_{ij} is of size n for each ij such that $v_i v_j \in E_G$. Moreover, C and all $V_1, \ldots, V_m$ are pairwise disjoint. Then, the edge set of $\xi(G)$ is defined so that C induces a clique and $V_{ij} \cup \{c_i, c_j\}$ induces a clique of order $n+2$ for each edge $v_i v_j$ of G.

For any $n > 0$ define $\mathcal{G}_n = \{\xi(G) \mid G \text{ is an } n\text{-vertex graph}\}$ and let $\mathcal{G} = \bigcup_{n>0} \mathcal{G}_n$. The above construction appears in [7], where it has been proved that the problem of deciding whether $\mathtt{pw}(G) \leq k$ is NP-complete for $G \in \mathcal{G}_n$ and $n < k < 2n-3$. Here, $\mathtt{pw}(G)$ denotes the pathwidth of G. One can prove that the result holds when we assume that $G \in \mathcal{G}_n$ when both n and k are even. To summarise, we have the following.

Theorem 5.1. *(See [7].) Given $G \in \mathcal{G}_n$, where n is even, and an even integer k^*, $2 \leq k^* \leq n-4$, the problem of deciding whether $\mathtt{pw}(G) \leq n + k^* - 1$ is NP-complete.*

In the remaining part of this section we assume that the integers n and k^* are even and $k^* \leq n-4$.

We recall the following definition from [7].

Definition 5.2. (See [7].) A path decomposition $(X_1, \ldots, X_l)$ of a graph G is *normalised* if for each maximal clique Y of G there exists exactly one $j \in \{1, \ldots, l\}$ such that $Y \subseteq X_j$.

Theorem 5.3. *(See [7].) For each graph G there exists an optimal normalised path decomposition.*

Now we recall the node search problem that we will use in our reduction. Given any graph G that contains a fugitive, a *node k-search* for G is a sequence of moves such that each move is composed of the following two actions:

1. Place a number of searchers on the vertices of G, which results in at most k vertices being occupied by the searchers.
2. Remove from G a subset of searchers that are present on the vertices of G.

A vertex is *contaminated* if it may contain the fugitive. Initially, all vertices are contaminated. The fugitive is invisible and fast, i.e., it can traverse at any moment along an arbitrary path that is free of searchers. A node search is *successful* if there are no contaminated vertices after the last move of a node search $\mathcal{N}$. The minimum number of searchers k such that there exists a successful node k-search strategy for G is called the *node search number* of G and is denoted by $\mathtt{ns}(G)$.

We will prove that for each $G \in \mathcal{G}_n$, $\mathtt{c}_0(G) \leq n/2 + k^*/2$ if and only if $\mathtt{ns}(G) \leq n + k^*$. First we focus on proving that $\mathtt{ns}(G) \leq n + k^*$ implies $\mathtt{c}_0(G) \leq n/2 + k^*/2$ and then we prove the reverse implication.

We have the following [9,11,12,14]:

Theorem 5.4. *For any graph G, $\mathtt{pw}(G) = \mathtt{ns}(G) - 1$.*

By Theorem 5.1 and by Theorem 5.4,

Theorem 5.5. *Given $G \in \mathcal{G}_n$ and k^*, the problem of deciding whether $\mathtt{ns}(G) \leq n + k^*$ is NP-complete.*

Let $\mathcal{P} = (X_1, \ldots, X_l)$ be a path decomposition of a graph G. We say that a node search $\mathcal{N}$ is *derived* from $\mathcal{P}$ if the j-th move of $\mathcal{N}$ places the searchers on the unoccupied vertices in X_j and then removes the searchers from the vertices in $X_j \setminus X_{j+1}$. Note that the node search strategy derived from a path decomposition is monotonic.

Lemma 5.6. *Let $G \in \mathcal{G}_n$ and let k^* be given. If $\mathtt{ns}(G) \leq n + k^*$, then $\mathtt{c}_0(G) \leq n/2 + k^*/2$.*

Proof. Suppose that $\mathtt{ns}(G) \leq n + k^*$. By Theorem 5.3 and by Theorem 5.4, there exists a normalised path decomposition $\mathcal{P} = (X_1, \ldots, X_l)$ of G of width $n + k^* - 1$. From Definition 5.2 we know that for each $i \in \{1, \ldots, l\}$ and for each $V' \in \mathcal{V}_G$,

$$V' \subseteq X_i \text{ or } V' \cap X_i = \emptyset. \tag{1}$$

Similarly as in [4], we obtain a cops' successful strategy $\mathcal{L}$ that uses at most $\lceil \max_i |X_i|/2 \rceil$ cops. Since n and k^* are even and $\max_i |X_i| \leq n + k^*$, $\mathcal{L}$ uses at most $(n + k^*)/2$ cops, which completes the proof. □

In the following we prove the reverse implication, that is, $\mathtt{c}_0(G) \leq n/2 + k^*/2$ implies $\mathtt{ns}(G) \leq n + k^*$. The key idea that we use is an observation that, due to the small diameter and the clique structure of the graphs in $\mathcal{G}_n$, the recontamination spreads, informally speaking, 'as quickly' in the cops and robber game as in the node search. We start with the following lemma.

Lemma 5.7. *Let $G \in \mathcal{G}_n$. If $\mathtt{c}_0(G) \leq k < n - 3$, then there exists a successful strategy $\mathcal{L} = \{l_i\}_{i=1}^k$ of length t in G such that:*

(i) *for each $j \in \{1, \ldots, t\}$ there exists $V' \in \mathcal{V}_G$ such that $\mathcal{L}_j \subseteq C \cup V'$, and*
(ii) *if $\mathcal{S}_{j-1} \cap V' \neq \emptyset$ and $\mathcal{S}_j \cap V' = \emptyset$ for some $j \in \{1, \ldots, t\}$ and $V' \in \mathcal{V}_G$, then $|\mathcal{L}_{j-1} \cap V'| \geq n/2$ and $|\mathcal{L}_j \cap V'| \geq n/2$.*

Proof. Since $\mathtt{c}_0(G) \leq k$, there exists a successful strategy $\mathcal{L}' = \{l'_i\}_{i=1}^k$ of length t on G. We may without loss of generality assume that, in $\mathcal{L}'$, no cop enters a $V' \in \mathcal{V}_G$ if all vertices of V' are clean, i.e., for each $i \in \{1, \ldots, k\}$, if all vertices of V' are clean in some turn $j - 1$, then $l'_i(j) \notin V'$. This follows from the structure of the graphs in $\mathcal{G}_n$.

We start by modifying $\mathcal{L}'$, if necessary. We describe a modification that is applied iteratively as long as a time point j defined below exists. Informally speaking, we find a time point j such that some cops are on a $V' \in \mathcal{V}_G$ during at least three consecutive turns starting with turn j. Then, we can 'postpone' the cops so that they enter in turn $j + 1$. Formally, suppose that there exists a minimum integer $j \in \{1, \ldots, t\}$ and $V' \in \mathcal{V}_G$ such that $V' \cap \mathcal{L}'_{j-1} = \emptyset$, $V' \cap \mathcal{L}'_j \neq \emptyset$, $V' \cap \mathcal{L}'_{j+1} \neq \emptyset$ and $V' \cap \mathcal{L}'_{j+2} \neq \emptyset$, (If no such j and V' exist, then no modification to $\mathcal{L}'$ is made.) Let $I = \{i \in \{1, \ldots, k\} \mid l'_i(j) \in V'\}$, i.e., I is the set of cops that are present on the vertices of V' in turn j. By assumption, $I \neq \emptyset$. Modify $\mathcal{L}'$ as follows. If $i \in \{1, \ldots, k\} \setminus I$, then l'_i does not change. If $i \in I$, then change $l'_i(j)$ to be an arbitrarily selected vertex $v \in N_G(V')$. Note that, after this modification, l'_i is still a walk because $N_G[V']$ induces a clique in G. If $\mathcal{L}''$ is the modified strategy, then we have $\mathcal{L}''_{j+1} \cap N_G[V'] = \mathcal{L}'_{j+1} \cap N_G[V']$ and $\mathcal{L}''_{j+2} \cap N_G[V'] = \mathcal{L}'_{j+2} \cap N_G[V']$. Thus, at the end of turn $j + 2$, the vertices of V' are clean in $\mathcal{L}''$ if and only if the vertices of V' are clean in $\mathcal{L}'$.

Denote by $\mathcal{L}$ the strategy obtained by iteratively applying to $\mathcal{L}'$ the transformation given above. We write $\mathcal{L}'$ to refer to the initial strategy. Note that the set of vertices in C occupied by the cops in $\mathcal{L}'$ in any turn $j \in \{1, \ldots, t\}$ is contained in the set of vertices in C occupied by the cops in $\mathcal{L}$ in turn j, i.e., $\mathcal{L}'_j \cap C \subseteq \mathcal{L}_j \cap C$. Moreover, we obtain that for each $V' \in \mathcal{V}_G$ and for each $j \in \{1, \ldots, t\}$, $V' \subseteq V_G \setminus \mathcal{S}_j$ if and only if $V' \subseteq V_G \setminus \mathcal{S}'_j$. This implies that $\mathcal{L}$ is successful.

We argue that $\mathcal{L}$ satisfies (i). Suppose for a contradiction that (i) does not hold, i.e., that there exists $j \in \{1, \ldots, t\}$ such that $V' \cap \mathcal{L}_j \neq \emptyset$ and $V'' \cap \mathcal{L}_j \neq \emptyset$ for some $V', V'' \in \mathcal{V}_G$. Select j to be the minimum index with this property. By construction of $\mathcal{L}$, we have two cases: either all vertices in $V' \cup V''$ become clear in turn $j + 1$, or the vertices of V' become clear in turn $j + 1$ and the vertices of V'' become clear in turn j. In both cases there are at least $n/2$ cops on both V' and V'' in turn $j + 1$. Hence, $k \geq n$, a contradiction.

To conclude the second part of the lemma observe that $k < n - 3$ implies that if all vertices of some clique $V' \in \mathcal{V}_G$ are dirty in some turn j, then $V' \nsubseteq \mathcal{L}_{j+1}$, i.e., all vertices of V' cannot be cleaned in turn $j + 1$. □

We say that a finite successful strategy $\mathcal{L} = \{l_i\}_{i=1}^k$ of length t for G is *normal* if there exist $1 < s_1 < s_2 < \cdots < s_p \leq t$ such that

(N1) $s_{j+1} - s_j \geq 3$ for each $j \in \{1, \ldots, p - 1\}$,
(N2) for each $j \in \{1, \ldots, p\}$, there exists a unique $V' \in \mathcal{V}_G$ such that $\mathcal{L}_q \subseteq C \cup V'$ and at least $n/2$ cops are on the vertices in $N_G[V']$ in turn q for each $q \in \{s_j - 2, \ldots, s_j + 1\}$.
(N3) $\mathcal{L}_j \subseteq C$ for each $j \in \bigcup_{j'=1}^{p} \{s_{j'-1} + 1, \ldots, s_{j'} - 2\}$, where $s_0 = 1$.

We call $s_1, \ldots, s_p$ the *clearing sequence of* $\mathcal{L}$. If $\mathcal{L}_q \cap V' \neq \emptyset$ for some $s_{j-1} < q \leq s_j$, then we say that V' is the clique *corresponding to* s_j.

Before we prove that normal strategies exist, let us give an intuition on the structure of such a strategy. Condition (N1) requires that two turns in which two cliques in $\mathcal{V}_G$ become clear are separated by at least two other turns. If $k < n$, then (informally speaking) $n/2$ cops need to be on the vertices of V' in two consecutive turns $s_j - 1$ and s_j to clear V' and therefore those cops are on $N_G[V']$ in the four moves to which we refer in (N2). Finally, (N3) says that the cops are on the vertices on C between the moves that clear the cliques corresponding to $s_1, \ldots, s_p$. Note that $s_{j+1} - s_j = 3$ is possible only if the two cliques that correspond to s_j and s_{j+1} have a common neighbour in C.

Lemma 5.8. *Let $G \in \mathcal{G}_n$. If $c_0(G) \leq k < n$, then there exists a normal strategy that uses k cops for G.*

Proof. Let $\mathcal{L} = \{l_i\}_{i=1}^k$ be a strategy that satisfies the conditions in Lemma 5.7. Select $s_1, \ldots, s_p$ as an increasing sequence of integers in $\{1, \ldots, t\}$ such that $V' \cap \mathcal{S}_{s_j - 1} \neq \emptyset$ and $V' \cap \mathcal{S}_{s_j} = \emptyset$ for some $V' \in \mathcal{V}_G$ if and only if $j \in \{1, \ldots, p\}$. Note that $s_1 > 1$ because $k < n$. By Lemma 5.7(ii), for each $j \in \{1, \ldots, p\}$ at least $n/2$ cops are on the vertices of some $V' \in \mathcal{V}_G$ in turns s_j and $s_j - 1$. Hence, by construction of G and Lemma 5.7(i), $\mathcal{L}_{s_j - 2} \cup \mathcal{L}_{s_j + 1} \subseteq C$. This proves (N1) and (N2) for $\mathcal{L}$. Condition (N3) follows directly from the fact that one may always assume that if all vertices of a $V' \in \mathcal{V}_G$ are clear at the beginning of some step j, then no cop enters V' in step j. □

Let $G \in \mathcal{G}_n$. Let $\mathcal{L} = \{l_i\}_{i=1}^k$ be a normal strategy of length t for G and let $s_1, \ldots, s_p$ be its clearing sequence. Define a node search $\mathcal{N}$ for G that consists of p *stages*, where the j-th stage is the following sequence of moves:

(S1) Place the searchers on all unoccupied vertices of C. Remove the searchers from all vertices in $C \setminus (\mathcal{L}_{s_j} \cup \mathcal{L}_{s_j - 1})$. (Note that this may cause recontamination for some vertices in C.)
(S2) Place the searchers on all vertices of the clique V' corresponding to s_j. Remove all searchers from the vertices in V'.

We say that $\mathcal{N}$ is the node search *derived from* $\mathcal{L}$. Below we prove that the definition of $\mathcal{N}$ is correct, i.e., $\mathcal{N}$ is indeed a node $2k$-search for G.

Lemma 5.9. *Let $G \in \mathcal{G}_n$. The node search $\mathcal{N}$ derived from a normal strategy $\mathcal{L}$ that uses k cops is successful and uses at most $2k$ searchers.*

Proof. First observe that $k \geq n/2$, because $\mathcal{L}$ is successful and G contains a clique on n vertices as a subgraph. Hence, $n \leq 2k$. Note that at the end of each stage of $\mathcal{N}$ the set of vertices that are occupied is a subset of C. In (S1) at most $|C|$ searchers are on the vertices of G, and $|C| = n \leq 2k$. Step (S2) uses $n + j$ searchers, where j is the number of searchers on the vertices of C. By construction, $j \leq 2k - n$, which proves that $\mathcal{N}$ uses at most $2k$ searchers.

Now we prove that $\mathcal{N}$ is successful. Denote by U_j the set of contaminated vertices in $V_G \setminus C$ at the end of j-th stage of $\mathcal{N}$ for each $j \in \{0, \ldots, p\}$. We argue, by induction on j, that

$$U_j \subseteq \mathcal{R}_{s_j + 1}. \tag{2}$$

In other words, we argue that after each move (S2) of $\mathcal{N}$ the set of vertices that are contaminated is contained in the set of vertices contaminated in $\mathcal{L}$ immediately after the robber's s_j-th turn (i.e., immediately before the cops' $(s_j + 1)$-st turn).

For $j = 0$ we take $s_0 = -1$ and $U_0 = \mathcal{R}_0 = V_G$, which reflects the initial states in both strategies.

Let $j \in \{0, \ldots, p - 1\}$. Suppose that (2) holds and we prove that it is satisfied for $j + 1$. Since $\mathcal{L}$ is normal, we obtain that, for each $V' \in \mathcal{V}_G$, $V' \cap \mathcal{R}_{s_j + 1} \neq \emptyset$ if and only if $V' \subseteq \mathcal{R}_{s_j + 1}$. Let $V_1, \ldots, V_q$ be all cliques in $\mathcal{V}_G$ such that each of them is contained in $\mathcal{R}_{s_j + 1}$, and let

$$X = \bigcup_{i=1}^{q} N_G(V_i).$$

If $X \subseteq \mathcal{L}_{s_{j+1}} \cup \mathcal{L}_{s_{j+1} - 1}$, then by (S1) and (S2) the vertices in $N_G(V_{q'})$ are guarded in all moves of the j-stage of $\mathcal{N}$ for each $q' \in \{1, \ldots, q\}$. Hence, by (S1), no recontamination occurs in any of the moves in the j-th stage of $\mathcal{N}$, and the claim follows for $j + 1$ in such case.

Thus, assume that $X \not\subseteq \mathcal{L}_{s_{j+1}} \cup \mathcal{L}_{s_{j+1} - 1}$. Then, there exists $v' \in C$ such that

$$v' \in \mathcal{R}_{s_{j+1} - 1} \text{ and } v' \notin \mathcal{L}_{s_{j+1}} \cup \mathcal{L}_{s_{j+1} - 1}.$$

Since C induces a clique in G, we have

$$u \in \mathcal{R}_{s_{j+1}} \text{ for each } u \in C \setminus (\mathcal{L}_{s_{j+1}} \cup \mathcal{L}_{s_{j+1} - 1}). \tag{3}$$

Finally, each clique V' corresponding to any such vertex u becomes contaminated after robber's s_{j+1}-th turn, i.e., $V' \subseteq \mathcal{R}_{s_{j+1} + 1}$. Thus, (3) and the definition of $\mathcal{N}$ in (S1) and (S2) complete the proof of (2).

Note that no clique in $\mathcal{V}_G$ is contaminated at the end of turn s_p because $\mathcal{L}$ is successful. Also, if there exists $v \in C \cap \mathcal{S}_{s_p}$, then some clique corresponding to v would be contaminated at the end of s_p-turn — a contradiction. Hence, $\mathcal{R}_{s_p+1} = \emptyset$ and it follows from (2) that $\mathcal{N}$ clears all vertices of G as required. □

Lemma 5.10. *Let $G \in \mathcal{G}_n$ and let k^* be given. If $\mathrm{c}_0(G) \leq n/2 + k^*/2$, then $\mathtt{ns}(G) \leq n + k^*$.*

Proof. Let $\mathcal{L} = \{l_i\}_{i=1}^{k}$ be a normal successful strategy for G. By Lemma 5.8 such a strategy exists. By Lemma 5.9, there exists a node $2k$-search for G. □

Lemmas 5.6 and 5.10 give the following.

Theorem 5.11. *Given $G \in \mathcal{G}_n$ and an integer $k > 0$, the problem of deciding whether $\mathrm{c}_0(G) \leq k$ is NP-complete.* □

Since $\mathcal{G}_n$ consists of starlike graphs for each $n > 0$, we have the following corollary.

Corollary 5.12. *The problem of computing the zero-visibility copnumber of a given starlike graph is NP-hard.* □

6. Conclusions and future directions

Having shown that the zero-visibility cops and robber game has a linear time algorithm for trees (and is NP-complete for starlike graphs), it is natural to consider other special classes of graphs for which similar complexity results are known for edge and node searching. Particularly, interval graphs (which can be essentially searched from left to right) and cycle-disjoint graphs (which are 'tree-like') are good candidates [6,20].

Outside of complexity, it also remains open to characterise those graphs with $\mathrm{c}_0(G) \leq 2$ [19,8]. This is a natural question, as characterisations exist for graphs whose edge search number is at most 3 [13]. As well, graphs with copnumber one have long been characterised [15], and there is a recent characterisation for larger copnumbers [3].

References

[1] D. Bienstock, P. Seymour, Monotonicity in graph searching, J. Algorithms 12 (1991) 239–245.
[2] A. Bonato, R.J. Nowakowski, The Game of Cops and Robbers on Graphs, Stud. Math. Libr., vol. 61, American Mathematical Society, Providence, Rhode Island, 2011.
[3] N.E. Clarke, G. MacGillivray, Characterizations of k-copwin graphs, Discrete Math. 312 (2012) 1421–1425.
[4] D. Dereniowski, D. Dyer, R.M. Tifenbach, B. Yang, Zero-visibility cops & robber game on a graph, in: Frontiers in Algorithmics and Algorithmic Aspects in Information and Management, in: Lecture Notes in Computer Science, vol. 7924, 2013, pp. 175–186.
[5] J.A. Ellis, I.H. Sudborough, J.S. Turner, The vertex separation and search number of a graph, Int. J. Inf. Comput. Sci. 113 (1994) 50–79.
[6] F.V. Fomin, P. Heggernes, R. Mihai, Mixed search number and linear-width of interval and split graphs, in: Graph-Theoretic Concepts in Computer Science, in: Lecture Notes in Computer Science, vol. 4769, 2007, pp. 304–315.
[7] J. Gustedt, On the pathwidth of chordal graphs, Discrete Appl. Math. 45 (1993) 233–248.
[8] D. Jeliazkova, Aspects of the cops and robber game played with incomplete information, Master's thesis, Acadia University, 2006.
[9] N.G. Kinnersley, The vertex separation number of a graph equals its path-width, Inform. Process. Lett. 42 (1992) 345–350.
[10] N.G. Kinnersley, Cops and robbers is EXPTIME-complete, arXiv:1309.5405 [math.CO], 2013.
[11] L.M. Kirousis, C.H. Papadimitriou, Interval graphs and searching, Discrete Appl. Math. 55 (1985).
[12] L.M. Kirousis, C.H. Papadimitriou, Searching and pebbling, Theoret. Comput. Sci. 47 (1986) 205–218.
[13] N. Megiddo, S.L. Hakimi, M. Garey, D. Johnson, C.H. Papadimitriou, The complexity of searching a graph, J. ACM 35 (1988) 18–44.
[14] R. Möhring, Graph problems related to gate matrix layout and PLA folding, in: Computational Graph Theory, in: Computing, Suppl., vol. 7, Springer-Verlag, Wien, 1990, pp. 17–51.
[15] R. Nowakowski, P. Winkler, Vertex-to-vertex pursuit in a graph, Discrete Math. 43 (1983) 235–239.
[16] T. Parsons, Pursuit-evasion in a graph, in: Proceedings of the International Conference on the Theory and Applications of Graphs, in: Lecture Notes in Mathematics, vol. 642, Springer-Verlag, 1978, pp. 426–441.
[17] A. Quilliot, Problèmes de jeux, de point fixe, de connectivité et de représentation sur des graphes, des ensembles ordonnés et des hypergraphes, PhD thesis, Université de Paris VI, 1983.
[18] A. Tang, Cops and robber with bounded visibility, Master's thesis, Dalhousie University, 2004.
[19] R. Tošić, Inductive classes of graphs, in: Proceedings of the Sixth Yugoslav Seminar on Graph Theory, University of Novi Sad, 1985, pp. 233–237.
[20] B. Yang, R. Zhang, Y. Cao, Searching cycle-disjoint graphs, in: Combinatorial Optimization and Applications, in: Lecture Notes in Computer Science, vol. 4616, Springer-Verlag, 2007, pp. 32–43.