Theoretical Computer Science 412 (2011) 5700–5713

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Connected searching of weighted trees☆

Dariusz Dereniowski

Department of Algorithms and System Modeling, Gdańsk University of Technology, Gdańsk, Poland

ARTICLE INFO

Article history:
Received 16 September 2010
Received in revised form 16 March 2011
Accepted 14 June 2011
Communicated by D. Peleg

Keywords:
Connected searching
Graph searching
Search strategy

ABSTRACT

In this paper we consider the problem of connected edge searching of weighted trees. Barrière et al. claim in [L. Barrière, P. Flocchini, P. Fraigniaud, N. Santoro, Capture of an intruder by mobile agents, in: SPAA'02: Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, NY, USA, 2002, pp. 200–209] that there exists a polynomial-time algorithm for finding an optimal search strategy, that is, a strategy that minimizes the number of used searchers. However, due to some flaws in their algorithm, the problem turns out to be open. It is proven in this paper that the considered problem is strongly NP-complete even for node-weighted trees (the weight of each edge is 1) with one vertex of degree greater than 2. It is also shown that there exists a polynomial-time algorithm for finding an optimal connected search strategy for a given bounded degree tree with arbitrary weights on the edges and on the vertices. This is an FPT algorithm with respect to the maximum degree of a tree.

1. Introduction

1.1. The background and related work

Given a simple undirected graph G, a fugitive is located on an edge of G. The task is to design a sequence of moves of a team of searchers that results in capturing the fugitive. The fugitive is invisible for the searchers — they can deduce the location of the fugitive only from the history of their moves; the fugitive is fast, i.e. whenever he moves, he can traverse a path of arbitrary length in the graph, as long as the path is free of searchers. Finally, the fugitive has a complete knowledge about the graph and about the strategy of the searchers, which means that he will avoid the capture as long as it is possible. The allowable moves for the searchers are, in general, placing a searcher on a vertex, removing a searcher from a vertex and sliding a searcher along an edge of G. An edge is *clear* if it cannot contain the fugitive, otherwise it is *contaminated*. Capturing the fugitive is then equivalent to clearing all the edges of G. The minimum number of searchers sufficient to clear the graph is the *search number* of G, denoted by $\mathtt{s}(G)$. The edge searching problem has been introduced by Parsons in [31]. The corresponding node searching problem was first studied by Kirousis and Papadimitriou in [24]. For surveys on graph searching problems, see [1,16].

A key property of a search strategy is the monotonicity. A search is *monotone* if the strategy ensures that the fugitive cannot reach an edge that has been already cleared. For most graph searching models it has been proven that there exists an optimal search strategy that is monotone. The minimum number of searchers needed to construct a monotone search strategy for G is denoted by $\mathtt{ms}(G)$.

We say that a search is *internal* if removing the searchers from the graph is not allowed, while for the search to be *connected* we require that after each move of the searchers, the subgraph of G that is clear is connected. The smallest number

☆ A preliminary version of this work was presented at MFCS'10 [9].
E-mail address: deren@eti.pg.gda.pl.

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.06.017

k for which a connected k-search strategy $\mathcal{S}$ exists is called the *connected search number* of G, denoted by $\mathtt{cs}(G)$. The minimum number of k searchers such that there exists a monotone connected k-search for G is called the *monotone connected search number* of G, and is denoted by $\mathtt{mcs}(G)$. If a (monotone) connected search strategy $\mathcal{S}$ uses ($\mathtt{mcs}(G)$) $\mathtt{cs}(G)$ searchers, then $\mathcal{S}$ is called an *optimal* (monotone) connected search strategy for G. Of particular interest are the connections between the searching numbers. Clearly, $\mathtt{cs}(G) \geq \mathtt{s}(G)$ for each graph G, since each connected search strategy is also a search strategy. For upper bounds on connected search number see e.g. [2,10,15,18].

It has been proven that recontamination does help for connected searching [35], and the difference between $\mathtt{cs}(G)$ and $\mathtt{mcs}(G)$ can be arbitrarily large for some graphs G [35]. However, if T is a tree then $\mathtt{cs}(T) = \mathtt{mcs}(T)$ [2]. Several algorithmic results for connected searching of special classes of graphs are known, including (unweighted) trees [2], chordal graphs [29], hypercubes [14,12], a pyramid [32], chordal rings and tori [13], or outerplanar graphs [17]. For results on searching planar graphs with small number of searchers and small number of connected components of the cleared subgraph see [30]. The non-connected searching problem for weighted trees has been proven to be NP-complete [28].

For different models of weighted graph searching and related robotic pursuit/evasion problems see e.g. [5,20,21,23,26,34].

1.2. The summary of the results

Authors in [3] claimed an efficient algorithm for connected searching of weighted trees. However, due to some flaws in the algorithm, it does not always produce an optimal solution (see [22,25] and Fig. 2 in Section 3 for some examples), which results in a heuristic. The complexity status of searching weighted trees turns out to be NP-complete, which we prove in this work. This gives a motivation for finding non-trivial subclasses of trees that are computationally tractable. From the NP-completeness proof presented here it follows that if a tree has been partially cleared, i.e. for a given vertex v a subset X of edges incident to v is contaminated, then finding the order of clearing the edges in X in an optimal connected search strategy is, in general, 'as difficult' as finding the strategy itself. For this reason we focus on an algorithm designed for bounded degree trees. However, unlike in the case of searching unweighted trees (both in the classical and connected models), if several pairwise disjoint subtrees of the tree to search are contaminated, then, in general, a connected search strategy that clears them sequentially (i.e. clears all the edges of one tree and then proceeds to clearing another tree) might not be optimal. We design an algorithm using the dynamic programming method (we keep a collection of search strategies for some subtrees) together with a greedy rule that allow us to narrow down the search space and derive a polynomial running time for bounded degree trees with arbitrary weight functions. This in particular implies that the connected searching problem for weighted trees is FPT on instances of bounded maximum degree.

This paper is organized as follows. In the next section we give the necessary definitions. In Section 3 we analyze the basic properties of connected searching of weighted trees. Then, in Section 4, we give an algorithm for computing optimal search strategies for weighted trees. The algorithm is exponential in the maximum degree of a tree. Thus, it is designed for trees of bounded degree. Section 5 deals with the complexity of searching trees. In Section 5.2 we prove that finding an optimal connected search of a weighted tree is strongly NP-hard, i.e. it is NP-hard for trees with integer weight functions with polynomially (in the size of the tree) bounded values on the vertices and edges. This justifies the exponential, in general, running time of the algorithm. In order to present the proof we need a preliminary result that a special instance of scheduling time-dependent tasks is NP-complete, which is proven in Section 5.1.

2. Preliminaries

In the following we assume that all graphs $G = (V(G), E(G), w)$ are connected, i.e. there exists a path between each pair of vertices of G. The sets $V(G)$ and $E(G)$ are, respectively, the vertices and the edges of G, while $w: V(G) \cup E(G) \rightarrow \mathbb{N}_+$ is a weight function. ($\mathbb{N}_+$ is the set of positive integers.) For the standard graph theoretic notation used in this paper see e.g. [11,33].

Now we give a formal definition of connected searching. Let $k \geq 0$ be an integer. Initially all the edges of a weighted graph $G = (V(G), E(G), w)$ are *contaminated*. A *connected k-search strategy* $\mathcal{S}$ selects a vertex $v_0 \in V(G)$, called the *homebase*, and places k searchers on v_0. Each move of $\mathcal{S}$ consists of sliding $j \geq 1$ searchers along an edge $e \in E(G)$. If e is contaminated, then we require $j \geq w(e)$, and e becomes *clear* as a result of the move. An edge $uv \in E(G)$ becomes *contaminated* if there exists an edge vy that can contain the fugitive (i.e. vy is contaminated) and less than $w(v)$ searchers occupy v. The set of clear edges has to form a connected subgraph after each move of $\mathcal{S}$. After the last move of $\mathcal{S}$ all the edges of G are clear. If no edge becomes recontaminated during $\mathcal{S}$, then $\mathcal{S}$ is called *monotone*. In the Connected Searching (CS) problem we ask, for the given G and k, whether there exists a connected k-search strategy for G. The minimum integer k such that there exists a (monotone) connected k-search strategy for G is the (*monotone*) *connected search number* of G, denoted by ($\mathtt{mcs}(G)$) $\mathtt{cs}(G)$. A (monotone) connected ($\mathtt{mcs}(G)$-) $\mathtt{cs}(G)$-search strategy is called *optimal*. In the optimization version of CS the goal is to minimize the number of searchers needed to clear G.

Given any strategy $\mathcal{S}$, $\mathtt{s}(\mathcal{S})$ is the number of searchers used by $\mathcal{S}$, $|\mathcal{S}|$ is the number of moves in $\mathcal{S}$ and $\mathcal{S}[i]$ is its ith move, $1 \leq i \leq |\mathcal{S}|$. For each $i = 1, \ldots, |\mathcal{S}|$, $\delta(\mathcal{S}[i])$ is the set of vertices v, occupied by searchers at the end of move i, such that there exists a contaminated edge incident to v. We say that the vertices in $\delta(\mathcal{S}[i])$ are *guarded* in step i. Thus, if at the end of move $\mathcal{S}[i]$ there exists a vertex $v \in \delta(\mathcal{S}[i])$ and less than $w(v)$ searchers occupy v, then a recontamination occurs.

Forcing a connected search strategy to select different homebases results in different number of searchers required to clear a graph G. The problem where the homebase is a part of the input is denoted by CSFH (Connected Searching problem with Fixed Homebase).

The number of searchers used for guarding at the end of step $\mathcal{S}[i]$ is denoted by $|\mathcal{S}[i]|$. Note that

$$|\mathcal{S}[i]| = \sum_{v\in\delta(\mathcal{S}[i])} w(v).$$

The searchers which are not used for guarding in a given step $\mathcal{S}[i]$ are called *free* searchers in step i. In particular, if more than $w(v)$ searchers occupy $v \in \delta(\mathcal{S}[i])$, then $w(v)$ of them are guarding v, while the remaining ones are considered to be free. Free searchers can move arbitrarily along the clear edges until the next move $\mathcal{S}[i']$, $i' > i$, which clears an edge uv, where $u \in \delta(\mathcal{S}[i])$. The move $\mathcal{S}[i']$ can be performed only if the required number of j searchers (with j' free searchers among them), which will slide along uv in $\mathcal{S}[i']$, is at u. So, each move among $\mathcal{S}[i+1], \ldots, \mathcal{S}[i'-1]$ which is not necessary for gathering the j searchers for clearing uv in $\mathcal{S}[i']$ can be performed after $\mathcal{S}[i']$. Moreover, each set of j' searchers, which are free at the end of move $\mathcal{S}[i]$, can be used to clear uv in $\mathcal{S}[i']$. For this reason, we do not list the moves of sliding searchers along clear edges. Thus, due to this simplifying assumption, $|\mathcal{S}| = |E(G)|$ for each monotone search strategy $\mathcal{S}$.

We say that a strategy is *partial* if it clears a subset of the edges of G. Given a search strategy $\mathcal{S}$ for G, the symbol $\mathcal{S}[\preceq i]$, $i \in \{1, \ldots, |\mathcal{S}|\}$, is used to denote the partial search strategy consisting of the moves $\mathcal{S}[1], \ldots, \mathcal{S}[i]$. Clearly, if $\mathcal{S}$ is connected, then $\mathcal{S}[\preceq i]$ is also connected (with the same homebase). Given a partial search strategy $\mathcal{S}'$, we extend our notation so that $\delta(\mathcal{S}')$ is the set of guarded vertices after the last move of $\mathcal{S}'$, $\delta(\mathcal{S}') = \delta(\mathcal{S}'[|\mathcal{S}'|])$. The symbol $C_E(\mathcal{S}')$ denotes the set of edges cleared by a partial strategy $\mathcal{S}'$. In particular, if $\mathcal{S}$ clears G, then $\delta(\mathcal{S}) = \emptyset$ and $C_E(\mathcal{S}) = E(G)$.

3. Searching trees — basic properties

We will make several simplifying assumptions on connected search strategies restricted to weighted trees $T = (V(T), E(T), w)$. We use the symbol $\texttt{cs}(T, r)$ to denote the minimum number of searchers needed to clear T when r is the homebase. Then,

$$\texttt{cs}(T) = \min\{\texttt{cs}(T, v) : v \in V(T)\}. \tag{1}$$

To simplify the notation, all considered trees T are rooted at the homebase $r \in V(T)$. In the remaining part of this paper we consider the CSFH problem with the homebase r, unless stated otherwise.

Given a tree $T = (V(T), E(T), w)$ rooted at $r \in V(T)$, E_v is the set of edges between v and its children, $v \in V(T)$, and T_v is the subtree of T rooted at v.

For each tree T it holds $\texttt{mcs}(T) = \texttt{cs}(T)$ [2,3]. Thus, in what follows each connected search strategy is monotone. As mentioned in Section 2, we only list the clearing moves of a search strategy $\mathcal{S}$, which implies $|\mathcal{S}| = |E(T)|$.

Consider a connected search strategy $\mathcal{S}$ for T. Let $\mathcal{S}[i]$ be a move of clearing an edge uv. If v is a leaf and $v \neq r$, then the number of searchers that need to slide along uv to clear it in step $\mathcal{S}[i]$ is $w(uv)$. When uv gets clear at the end of move $\mathcal{S}[i]$, there is no need to guard v, which means that the searchers that reach v in $\mathcal{S}[i]$ are free at the end of the move $\mathcal{S}[i]$. This holds regardless of the weight of v, $w(v)$. Similarly, if v is a leaf and $v = r$, then $i = 1$ and $\max\{w(uv), w(u)\}$ searchers suffice to clear uv, and r does not have to be guarded at the end of move $\mathcal{S}[1]$. So, we may w.l.o.g. assume that

$$w(v) = 1 \quad \text{for each leaf } v \in V(T). \tag{2}$$

Given a connected search strategy $\mathcal{S}$ for T with homebase r, consider a move $\mathcal{S}[i]$ of clearing an edge uv, where v is a child of u. At the beginning of $\mathcal{S}[i]$ the vertex v is unoccupied and u is guarded by $w(u)$ searchers. To clear uv we need to slide $\max\{w(uv), w(v)\}$ searchers along uv. If $w(uv) < w(v)$, then by (2) v is not a leaf of T, which means that at the end of move $\mathcal{S}[i]$ at least $w(v)$ searchers have to occupy v. This means that we have to slide $w(v)$ searchers along uv regardless of $w(uv)$. Thus, for each edge uv, where u is the parent of v we w.l.o.g. obtain

$$w(uv) \geq w(v). \tag{3}$$

Our next simplifying assumption is considering the CS and CSFH problems for node-weighted trees only, and we argue that it does not lead to the loss generality. Consider now a new tree $T' = (V(T'), E(T'), w')$ obtained from T by replacing each edge uv by two edges ux_{uv} and vx_{uv}, where x_{uv} is a new vertex of T' corresponding to the edge uv of T (in other words, we subdivide the edges of T to obtain T'). Let $w'(ux_{uv}) = w'(vx_{uv}) = 1$ and $w(x_{uv}) = w(uv)$ for each $uv \in E(T)$ and let $w'(v) = w(v)$ for each $v \in V(T)$. Clearly, $|E(T')| = 2|E(T)|$.

For an example of all the transformations given above see Fig. 1.

Lemma 1. *For each T and its corresponding tree T', $\texttt{cs}(T', r) = \texttt{cs}(T, r)$ for each $r \in V(T)$.*

Proof. Given a connected search strategy $\mathcal{S}$ for T, we construct a connected search strategy $\mathcal{S}'$ for T' as follows. Each move $\mathcal{S}[i]$, $1 \leq i \leq |\mathcal{S}|$, clearing an edge uv, where u is the parent of v, is replaced by two moves $\mathcal{S}'[2i-1]$ and $\mathcal{S}[2i]$ of clearing the edges ux_{uv} and vx_{uv}, respectively. A simple induction on the number of moves in $\mathcal{S}$ allows us to prove that $\texttt{s}(\mathcal{S}') = \texttt{s}(\mathcal{S})$. Indeed, by (3), clearing uv in $\mathcal{S}$ requires $w(uv)$ searchers excluding the searchers used for guarding, and by the definition of

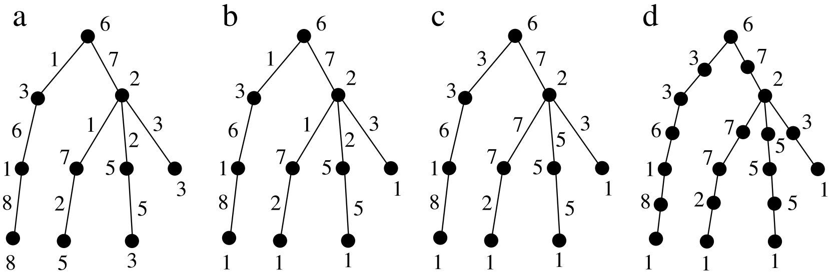

Fig. 1. (a) A rooted tree with node and edge weights; (b) the weight of each leaf is 1; (c) the corresponding tree satisfying (3); (d) the node-weighted tree T' obtained from T.

T', $w(uv)$ searchers are sufficient to clear ux_{uv} and vx_{uv} resulting in the same set of guarded vertices in $\mathcal{S}$ and $\mathcal{S}'$ after moves $\mathcal{S}[i]$ and $\mathcal{S}'[2i]$, respectively. This proves that $\mathtt{cs}(T', r) \leq \mathtt{cs}(T, r)$.

Let $\mathcal{S}'$ be a connected search strategy for T'. We may w.l.o.g. assume that if $\mathcal{S}'[i]$ clears an edge ux_{uv}, where x_{uv} is a child of u then, a move of clearing vx_{uv} follows, because $w(v) \leq w(vx_{uv})$ by (3). Two consecutive moves of clearing ux_{uv} and vx_{uv} in $\mathcal{S}'$ can be translated into clearing uv in a connected search strategy which requires $w(uv) = w'(x_{uv})$ searchers. Thus, $\mathtt{s}(\mathcal{S}) = \mathtt{s}(\mathcal{S}')$, and consequently $\mathtt{cs}(T, r) \leq \mathtt{cs}(T', r)$. This proves that $\mathtt{cs}(T, r) = \mathtt{cs}(T', r)$. □

In the remaining part of this paper we assume that the weight of each edge $e \in E(T)$ is 1.

Definition 1. Let $\mathcal{S}$ and $\mathcal{S}'$ be partial (not necessarily connected) search strategies for T and $T - C_{\mathrm{E}}(\mathcal{S})$, respectively. Let $R \subseteq \delta(\mathcal{S})$ be the set of vertices initially occupied in $\mathcal{S}'$. We define a search strategy $\mathcal{S} \oplus \mathcal{S}'$ as follows:

1. $(\mathcal{S} \oplus \mathcal{S}')[i] = \mathcal{S}[i]$ and $\delta((\mathcal{S} \oplus \mathcal{S}')[i]) = \delta(\mathcal{S}[i])$ for each $i = 1, \ldots, |\mathcal{S}|$,
2. $(\mathcal{S} \oplus \mathcal{S}')[|\mathcal{S}| + i]$, $i = 1, \ldots, |\mathcal{S}'|$, clears the edge cleared in the move $\mathcal{S}'[i]$, while the set of guarded vertices at the end of the move $(\mathcal{S} \oplus \mathcal{S}')[|\mathcal{S}| + i]$ is $\delta((\mathcal{S} \oplus \mathcal{S}')[|\mathcal{S}| + i]) = (\delta(\mathcal{S}) \setminus R) \cup \delta(\mathcal{S}'[i])$.

In other words, $\mathcal{S} \oplus \mathcal{S}'$ clears all the edges cleared by $\mathcal{S}$ and $\mathcal{S}'$ in the order corresponding to the moves $\mathcal{S}[1], \ldots, \mathcal{S}[|\mathcal{S}|], \mathcal{S}'[1], \ldots, \mathcal{S}'[|\mathcal{S}'|]$. Note that in particular $C_{\mathrm{E}}((\mathcal{S} \oplus \mathcal{S}')[\preceq i]) = C_{\mathrm{E}}(\mathcal{S}[\preceq i])$ for each $i = 1, \ldots, |\mathcal{S}|$, and $C_{\mathrm{E}}((\mathcal{S} \oplus \mathcal{S}')[\preceq (|\mathcal{S}| + i)]) = C_{\mathrm{E}}(\mathcal{S}) \cup C_{\mathrm{E}}(\mathcal{S}'[\preceq i])$ for each $i = 1, \ldots, |\mathcal{S}'|$. Furthermore, for $\mathcal{S} \oplus \mathcal{S}'$ to be a partial connected search with homebase r, $\mathcal{S}$ has to be a partial connected search with homebase r, while $\mathcal{S}'$ does not have to be connected, but the choice of R guarantees that each subgraph cleared by $\mathcal{S}'$ has a common vertex with $\delta(\mathcal{S})$.

Definition 2. Suppose that we are given a tree T rooted at a homebase r, a vertex $v \in V(T)$, and an integer $k \geq 0$. We say that a partial connected k-search strategy $\mathcal{S}_v$ for T_v with homebase v, $v \in V(T)$, is (k, v)-*minimal* if $w(\delta(\mathcal{S}_v)) \leq w(v)$ and $w(\delta(\mathcal{S}_v)) \leq w(\delta(\mathcal{S}'_v))$ for each partial connected k-search $\mathcal{S}'_v$ for T_v with homebase v.

It follows from the definition that a (k, v)-minimal search strategy is also assumed to be partial and connected.

Before we continue, we give an informal description of the above concept of minimal strategies. A (k, v)-minimal search strategy $\mathcal{S}_v$ (v is the homebase) partially clears T_v and uses at most k searchers. When $\mathcal{S}_v$ finishes, i.e. at the end of its last move, the total weight of all guarded vertices in T_v is not greater than $w(v)$ (guarding is necessary to protect the cleared subgraph from recontamination, which can occur due to the edges that have not been cleared by $\mathcal{S}_v$). Moreover, $\mathcal{S}_v$ is the 'best' strategy that achieves that, i.e. no partial connected k-search strategy with homebase v can 'reach' in T_v a 'border' of smaller total weight than that of $\mathcal{S}_v$.

A strategy $\mathcal{S}_v$ is *not minimal* if there exists no k such that $\mathcal{S}$ is (k, v)-minimal. A partial connected search strategy $\mathcal{S}$ for T_r can be *extended* to a (k, r)-minimal search strategy for T_r if there exists a search strategy $\mathcal{S}'$ such that $\mathcal{S} \oplus \mathcal{S}'$ is a (k, r)-minimal search strategy for T_r. The latter in particular implies that $\mathtt{s}(\mathcal{S}) \leq k$. Given a tree T_r and $E' \subseteq E(T_r)$, $T_r - E'$ is the set of maximal rooted subtrees induced by the edges in $E(T_r) \setminus E'$.

The following lemma will be used to simplify our method of extending a partial search strategy $\mathcal{S}$ to obtain a minimal one. Informally speaking, we select a vertex v in $\delta(\mathcal{S})$ and extend $\mathcal{S}$ by using a (k', v)-minimal strategy $\mathcal{S}_v$ for T_v, where k' is selected to be the maximum number of searchers that $\mathcal{S}_v$ can use so that $\mathtt{s}(\mathcal{S} \oplus \mathcal{S}_v)$ is at most the desired number of k searchers.

Lemma 2. *A partial non-minimal connected search strategy $\mathcal{S}$ for T_r can be extended to a (k, r)-minimal search strategy for T_r if and only if there exist T'_v (rooted at v) in $T - C_{\mathrm{E}}(\mathcal{S})$ and a $(k - w(\delta(\mathcal{S}) \setminus \{v\}), v)$-minimal search strategy $\mathcal{S}_v$ for T'_v, such that $\mathcal{S} \oplus \mathcal{S}_v$ can be extended to a (k, r)-minimal search for T_r. Moreover, for each such T'_v and $\mathcal{S}_v$ the strategy $\mathcal{S} \oplus \mathcal{S}_v$ can be extended to a (k, r)-minimal search for T_r.*

Proof. The "only if" part is obvious. To prove the "if" part let $\mathcal{S} \oplus \mathcal{S}_1$ be a (k, r)-minimal search for T_r. For each $v \in \delta(\mathcal{S})$ there exists a contaminated edge in E_v, which implies that there exists a nonempty subtree T'_v in $T_r - C_{\mathrm{E}}(\mathcal{S})$ rooted at v. (If all edges in E_v are contaminated, then $T'_v = T_v$.) First we argue that there exist $v \in \delta(\mathcal{S})$ and a $(k - w(\delta(\mathcal{S}) \setminus \{v\}), v)$-minimal search strategy $\mathcal{S}_v$ for T'_v. For each $v \in \delta(\mathcal{S})$ and for each move $\mathcal{S}_1[i]$ define $B(i, v) = \delta(\mathcal{S}_1[i]) \cap V(T'_v)$. Find the minimum l such that $w(B(l, v)) < w(v)$ for some $v \in \delta(\mathcal{S})$. Such an integer l does exist, because otherwise $w(\delta(\mathcal{S} \oplus \mathcal{S}_1)) \geq \delta(\mathcal{S})$ which contradicts

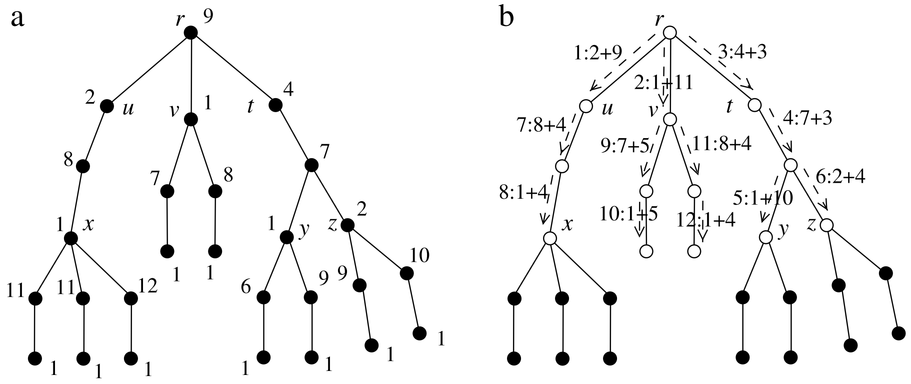

Fig. 2. (a) A node-weighted tree T_r; (b) $\mathcal{S} \oplus \mathcal{S}_t \oplus \mathcal{S}_u \oplus \mathcal{S}_v$.

the minimality of $\mathcal{S} \oplus \mathcal{S}_1$. Let $\mathcal{S}'_v$ be $\mathcal{S}_1$ restricted to clearing the edges in $C_{\mathrm{E}}(\mathcal{S}_1[\preceq l]) \cap E(T'_v)$ in the same order as they are cleared by $\mathcal{S}_1$. $\mathcal{S} \oplus \mathcal{S}'_v$ uses at most k searchers (which gives that $\mathtt{s}(\mathcal{S}'_v) \leq k - w(\delta(\mathcal{S}) \setminus \{v\})$), and $w(\delta(\mathcal{S}'_v)) = w(B(l, v)) < w(v)$. So, the set of $(k - w(\delta(\mathcal{S}) \setminus \{v\}))$-search strategies $\mathcal{S}'_v$ for T'_v satisfying $w(\delta(\mathcal{S}'_v)) < w(v)$ is nonempty and, by the definition, a strategy $\mathcal{S}_v$ with the minimum $w(\delta(\mathcal{S}_v))$ is $(k - w(\delta(\mathcal{S}) \setminus \{v\}), v)$-minimal.

Let $\mathcal{S}_v$ and T'_v that satisfy the conditions given in the lemma be selected arbitrarily. We will use $\mathcal{S}_1$ to extend $\mathcal{S} \oplus \mathcal{S}_v$ to a (k, r)-minimal search $\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2$ for T_r. To obtain $\mathcal{S}_2$ we simply remove from $\mathcal{S}_1$ all the operations of clearing the edges in $C_{\mathrm{E}}(\mathcal{S}_v)$, preserving the order of clearing the remaining edges in $\mathcal{S}_1$. One can prove that $\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2$ is connected.

It remains to prove that $\mathtt{s}(\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2) \leq k$. By the definition, $\mathtt{s}(\mathcal{S} \oplus \mathcal{S}_v) \leq k$, so let us consider a move $(\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2)[i_2]$ of clearing an edge e, $i_2 > |\mathcal{S} \oplus \mathcal{S}_v|$. Select $i_1 > |\mathcal{S}|$ so that $(\mathcal{S} \oplus \mathcal{S}_1)[i_1]$ is the move of clearing e. It is sufficient to prove that $|(\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2)[i_2]| \leq |(\mathcal{S} \oplus \mathcal{S}_1)[i_1]|$. Let

$$U = \delta((\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2)[i_2]) \setminus \delta((\mathcal{S} \oplus \mathcal{S}_1)[i_1]). \tag{4}$$

In other words, U is the set of vertices guarded in step i_2 of $\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2$ but unguarded in step i_1 of $\mathcal{S} \oplus \mathcal{S}_1$. Clearly, $U \subseteq \delta(\mathcal{S}_v)$. For each $u \in U$ there exists a vertex $x_u \in \delta((\mathcal{S} \oplus \mathcal{S}_1)[i_1])$ on the path connecting v and u in T'_v, because $C_{\mathrm{E}}((\mathcal{S} \oplus \mathcal{S}_1)[\preceq i_1]) \cap E(T'_v) \subseteq C_{\mathrm{E}}((\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2)[\preceq i_2]) \cap E(T'_v)$. Let X_U be the set of all such vertices x_u, $u \in U$. We argue that

$$w(X_U) \geq w(U). \tag{5}$$

Suppose for a contradiction that (5) does not hold. Find a set X, with minimum $w(X)$, such that each path connecting v and u, $u \in \delta(\mathcal{S}_v)$, contains a vertex in X (possibly u). We obtain $w(X) < w(\delta(\mathcal{S}_v))$, because $w((\delta(\mathcal{S}_v) \setminus U) \cup X_U) < w(\delta(\mathcal{S}_v))$ and by the minimality of X, $w(X) \leq w((\delta(\mathcal{S}_v) \setminus U) \cup X_U)$. Define $\mathcal{S}'_v$ which clears the edges in

$$C_{\mathrm{E}}(\mathcal{S}_v) \setminus \bigcup_{x \in X} E(T_x)$$

in the same order as they are cleared in $\mathcal{S}_v$. Then, $\mathtt{s}(\mathcal{S}'_v) \leq \mathtt{s}(\mathcal{S}_v)$ and $w(\delta(\mathcal{S}'_v)) = w(X) < w(\delta(\mathcal{S}_v))$. Thus, $\mathcal{S}_v$ is not $(k - w(\delta(\mathcal{S}) \setminus \{v\}), v)$-minimal — a contradiction, which proves (5). Hence, $|(\mathcal{S} \oplus \mathcal{S}_v \oplus \mathcal{S}_2)[i_2]| \leq |(\mathcal{S} \oplus \mathcal{S}_1)[i_1]| \leq \mathtt{cs}(T, r)$. Since i_2 has been chosen arbitrarily, we have proven the thesis. $\square$

As an example consider a tree in Fig. 2(a). Let $\mathcal{S}$ be a partial strategy clearing the edges ru, rv, rt (in this order). $\mathtt{s}(\mathcal{S}) = 12$ and $\delta(\mathcal{S}) = \{u, v, t\}$. Denote by $\mathcal{S}_v$ the $(8, v)$-minimal search strategy for T_v ($\mathcal{S}_v$ completely clears the left branch of T_v first, for otherwise more than 8 searchers will be used). Sliding the searchers searcher along the two edges from u to x gives a $(8, u)$-minimal strategy $\mathcal{S}_u$ for T_u with $\delta(\mathcal{S}_u) = \{x\}$. There exists a $(8, t)$-minimal search $\mathcal{S}_t$ for T_t, where $\delta(\mathcal{S}_t) = \{y, z\}$ ($\mathcal{S}_t$ clears the three edges on the paths connecting t and y, z). Fig. 2(b) depicts a partial strategy $\mathcal{S} \oplus \mathcal{S}_t \oplus \mathcal{S}_u \oplus \mathcal{S}_v$, where the dashed arrows represent the moves of the strategy. Their labels $i : c + g$ indicate the number i of the clearing move, while c and g are, respectively, the number of searchers that move along the corresponding edge and guard other vertices in this move.

Observe that $\mathcal{S} \oplus \mathcal{S}_t \oplus \mathcal{S}_u \oplus \mathcal{S}_v$ can be extended to a connected 12-search strategy for T by clearing the subtrees T_y, T_z and T_x (in this order, and each of those subtrees is cleared by processing its branches from left to right).

The algorithm in [3] clears any subtree T_v in the following way. While guarding the root v of T_v, a child u of v is selected and the algorithm clears T_u completely. Once this is achieved, it proceeds to the next child of v and repeats this step until all children are processed. Note that clearing the subtrees T_u, T_v and T_t in Fig. 2 requires 12, 8 and 11 searchers, respectively. Therefore, clearing any of them while guarding r results in a search strategy using more than 12 searches (regardless of the order of clearing T_u, T_v and T_t), which implies that the algorithm for searching weighted trees presented in [3] is not optimal.

Also note that if T is a rooted full binary n-node tree with each vertex with even (odd) distance from the root having the weight K (1, respectively), and with each edge having the weight equal to 1, then the above-mentioned algorithm is forced to use $\Omega(K \log n)$ searchers for T. A modified method, where one immediately proceeds to both children whenever a vertex of weight K becomes guarded, never guards two vertices of weight K simultaneously. Therefore, the number of searchers used is $O(K + \log n)$, which means that the algorithm in [3] is not a constant factor approximation.

4. An efficient algorithm for bounded degree trees

In Sections 4 and 5 we provide the algorithm for CSFH problem on bounded degree trees and a polynomial-time reduction from a NP-complete problem to the CSFH problem for general trees. In both cases we conclude that the corresponding result (an efficient algorithm or a polynomial-time reduction) holds for the CS problem on trees as well.

In an informal way, our method for clearing T_r may be described as follows. We start by placing k searchers at the root r. Assume that the algorithm calculated a partial search strategy $\mathcal{S}$. If $\delta(\mathcal{S}) = \emptyset$ then $\mathcal{S}$ clears T_r and the computation stops. Otherwise we select a vertex $v \in \delta(\mathcal{S})$ and we find a partial connected search $\mathcal{S}_v$ for T_v. We continue with $\mathcal{S} \oplus \mathcal{S}_v$. Note that $\mathcal{S} \oplus \mathcal{S}_v$ requires $\mathtt{s}(\mathcal{S})$ searchers to perform $\mathcal{S}$ and then the moves of $\mathcal{S}_v$ follow, where $w(\delta(\mathcal{S}) \setminus \{v\})$ searchers are used to guard the vertices that are not in T_v and, in addition, $\mathtt{s}(\mathcal{S}_v)$ searchers work on the subtree T_v. So, if $\mathcal{S}$ can be extended to a connected k-search for T_r and we are able to find a $(k - w(\delta(\mathcal{S}) \setminus \{v\}), v)$-minimal strategy $\mathcal{S}_v$, then, by Lemma 2, $\mathcal{S} \oplus \mathcal{S}_v$ can be also extended to a connected k-search for T_r. The fact that any such vertex v is sufficient reduces the size of the search space for the algorithm. However, it follows immediately from the NP-completeness proof in Section 5 that finding a strategy $\mathcal{S}_v$ is intractable, unless $P = NP$.

For each $v \in V(T_r)$ a set $\mathcal{C}_v$ is a global variable and will contain (k, v)-minimal search strategies for a subtree T_v and for selected values of k.

We start by describing a procedure, called `MCPS` (*Minimal Connected Partial Strategy*), which for a given integer k, a rooted tree T_r, and an ordering $rv_1, \ldots, rv_d$ of the edges in E_r, finds a (k, r)-minimal search strategy $\mathcal{S}$, which clears the edges in E_r according to the given order, whenever such a strategy exists. Our final algorithm will process T_r in a bottom-up fashion, so when `MCPS` is called, then for each $v \in V(T_r) \setminus \{r\}$ some (k', v)-minimal search strategies for T_v belong to $\mathcal{C}_v$ for some integers k'. Moreover, $w(r)$ searchers already occupy r when `MCPS` starts. The procedure is as follows:

Procedure `MCPS`$(T_r, k, (rv_1, \ldots, rv_d))$ (*Minimal Connected Partial Strategy*)
Input: A weighted tree T_r, an integer k, and an ordering of the edges in E_r.
Output: A (k, r)-minimal search strategy for T_r that clears the edges in E_r in the given order; 'failure' when no such strategy exists.

Step 1. For each $i = 1, \ldots, d$ repeat the following:
- (i) if k searchers are sufficient to clear rv_i, then clear rv_i as the next step of $\mathcal{S}$ and find (k', v_i)-minimal search $\mathcal{S}_{v_i} \in \mathcal{C}_{v_i}$ with maximum k' such that $k' \le k - w(\delta(\mathcal{S}) \setminus \{v_i\})$. If $\mathcal{S}_{v_i}$ exists, then let $\mathcal{S} := \mathcal{S} \oplus \mathcal{S}_{v_i}$, otherwise proceed to $i + 1$;
- (ii) if more than k searchers are needed for the move of clearing rv_i, then return 'failure'.

Step 2. While there exist $v \in \delta(S)$ and $\mathcal{S}_v \in \mathcal{C}_v$ such that $\mathcal{S}_v$ is (k', v)-minimal, $k' \le k - w(\delta(\mathcal{S}) \setminus \{v\})$, then $\mathcal{S} := \mathcal{S} \oplus \mathcal{S}_v$.
Step 3. Return $\mathcal{S}$.

Lemma 3. *If there exists a (k, r)-minimal search strategy for T_r that clears the edges in E_r according to the order $\pi = (rv_1, \ldots, rv_d)$, then* `MCPS` *returns such a strategy.*

Proof. Assume that there exists a (k, r)-minimal search strategy $\mathcal{S}_{\mathrm{opt}}$ clearing the edges in E_r according to the order π. Let, for brevity, $\mathcal{S}_i$ denote the partial connected search strategy calculated in Step 1 of `MCPS`, where clearing rv_i is the last move of $\mathcal{S}_i$, $i = 1, \ldots, d$.

We use an induction on $i = 1, \ldots, d$ to prove that $\mathcal{S}_i$ can be extended to (k, r)-minimal search for T_r. The claim follows immediately for $i = 1$, since by assumption, $\mathcal{S}_{\mathrm{opt}}$ starts by clearing rv_1. (For a connected search starting at r an edge in E_r has to be cleared first.) Assume that rv_i has been cleared by $\mathcal{S}_i$, $i < d$. The procedure `MCPS` proceeds in Step 1 by finding a $(k - w(\delta(\mathcal{S}_i) \setminus \{v_i\}), v_i)$-minimal search strategy $\mathcal{S}_{v_i}$ for T_{v_i}. If $\mathcal{S}_{v_i}$ exists, then by Lemma 2, $\mathcal{S}_i \oplus \mathcal{S}_{v_i}$ can be extended to a (k, r)-minimal search strategy for T_r. By the definition, there is no $v \in \delta(\mathcal{S}_i \oplus \mathcal{S}_{v_i}) \setminus \{r\}$ for which there exists a $(k - w(\delta(\mathcal{S}_i \oplus \mathcal{S}_{v_i}) \setminus \{v\}), v)$-minimal search for T_v. Thus, the next edge e cleared by $\mathcal{S}_i \oplus \mathcal{S}_{v_i}$ must be in E_r. On the other hand, if $\mathcal{S}_{v_i}$ does not exist, then the next edge e to clear is in E_r. Hence, in both cases $e = rv_{i+1}$ which results in strategy $\mathcal{S}_{i+1}$.

Thus, we obtain that $\mathcal{S}_d$ can be extended to a (k, r)-minimal search for T_r. Then, `MCPS` finds in the last iteration in Step 1 and in Step 2 a sequence of vertices $v_{d+1}, \ldots, v_{d+l}$ and search strategies $\mathcal{S}_{d+1}, \ldots, \mathcal{S}_{d+l}$ such that $\mathcal{S}_{d+i}$ is $(k - w(\delta(\mathcal{S}_d \oplus \cdots \oplus \mathcal{S}_{d+i-1}) \setminus \{v_{d+i}\}), v_{d+i})$-minimal and $v_{d+i} \in \delta(\mathcal{S}_d \oplus \cdots \oplus \mathcal{S}_{d+i-1})$. By Lemma 2, each strategy $\mathcal{S}_d \oplus \cdots \oplus \mathcal{S}_{d+i}$, $i = 0, \ldots, l$, can be extended to a (k, r)-minimal search for T.

Let for brevity $\mathcal{S} = \mathcal{S}_d \oplus \cdots \oplus \mathcal{S}_{d+l}$. We obtain that $\mathcal{S}$ is (k, r)-minimal, because otherwise, as proved above, it can be extended to a (k, v)-minimal search for T_r, and consequently, by Lemma 2, there exists $v \in \delta(\mathcal{S})$ and a $(k - w(\delta(\mathcal{S}) \setminus \{v\}), v)$-minimal search $\mathcal{S}_v$ such that $\mathcal{S} \oplus \mathcal{S}_v$ can be extended to a (k, v)-minimal search for T_r, which gives a contradiction with the fact that no such vertex has been found following v_{d+l} by `MCPS`. □

Now we are ready to give a listing of the algorithm `CTS` (*Connected Tree Searching*) for finding an optimal connected search strategy for a rooted tree T_r with homebase r. This algorithm is exponential in the maximum degree of T, $\Delta = \max\{\deg_T(v) : v \in V(T)\}$.

Procedure `CTS`(T_r) (*Connected Tree Searching*)
Input: A weighted tree T_r rooted at r.
Output: A collection $\mathcal{C}_r$ of partial connected search strategies with homebase r for T_r.

Step 1. Let initially $\mathcal{C}_r := \emptyset$. For each child v of r call $\mathcal{C}_v := \mathtt{CTS}(T_v)$.
Step 2. Fix a permutation $\pi = (rv_1, \ldots, rv_d)$ of the edges in E_r. Set $k := 1$. If Step 3 has been executed for all the $d!$ permutations π, then Exit.
Step 3. Call $\mathcal{S}_r := \mathtt{MCPS}(T_r, k, \pi)$. If 'failure' has been returned, then increase k and repeat Step 3.[1] Otherwise, if there is no $\mathcal{S} \in \mathcal{C}_r$ such that $w(\delta(\mathcal{S})) \leq w(\delta(\mathcal{S}_r))$ and $\mathtt{s}(\mathcal{S}) \leq \mathtt{s}(\mathcal{S}_r)$, then add $\mathcal{S}_r$ to $\mathcal{C}_r$ and remove from $\mathcal{C}_r$ all search strategies $\mathcal{S} \neq \mathcal{S}_r$ such that $w(\delta(\mathcal{S})) \geq w(\delta(\mathcal{S}_r))$ and $\mathtt{s}(\mathcal{S}) \geq \mathtt{s}(\mathcal{S}_r)$. If $\delta(\mathcal{S}_r) = \emptyset$ then go to Step 2 to fix the next permutation π. Otherwise increase k and repeat Step 3.
Step 4. Return $\mathcal{C}_r$.

Note that Step 1 of CTS guarantees that for each $v \in V(T) \setminus \{r\}$ the collection $\mathcal{C}_v$ of all minimal search strategies for T_v is calculated (which is required for subsequent calls of MCPS). The next lemma gives a characterization of the strategies that belong to $\mathcal{C}_r$ computed by CTS.

Lemma 4. *Let k be an integer. As a result of the execution of* $\mathtt{CTS}(T_r)$ *the set $\mathcal{C}_r$ contains a (k, r)-minimal search strategy for T_r whenever such a strategy exists.*

Proof. We prove the lemma by induction on the number of vertices of a tree. For a tree with one vertex the claim follows.

Let T be a tree with $n > 1$ vertices. By the induction hypothesis, after Step 1 of CTS, the set $\mathcal{C}_v$ contains a (k', v)-minimal search strategy for each $v \in V(T) \setminus \{r\}$ and for each $k' \geq 1$ whenever such a strategy exists.

Then, CTS iterates over all permutations π of the edges in E_r and for each permutation all integers k are used (we stop when a strategy clearing T_r has been found). Moreover, whenever a search strategy $\mathcal{S}$ is removed from $\mathcal{C}_r$ or is computed by CTS, but not added to $\mathcal{C}_r$, then the instructions in Step 3 of CTS guarantee that another search strategy $\mathcal{S}'$ belongs to $\mathcal{C}$, where $w(\delta(\mathcal{S}')) \leq w(\delta(\mathcal{S}))$ and $\mathtt{s}(\mathcal{S}') \leq \mathtt{s}(\mathcal{S})$. Thus, if $\mathcal{S}$ is (k, r)-minimal for some integer k, then so is $\mathcal{S}'$. Lemma 3 gives the thesis. □

Lemma 4 in particular implies that CTS finds an optimal solution to the CSFH problem, because an optimal connected search strategy $\mathcal{S}$ is $(\mathtt{cs}(T, r), r)$-minimal and $\delta(\mathcal{S}) = \emptyset$. We finish this section with some complexity remarks.

Lemma 5. *Let T_v be a tree rooted at v and let π be a permutation of the edges in E_v. If $\mathcal{S} = \mathtt{MCPS}(T_v, k, \pi)$ and $\mathcal{S}' = \mathtt{MCPS}(T_v, k', \pi)$, where $k \leq k'$, then $C_{\mathrm{E}}(\mathcal{S}) \subseteq C_{\mathrm{E}}(\mathcal{S}')$ and $w(\delta(\mathcal{S})) \geq w(\delta(\mathcal{S}'))$.*

Proof. First note that, by construction, each partial search strategy computed by MCPS has the property that for each vertex $u \in T_v$ either all or none of the edges in E_u have been cleared. Define A to be the set of vertices $u \in \delta(\mathcal{S})$ such that the edge connecting u to its parent has not been cleared by $\mathcal{S}'$. Let A' be the set of vertices $u \in \delta(\mathcal{S}')$ such that the edges in E_u have been cleared by $\mathcal{S}$. Informally speaking, A' consists of the vertices u in $\delta(\mathcal{S}')$ such that the strategy $\mathcal{S}$ managed to continue the search from u in T_u and the 'border' vertices reached in T_u by $\mathcal{S}$ belong to A. Clearly, $A = \emptyset$ if and only if $A' = \emptyset$.

We argue that $A = \emptyset$. Suppose for a contradiction that $A \neq \emptyset$. Let $i \in \{1, \ldots, |\mathcal{S}|\}$ be the index such that no edge in $\bigcup_{u \in A'} E_u$ belongs to $C_{\mathrm{E}}(\mathcal{S}[\preceq i])$ and $\mathcal{S}[i+1]$ clears an edge in E_u for some $u \in A'$ (recall that no edge in E_u has been cleared by $\mathcal{S}'$). By construction, $\mathcal{S} = \mathcal{S}[\preceq i] \oplus \mathcal{S}_u \oplus \mathcal{S}''$ for some (j, u)-minimal strategy $\mathcal{S}_u$ for T_u, where $j \leq k - w(\delta(\mathcal{S}[\preceq i]) \setminus \{u\})$. By the choice of i, $w(\delta(\mathcal{S}')) \leq w(\delta(\mathcal{S}[\preceq i]))$ and consequently $w(\delta(\mathcal{S}') \setminus \{u\}) \leq w(\delta(\mathcal{S}[\preceq i]) \setminus \{u\})$. Since $k' \geq k$,

$$\begin{aligned} \mathtt{s}(\mathcal{S}' \oplus \mathcal{S}_u) &\leq \max\{k', w(\delta(\mathcal{S}') \setminus \{u\}) + \mathtt{s}(\mathcal{S}_u)\} \\ &\leq \max\{k', w(\delta(\mathcal{S}[\preceq i]) \setminus \{u\}) + j\} \leq \max\{k', k\} = k'. \end{aligned}$$

Also by definition, $w(\delta(\mathcal{S}' \oplus \mathcal{S}_u)) \leq w(\delta(\mathcal{S}'))$. By Lemma 4, $\mathcal{S}_u \in \mathcal{C}_u$. Therefore, instead of returning $\mathcal{S}'$, the procedure MCPS finds in Step 3 the search strategy $\mathcal{S}_u$ and proceeds with $\mathcal{S}' \oplus \mathcal{S}_u$ – a contradiction. □

Lemma 6. *Given a tree T of maximum degree Δ, the running time of the algorithm* CTS *is $O(\Delta! n^3 \log(\Delta! n))$, where $n = |V(T)|$.*

Proof. By Lemma 5, for the given rooted tree T_v and a permutation π of the edges in E_v there are at most n different search strategies that can be returned by MCPS. Thus, $|\mathcal{C}_v| \leq \Delta! n$. We maintain $\mathcal{C}_v$ as a balanced binary search tree, where each node corresponds to a partial search strategy and the associated key value of the node is the number of searchers the strategy uses. This gives that inserting, removing and finding search strategies takes $O(\log(\Delta! n))$ time.

The running time of MCPS is $O(n \log(\Delta! n))$, because there at most $O(n)$ strategies in $\bigcup_{u \in V(T_v)} \mathcal{C}_u$ that have been used in Step 1 and Step 2 of MCPS to construct $\mathcal{S}$. The latter follows from the fact, that each such strategy in $\mathcal{C}_u$ 'contributes' to $\mathcal{S}$ by clearing at least one edge of the input tree T_v.

As to the complexity of CTS, we first analyze one repetition of Step 3 of CTS (i.e. its execution for fixed k and π). It takes $O(n \log(\Delta! n))$ time to execute MCPS, and $O(|\mathcal{C}_v| \log |\mathcal{C}_v|) = O(\Delta! n \log(\Delta! n))$ time to iterate over $\mathcal{C}_v$ to remove unnecessary strategies from the collection $\mathcal{C}_v$. Therefore, the running time of Step 3 of CTS for fixed k and π is $O(\Delta! n \log(\Delta! n))$.

By Lemma 5 it is enough to execute Step 3 of CTS for at most n values of k, so it remains to argue that we can compute them efficiently. To that end we modify MCPS so that each search in $\mathcal{C}_v$ in Step 1 and Step 2 that results in finding $\mathcal{S}_v$ is followed by one additional search that finds the 'successor' $\mathcal{S}'_v$ of $\mathcal{S}_v$ in $\mathcal{C}_r$. By construction, $\mathcal{S}'_v$ is (k'', v)-minimal and

[1] We define in the proof of Lemma 6 how k is increased, since in order to improve the complexity we do not try the integers consecutively.

$k'' > k - w(\delta(\mathcal{S}) \setminus \{v\})$. Thus, if we increase k by $k'' + w(\delta(\mathcal{S}) \setminus \{v\}) - k$, then we are guaranteed that MCPS returns a partial connected search strategy that clears more edges of T_r than the strategy returned for the initial value of k. Thus, we record the corresponding value of $k'' + w(\delta(\mathcal{S}) \setminus \{v\}) - k$ each time we extend the current search strategy $\mathcal{S}$ during the execution of MCPS. Then, we increase k in Step 3 of CTS by the minimum value recorded. This modification does not increase the complexity of MCPS. Therefore, the complexity of Step 3 of CTS is $O(\Delta!n^2 \log(\Delta!n))$.

The procedure CTS is called n times in total, once for each vertex. Thus, the overall execution time of CTS is $O(\Delta!n^3 \log(\Delta!n))$. □

Since the algorithm solves the CSFH problem, where the homebase is given, in order to solve the CS problem, a straightforward approach is to call CTS for each vertex of T as the homebase and the solution is the best strategy found. However, we can reduce the running time. For different roots $r \in V(T)$ for each $v \in V(T)$ there are at most $\deg_T(v) + 1$ different subtrees T_v for which CTS calculates search strategies, namely each neighbor of v can be its parent and v may be the root itself. This gives that there are in total at most $\sum_{v \in V(T)}(\deg_T(v) + 1) = 2|E(T)| + |V(T)| \leq 3n$ different subtrees T_v to consider. Our final remark is that CSFH has been designed to clear node-weighted trees. Thus, for clearing a tree T with non-unit edge weights we apply first to T the transformations from Section 3, which results in a tree T' with unit edge weights, $\texttt{cs}(T) = \texttt{cs}(T')$ and an optimal connected search strategy for T can be obtained on the basis of the strategy calculated by CTS for T' as described in the proof of Lemma 1.

Theorem 1. *Given a tree T of maximum degree Δ, an optimal connected search strategy for T can be computed in $O(\Delta!n^3 \log(\Delta!n))$ time, where $n = |V(T)|$.* □

Corollary 1. *Given a bounded degree T, an optimal connected search strategy for T can be computed in $O(n^3 \log n)$ time, where $n = |V(T)|$.* □

Corollary 2. *The problem of connected searching of weighted trees is fixed parameter tractable with respect to the maximum degree of the tree.* □

5. Connected searching of weighted trees is hard

5.1. Scheduling time-dependent tasks

In this section we recall a problem of scheduling time-dependent (deteriorating) tasks. The execution time of a task depends on its starting time. The set of tasks is denoted by $\mathcal{J} = \{J_1, \ldots, J_n\}$. Each task $J_j \in \mathcal{J}$ is characterized by two parameters, deadline d_j and running time p_j, which depends on s_j, the point of time when the execution of J_j starts. Since the execution time depends on the starting point, we will write $p_j(t)$ to refer to the execution time of J_j when it starts at $t \geq 0$. The completion time of J_j is $C_j = s_j + p_j(s_j)$. We are interested in the single machine scheduling. A schedule D is *feasible* if the completion time C_j of each task J_j is not greater than its deadline, $C_j \leq d_j$, and the execution intervals of two different tasks do not overlap. The *makespan* of a schedule D is $\mathrm{ms}(D) = \max\{C_j : J_j \in \mathcal{J}\}$. Observe that a schedule D can be described by a permutation $\pi_D : \{1, \ldots, |\mathcal{J}|\} \to \mathcal{J}$, because the idle times between the execution of two consecutive tasks are not necessary for non-decreasing (in time) execution times. In the Time-Dependent Scheduling (TDS) problem we ask whether there exists a feasible schedule for $\mathcal{J}$. A good survey and a more detailed description of this problem can be found in [8]. For a survey on scheduling problems and terminology see [4,6].

There are several NP-completeness results for very restricted (linear) functions for execution time of a task [7,27]. However, we need for the reduction described in the next subsection the TDS problem instances, such that each task starts and ends at integers, which are bounded by a polynomial in the number of tasks. This property does not follow directly from the reductions in [7,27]. For this reason we will prove NP-hardness of the TDS problem instances having the properties we need.

We will reduce the 3-partition problem [19] to TDS. The former one can be stated as follows. Given a positive integer B, a set of integers $A = \{a_1, \ldots, a_{3m}\}$ such that $\sum_{j=1,\ldots,3m} a_j = mB$ and $B/4 < a_j < B/2$ for each $j = 1, \ldots, 3m$, find subsets $A_1, \ldots, A_m$ of A such that $A = \bigcup_{i=1,\ldots,m} A_i$, $A_i \cap A_{i'} = \emptyset$ for $i \neq i'$, and $\sum_{a_j \in A_i} a_j = B$ for each $i = 1, \ldots, m$.

Given B and A of cardinality $3m$, we define the instance of the TDS problem. Let $L = mB^3 + Bm(m+1)/2$. To simplify the statements we partition the interval $[0, L)$ into intervals $I_1, \ldots, I_m$ as follows:

$$I_i = \left[(i-1)B^3 + \frac{(i-1)i}{2}B, iB^3 + \frac{i(i+1)}{2}B\right), \quad i = 1, \ldots, m. \tag{6}$$

We use the symbols l_i, r_i to denote the endpoints of an interval I_i, i.e. $I_i = [l_i, r_i)$, $i = 1, \ldots, m$. Clearly, $\bigcup_{i=1,\ldots,m} I_i = [0, L)$ and $r_i = l_{i+1}$ for each $i = 1, \ldots, m-1$. Note that the length of I_i is $|I_i| = B^3 + iB$ for each $i = 1, \ldots, m$.

Now we define the tasks in the TDS problem. For each $a_j \in A$ we introduce a task $J_j \in \mathcal{J}$ with parameters

$$d_j = L, \quad \text{and} \quad p_j(t) = ia_j \quad \text{for each } t \in I_i.$$

In addition, for each $i = 1, \ldots, m$ we define a task $\widetilde{J}_i$ with deadline $\widetilde{d}_i$ and execution time $\widetilde{p}_i$, where

$$\widetilde{d}_i = l_i + B^3, \quad \text{and} \quad \widetilde{p}_i(t) = B^3 \quad \text{for each } t \geq 0,$$

$i = 1, \ldots, m$. Let $\widetilde{\mathcal{J}} = \{\widetilde{J}_1, \ldots, \widetilde{J}_m\}$. Observe that in each valid schedule all tasks are executed within $[0, L]$.

For a given schedule D for $\mathcal{J} \cup \widetilde{\mathcal{J}}$, s_j and C_j denote, respectively, the start and completion time of $J_j \in \mathcal{J}$. Similarly, $\widetilde{s}_i$ and $\widetilde{C}_i$ are start and completion times of $\widetilde{J}_i \in \widetilde{\mathcal{J}}$. We say that a task J *precedes* J' in a given schedule if J starts earlier than J'.

In the next three lemmas we prove several properties of every schedule for $\mathcal{J} \cup \widetilde{\mathcal{J}}$. Then, in Lemma 10 we prove that there exists a schedule for $\mathcal{J} \cup \widetilde{\mathcal{J}}$ if and only if there exists a 3-partition for B and A of cardinality $3m$.

Lemma 7. *In each schedule D for $\mathcal{J} \cup \widetilde{\mathcal{J}}$ the task $\widetilde{J}_i$ precedes $\widetilde{J}_{i+1}$ for each $i = 1, \ldots, m-1$.*

Proof. Suppose, for a contradiction, that the claim does not hold for D. Let $\widetilde{\pi}_D$ be the permutation of tasks in $\widetilde{\mathcal{J}}$ such that for each pair of tasks $\widetilde{J}_i, \widetilde{J}_{i'} \in \widetilde{\mathcal{J}}$ it holds $\widetilde{\pi}_D^{-1}(\widetilde{J}_i) < \widetilde{\pi}_D^{-1}(\widetilde{J}_{i'})$ if and only if $\pi_D^{-1}(\widetilde{J}_i) < \pi_D^{-1}(\widetilde{J}_{i'})$. In other words, to obtain $\widetilde{\pi}_D$ we simply restrict π_D to tasks in $\widetilde{\mathcal{J}}$. Then, find the smallest index $i \in \{1, \ldots, m\}$ such that $\widetilde{\pi}_D(i) \neq \widetilde{J}_i$. Clearly, $\widetilde{\pi}_D(i) = \widetilde{J}_k$, $k > i$, and

$$\widetilde{C}_k \geq \widetilde{p}_k(\widetilde{s}_k) + \sum_{i'=1,\ldots,i-1} \widetilde{p}_{i'}(\widetilde{s}_{i'}) = iB^3.$$

Since $\widetilde{J}_i$ is executed in D later than $\widetilde{J}_k$,

$$\widetilde{C}_i \geq \widetilde{C}_k + \widetilde{p}_i(\widetilde{s}_i) \geq (i+1)B^3 > iB^3 + \frac{i(i+1)}{2}B = \widetilde{d}_i,$$

because $B^3 > m^2B \geq i(i+1)/2B$ for $i \leq m < B$. This gives the desired contradiction. $\square$

Given a schedule D for $\mathcal{J} \cup \widetilde{\mathcal{J}}$, define $\widetilde{I}_i = [\widetilde{l}_i, \widetilde{r}_i) = [\widetilde{C}_i, \widetilde{s}_{i+1})$ for $i = 1, \ldots, m-1$ and let $\widetilde{I}_m = [\widetilde{C}_m, L)$. By Lemma 7, this definition is valid and all the tasks in $\mathcal{J}$ have to be scheduled within $\bigcup_{i=1,\ldots,m} \widetilde{I}_i$.

Lemma 8. *If D is a schedule for $\mathcal{J} \cup \widetilde{\mathcal{J}}$, then $\widetilde{I}_i \subseteq I_i$ for each $i = 1, \ldots, m$.*

Proof. By the definition, $\widetilde{C}_i = \widetilde{l}_i$, and, by Lemma 7,

$$\widetilde{C}_i \geq \sum_{1 \leq i' \leq i} \widetilde{p}_{i'}(\widetilde{s}_{i'}) = iB^3 \geq (i-1)B^3 + \frac{(i-1)i}{2}B = l_i, \quad i = 1, \ldots, m. \tag{7}$$

For the right endpoint of $\widetilde{I}_i$, $i \in \{1, \ldots, m-1\}$, it holds

$$\widetilde{r}_i = \widetilde{s}_{i+1} \leq \widetilde{d}_{i+1} - \widetilde{p}_{i+1}(\widetilde{s}_{i+1}) = l_{i+1} + B^3 - B^3 = l_{i+1} = r_i. \tag{8}$$

Since $\widetilde{r}_m = L = r_m$, by (7) and (8), $\widetilde{l}_i \geq l_i$ and $\widetilde{r}_i \leq r_i$, which implies $\widetilde{I} = [\widetilde{l}_i, \widetilde{r}_i) \subseteq I_i$ for each $i = 1, \ldots .m$. $\square$

Lemma 9. *If D is a schedule for $\mathcal{J} \cup \widetilde{\mathcal{J}}$, then $|\widetilde{I}_i| = iB$ for each $i = 1, \ldots, m$.*

Proof. We assume, for a contradiction, that the thesis does not hold for D. We define a new set of tasks corresponding to $\mathcal{J}$, namely $J_j^1, \ldots, J_j^{a_j}$ are a_j tasks corresponding to $J_j \in \mathcal{J}$. The set of all tasks J_j^l is denoted by $\mathcal{J}'$. Note that $|\mathcal{J}'| = mB$. For each $J_j^l \in \mathcal{J}'$ we define the deadline to be the same as for J_j, while the execution time is $p_j^l(t) = i$ for $t \in I_i$, $l = 1, \ldots, a_j$. Consider a schedule D_0 for $\mathcal{J}' \cup \widetilde{\mathcal{J}}$ obtained from D in such a way that each task $J_j \in \mathcal{J}$ is replaces by the sequence $J_j^1, \ldots, J_j^{a_j}$. A task J_j executes within $\widetilde{I}_i$ for some $i \in \{1, \ldots, m\}$, and, by Lemma 8, $\widetilde{I}_i \subseteq I_i$, which means that its execution time is ia_i. Also by Lemma 8, the sum of execution times of $J_j^1, \ldots, J_j^{a_j}$ is $\sum_{l=1,\ldots,a_i} i = ia_i$. This in particular means that $\mathrm{ms}(D) = \mathrm{ms}(D_0)$ and all tasks in $\widetilde{J}$ are executed in the same time intervals in both schedules.

Now we will perform a sequence of modifications to the schedule D_0, obtaining a sequence of schedules $D_1, D_2, \ldots, D_q$ for the set of tasks $\mathcal{J}' \cup \widetilde{\mathcal{J}}$. We describe the first modification leading us from D_0 to D_1 and the migration from D_p to D_{p+1} is analogous for each p, $0 < p < q$. In the remaining part of this proof we use symbols $\widetilde{s}_i(D_p)$, $\widetilde{C}_i(D_p)$ to distinguish the parameters of tasks which depend on a schedule D_p, $p \geq 0$. Consequently we write $\widetilde{I}_i(D_p)$ since the endpoints depend on the execution time of $\widetilde{J}_i$'s. For a task $J_j^l \in \mathcal{J}'$ its start and completion time in a schedule D_p is $s_j^l(D_p)$ and $C_j^l(D_p)$, respectively. Find in D_0 the interval $\widetilde{I}_i(D_0)$ such that $|\widetilde{I}_i(D_0)| \neq iB$ and $|\widetilde{I}_{i'}(D_0)| = i'B$ for each $i' = 1, \ldots, i-1$. Such $\widetilde{I}_i(D_0)$ does exist since we assumed for a contradiction that the thesis does not hold. Moreover, $i < m$.

If $|\widetilde{I}_i(D_0)| > iB$, then $\widetilde{J}_{i+1}$ starts at

$$\begin{aligned}\widetilde{s}_{i+1}(D_0) &= iB^3 + |\widetilde{I}_i(D_0)| + \sum_{i'=1,\ldots,i-1} i'B \\ &= iB^3 + |\widetilde{I}_i(D_0)| - iB + \sum_{i'=1,\ldots,i} i'B = l_{i+1} + |\widetilde{I}_i(D_0)| - iB.\end{aligned}$$

This, however, means that $\widetilde{J}_{i+1}$ does not finish before its deadline, $\widetilde{C}_{i+1}(D_0) = \widetilde{s}_{i+1}(D_0) + B^3 > l_{i+1} + B^3 = \widetilde{d}_{i+1}$. So, $|\widetilde{I}_i(D_0)| < iB$. Therefore, $|\widetilde{I}_i(D_0)| < iB$.

To obtain D_1, let initially $D_1 = D_0$ and we apply the following modifications to D_1. Find in D_1 the task $J_j^l \in \mathcal{J}'$ which executes first in the interval $[\widetilde{r}_i(D_1), L]$. Then, let $s_j^l(D_1) = \widetilde{r}_i(D_1)$. Note that only tasks in $\widetilde{\mathcal{J}}$ are executed in the interval $[\widetilde{r}_i(D_0), s_j^l(D_0)]$. To make the schedule D_1 feasible, shift i units to the right all tasks in $\widetilde{\mathcal{J}}$ which are executed in $[\widetilde{r}_i(D_0), s_j^l(D_0)]$. In the new schedule D_1 no two tasks overlap, because by the definition and by Lemma 8 the execution time of J_j^l in D_0 is at least $(i+1)B$, while its execution time in D_1 is iB. To prove that the schedule is feasible after shifting the tasks it is enough to argue that the task $\widetilde{J}_{i+1}$ succeeding J_j^l in D_1 finishes before its deadline. To prove it observe that for each $i' < i$ it holds $|\widetilde{I}_{i'}| = i'B$, which implies that

$$\widetilde{s}_i(D_1) = \sum_{i'=1,\ldots,i-1} (B^3 + i'B) = (i-1)B^3 + \frac{(i-1)i}{2}B = l_i,$$

which means that $\widetilde{C}_i(D_1) = \widetilde{s}_i(D_1) + B^3 = l_i + B^3$, and

$$\widetilde{C}_{i+1}(D_1) = \widetilde{C}_i(D_1) + |\widetilde{I}_i(D_1)| + B^3 = l_i + |\widetilde{I}_i(D_1)| + 2B^3 \le l_{i+1} + B^3 = \widetilde{d}_{i+1},$$

because $|\widetilde{I}_i(D_1)| \le iB$. If more tasks in $\widetilde{\mathcal{J}}$ have been shifted while computing D_1, then they also finish before their deadlines, because they are executed consecutively, following $\widetilde{J}_{i+1}$. Note that there is now an idle time in D_1, because $s_l^j(D_1) \in I_i$ and $s_l^j(D_0) > r_i$, which by Lemma 8 means that the execution time of J_j^l is strictly bigger in D_0 than in D_1. (Assume that the difference in execution times is $x > 0$.) So, each task which succeeds J_j^l in D_0 is executed in D_1 at least x time units earlier, because the execution time of each task does not increase when the execution starts earlier. Consequently, $\mathrm{ms}(D_0) > \mathrm{ms}(D_1)$. Similarly, we obtain that $\mathrm{ms}(D_i) > \mathrm{ms}(D_{i+1})$ for each $i = 1, \ldots, q-1$.

The schedule D_q has the property that each interval $\widetilde{I}_i(D_q)$, $i = 1, \ldots, m$, is of length iB. So, the makespan of D_q is $\mathrm{ms}(D_q) = mB^3 + \sum_{i=1,\ldots,m} iB = mB^3 + \frac{m(m+1)}{2}B = L$. Thus,

$$\mathrm{ms}(D) = \mathrm{ms}(D_0) > \mathrm{ms}(D_1) > \cdots > \mathrm{ms}(D_q) = L.$$

In particular we obtain that the makespan of D exceeds L, while the deadline of each task in $\mathcal{J} \cup \widetilde{\mathcal{J}}$ is at most L — a contradiction. □

Lemma 10. *There exists a schedule for $\mathcal{J} \cup \widetilde{\mathcal{J}}$ if and only if there exists a 3-partition for A and B.*

Proof. Let $A_1, \ldots, A_m$ be a 3-partition of A. Let $\mathcal{J}_i = \{J_j \in \mathcal{J} : a_j \in A_i\}$. Create a schedule D in such a way that

$$\pi_D = (\widetilde{J}_1, \mathcal{J}_1, \ldots, \widetilde{J}_i, \mathcal{J}_i, \ldots, \widetilde{J}_m, \mathcal{J}_m)$$

(the tasks in each $\mathcal{J}_j$ are executed in any order). We use induction on i to prove that the tasks in $\{\widetilde{J}_i\} \cup \mathcal{J}_i$ are executed in time interval I_i. The case when $i = 1$ and $i > 1$ are analogous, so assume that all the tasks in $\bigcup_{1 \le i' \le i}(\{\widetilde{J}_{i'}\} \cup \mathcal{J}_{i'})$ are executed within $I_1 \cup \cdots \cup I_i = [0, r_i]$ for some $1 \le i < m$. For $\{\widetilde{J}_{i+1}\} \cup \mathcal{J}_{i+1}$ we obtain that $\widetilde{J}_{i+1}$ is scheduled first and its execution time is B^3. Then, the tasks in $\mathcal{J}_{i+1}$ follow in any order. Moreover, for each $t \in I_{i+1}$ we obtain $\sum_{J_j \in \mathcal{J}_{i+1}} p_j(t) = (i+1) \sum_{a_j \in A_{i+1}} a_j = (i+1)B$, because A_{i+1} is a part of the solution to the 3-partition problem. Thus, by (6), the tasks in $\{\widetilde{J}_{i+1}\} \cup \mathcal{J}_{i+1}$ can be executed within $[r_i, r_i + B^3 + (i+1)B] = [l_{i+1}, l_{i+1} + B^3 + (i+1)B] = I_i$.

Let D be a schedule for $\mathcal{J} \cup \widetilde{\mathcal{J}}$. By Lemma 9, $|\widetilde{I}_i| = iB$ for each $i = 1, \ldots, m$. Let $i \in \{1, \ldots, m\}$. Since, by the definition of $\widetilde{I}_i$'s the tasks executed within $\widetilde{I}_i$ belong to $\mathcal{J}$ and, by Lemma 8, executing J_j in $\widetilde{I}_i$ takes ia_j time. Thus, for the tasks $\mathcal{J}_i \subseteq \mathcal{J}$ executed within $\widetilde{I}_i$ the total running time is iB, i.e. $\sum_{J_j \in \mathcal{J}_i} ia_j = iB$. So, $A_i = \{a_j : J_j \in \mathcal{J}_i\}$, $i = 1, \ldots, m$, is a solution to the 3-partition problem. □

Theorem 2. *Given a set of tasks $\mathcal{J}$ with integer deadlines and integer non-decreasing (in time) execution times, the problem of deciding if there exists a feasible schedule for $\mathcal{J}$ is strongly* NP-*complete.* □

5.2. *Reducing* TDS *to* CS

In this subsection we prove NP-hardness of CS problem. We start by reducing TDS to CSFH, then we conclude that CS is NP-complete as well.

The instance of TDS consists of a set of tasks $\mathcal{J}$, where each task $J_j \in \mathcal{J}$ has its integer deadline d_j and a non-decreasing function $p_j : \{0, \ldots, d_j - 1\} \to \mathbb{N}_+$ describing the execution time. As argued in the previous section, the integer valued functions p_j imply that in each schedule s_j and C_j are integers, $J_j \in \mathcal{J}$, which also justifies that we may consider the values of p_j only at integer points. For each $J_j \in \mathcal{J}$ let f_j be the latest possible integer starting point for J_j, i.e. $f_j = \max\{t \in \mathbb{N} : t + p_j(t) \le d_j\}$. The integer L is selected to be an upper bound for the length of each feasible schedule,

$$L = \max\{d_j : J_j \in \mathcal{J}\}. \tag{9}$$

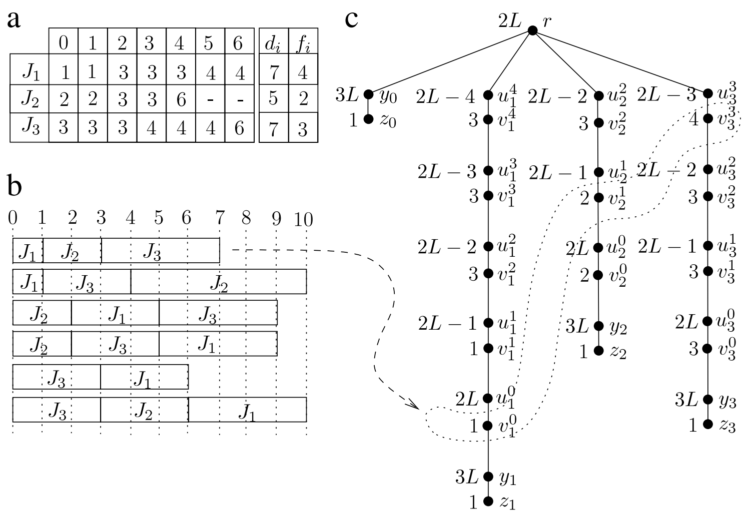

Fig. 3. (a) J_1, J_2 and J_3 with execution times p_j, deadlines d_j and the latest possible starting times f_j, $j = 1, 2, 3$; (b) all possible schedules for the three tasks; (c) the corresponding weighted tree T.

Given $\mathcal{J}$, we construct a node-weighted tree $T = (V(T), E(T), w)$ rooted at r (the weight of each edge is 1). For each $J_j \in \mathcal{J}$ define a path P_j with

$$V(P_j) = \{u_j^i, v_j^i : i = 0, \ldots, f_j\},$$
$$E(P_j) = \{u_j^i v_j^i : i = 0, \ldots, f_j\} \cup \{v_j^{i+1} u_j^i : i = 0, \ldots, f_j - 1\}.$$

The tree T, in addition to the vertices in $\bigcup_{J_j \in \mathcal{J}} V(P_j)$, contains the vertices r and $y_j, z_j, j = 0, \ldots, |\mathcal{J}|$. The root r is adjacent to y_0 and to the endpoint $u_j^{f_j}$ of each path P_j, $j = 1, \ldots, |\mathcal{J}|$. The other endpoint of P_j, namely the vertex v_j^0, is adjacent to y_j for each $j = 1, \ldots, |\mathcal{J}|$. Finally, for each $0 = 1, \ldots, |\mathcal{J}|$ the vertex y_j is the parent of z_j.

The weight function $w : V(G) \to \mathbb{N}_+$ is as follows

$$w(r) = 2L, \tag{10}$$

$$w(y_j) = 3L, \qquad w(z_j) = 1 \quad j = 0, \ldots, |\mathcal{J}|, \tag{11}$$

$$w(u_j^i) = 2L - i \quad \text{and} \quad w(v_j^i) = p_j(i) \tag{12}$$

for each $j = 1, \ldots, |\mathcal{J}|$, $i = 0, \ldots, f_j$. Finally, let $k = 4L$ be the number of available searchers.

Fig. 3 gives an example of this reduction for $\mathcal{J}$ of size 3. Fig. 3(a) shows a function p_j that defines the execution times, the deadline d_j and the value of f_j for each $j = 1, 2, 3$. (Note that f_j is the latest possible starting time point for $J_j \in \mathcal{J}$.) All possible schedules for $\mathcal{J}$ are depicted in Fig. 3(b), where the schedule for the execution order (J_3, J_1, J_2) is not complete, because the starting time of J_2 would exceed its deadline. Note that only the first schedule is valid, for in any other one at least one task does not finish at or prior to its deadline. The weighted tree T that corresponds to the TDS problem instance is shown in Fig. 3(c).

Informally speaking, the idea used in this reduction is as follows. In the TDS problem if we start a task 'early', then we benefit from the fact that this task executes faster, i.e. its execution time interval is of smaller length. In the CS problem we find a similar 'structure': if we guard the root and we start working on a particular path P_j early, then can in general 'reach' in P_j a border of smaller weight, and we benefit by having more free searchers for clearing other paths. Now we continue with a formal analysis.

Note that for each u_j^i and $v_j^{i'}$, $0 \leq i, i' \leq f_j$, it holds

$$w(u_j^i) > L \geq w(v_j^{i'}), \tag{13}$$

because $f_j < L$ for each $j = 1, \ldots, |\mathcal{J}|$. Other simple facts that will be useful in the following are

$$w(u_j^0) > w(u_j^1) > \cdots > w(u_j^{f_j}), \quad j = 1, \ldots, |\mathcal{J}|, \tag{14}$$

$$w(v_j^{f_j}) \geq w(v_j^{f_j - 1}) \geq \cdots \geq w(v_j^0), \quad j = 1, \ldots, |\mathcal{J}|. \tag{15}$$

We start by describing a search strategy $\mathcal{S}$ for T_r, assuming that a schedule D for $\mathcal{J}$ is given (recall that π_D is the order of executing the jobs in D):

Step 1: Initially $4L$ searchers occupy r.

Step 2: For each $i = 1, \ldots, |\mathcal{J}|$ do the following: let $J_j = \pi_D(i)$; clear the path $P_j(D) \subseteq P_j$ consisting of the vertices $u_j^{f_j}$, $v_j^{f_j}, \ldots, u_j^{s_j}, v_j^{s_j}$. (After this step, by (12), $w(v_j^{s_j}) = p_j(s_j)$ searchers occupy $v_j^{s_j}$ to guard it.)

Step 3: Clear the edges ry_0 and y_0z_0.

Step 4: For each $J_j \in \mathcal{J}$ clear the path $u_j^{s_j-1}, v_j^{s_j-1}, \ldots, u_j^0, v_j^0, y_j, z_j$ (after this step the subtree rooted at $u_j^{f_j}$ is clear).

In the example in Fig. 3 the only valid schedule executes the tasks according to the order (J_1, J_2, J_3). We use it in Step 2 above, which gives us that in $\mathcal{S}$ we 'reach' in the branches P_1, P_2 and P_3 the vertices v_1^0, v_2^1 and v_3^3, respectively, (distinguished in Fig. 3(c)) of total weight at most L. The following lemma states that $\mathcal{S}$ is always valid and uses at most $4L$ searchers.

Lemma 11. *$\mathcal{S}$ is a connected search strategy for T. Moreover, $\mathtt{s}(\mathcal{S}) \leq k = 4L$.*

Proof. It is easy to see that after each step the subtree that is clear is connected. Now we prove that the number of searchers used is at most $k = 4L$. Initially $2L$ searchers guard r. We prove by induction on $i = 1, \ldots, |\mathcal{J}|$ that k searchers suffice to clear the path $P_j(D)$ in Step 2, where $J_j = \pi_D(i)$, and the number of searchers used in $\mathcal{S}$ for guarding when the vertex $v_j^{s_j}$ becomes guarded is

$$x_i = 2L + \sum_{j':\pi_D^{-1}(J_{j'}) \leq i} p_{j'}(s_{j'}). \tag{16}$$

The cases when $i = 1$ and $i > 1$ are analogous ($x_0 = 2L$), so we prove it for i, assuming that it is true for $i-1$, $1 \leq i \leq |\mathcal{J}|$.

Let $P_j(D) \subseteq P_j$ be the ith path cleared, i.e. $J_j = \pi_D(i)$. By (13) and (14),

$$w(u_j^{s_j}) = \max\{w(v) : v \in V(P_j(D))\}.$$

So, by (16), $w(u_j^{s_j}) + x_{i-1}$ searchers are needed to clear $P_j(D)$. We obtain

$$w(u_j^{s_j}) + x_{i-1} = (2L - s_j) + 2L + \sum_{j':\pi_D^{-1}(J_{j'}) < i} p_{j'}(s_{j'}) = 4L,$$

because, by the definition of a schedule for time-dependent tasks, the execution of a task J_j starts immediately after the execution of the preceding task, which can be stated as

$$s_j = \sum_{j':\pi_D^{-1}(J_{j'}) < i} p_{j'}(s_{j'}).$$

Thus, $x_i = x_{i-1} + w(v_j^{s_j}) = x_{i-1} + p_j(s_j)$ and (16) follows. This proves that $4L$ searchers are used during search moves defined in Steps 1 and 2 above. When the execution of all search operations constructed in Step 2 is completed, $2L$ searchers are used for guarding r, while for guarding the vertices $v_j^{s_j}$, $j = 1, \ldots, |\mathcal{J}|$ we need

$$\sum_{j=1,\ldots,|\mathcal{J}|} w(v_j^{s_j}) = \sum_{j=1,\ldots,|\mathcal{J}|} p_j(s_j) \leq L \tag{17}$$

searchers. The last inequality follows from Eq. (9) and from the fact that in a valid schedule D each task is completed within the interval $[0, L]$. Thus, we can use $3L$ searchers to clear ry_0, y_0z_0 and then the remaining subpaths $u_j^{s_j-1}, v_j^{s_j-1}, \ldots, u_j^0, v_j^0$, y_j, z_j. $\square$

Corollary 3. *If there exists a valid schedule for $\mathcal{J}$, then there exists a connected $4L$-search strategy for the weighted tree T rooted at r.* $\square$

Now we prove the reverse implication, i.e. that the existence of a search strategy for T_r gives a valid schedule for $\mathcal{J}$. We start with a technical lemma.

Lemma 12. *In each connected $4L$-search strategy $\mathcal{S}$ for T_r, ry_0 is the edge that is cleared last among the edges in E_r.*

Proof. Let $\mathcal{S}[i]$ be the move of clearing ry_0. If at least one edge in $E_r \setminus \{ry_0\}$ is contaminated during clearing ry_0, the vertex r has to be guarded while clearing ry_0. That would imply $|\mathcal{S}[i]| = w(r) + w(y_0) = 5L$ – a contradiction. $\square$

Lemma 13. *If there exists a connected $4L$-search strategy $\mathcal{S}$ for the weighted tree T_r with homebase r, then there exists a valid schedule for $\mathcal{J}$.*

Proof. Given $\mathcal{S}$, define a schedule D, where $\pi_D(i) = J_j$ if and only if $ru_j^{f_j}$ is the ith cleared edge among the edges in $E_r \setminus \{r, y_0\}$. In other words, the order of clearing the edges in E_r determines the order of task execution in D.

Let $\mathcal{S}[c_j]$ be clearing of $ru_j^{f_j}$, $j = 1, \ldots, |\mathcal{J}|$, and let $\mathcal{S}[c_{|\mathcal{J}|+1}]$ be the move of clearing ry_0. By Lemma 12, ry_0 is cleared last among the edges in E_v.

In order to prove that D is valid we show by induction on $j = 1, \ldots, |\mathcal{J}|$ the two following facts.

Fact 1: $s_j \le f_j$ for each $j = 1, \ldots, |\mathcal{J}|$.
Fact 2: The move $\mathcal{S}[c_{j+1} - 1]$ clears the edge $u_j^{s_j} v_j^{s_j}$, $j = 1, \ldots, |\mathcal{J}|$.

To simplify the presentation we proceed with the assumption that $\pi_D(j) = J_j$ for each $j = 1, \ldots, |\mathcal{J}|$, i.e. the order of clearing the edges in E_r is $ru_1^{f_1}, \ldots, ru_{|\mathcal{J}|}^{f_{|\mathcal{J}|}}, ry_0$.

Let $j = 1$. Clearly J_1 starts at $s_1 = 0$ in D, which implies Fact 1 for $j = 1$. $2L$ searchers guard r while clearing a subpath of P_1. Since $w(v) \le 2L$ for each $v \in V(P_1)$, the searchers clear the whole path P_1, ending at $v_1^0 = v_1^{s_1}$. Then, y_1 cannot be cleared, because $w(y_1) = 3L$, and $w(r) = 2L$ searchers occupy r to guard it. So, the next move is $\mathcal{S}[c_2]$ which proves Fact 2 for $j = 1$.

Assume now that Fact 1 and Fact 2 hold for some $j - 1 \in \{1, \ldots, |\mathcal{J}| - 1\}$.

For D it holds $s_j = \sum_{i=1,\ldots,j-1} p_i(s_i)$. By the induction hypothesis (Fact 2), the number of searchers used to guard the vertices in subtrees rooted at $u_1^{f_1}, \ldots, u_{j-1}^{f_{j-1}}$ is $\sum_{i=1,\ldots,j-1} w(v_i^{s_i})$. By (12), $w(v_i^{s_i}) = p_i(s_i)$, which implies that $2L + w(v) + \sum_{i=1,\ldots,j-1} p_i(s_i) = 2L + w(v) + s_j$ is the number of searchers used while reaching $v \in V(P_j)$. In particular, the number of searchers used to reach $u_j^{f_j}$ is $2L + 2L - f_j + s_j$. Since $\mathcal{S}$ uses $4L$ searchers, $s_j \le f_j$ which proves Fact 1.

The move $\mathcal{S}[c_j]$ clears $ru_j^{f_j}$ and then the searchers clear partially the subtree rooted at $u_j^{f_j}$, ending by clearing a vertex v_j^x, $0 \le x \le f_j$ and then the move $\mathcal{S}[c_{j+1}]$ follows. (y_j cannot be cleared when r is guarded, because $w(y_j) = 3L$). Moreover, the search does not stop at a vertex u_j^i, because by (13) it is possible to continue by clearing v_j^i for each $i = 0, \ldots, f_j$.)

If $x < s_j$, then, in particular, the vertex $u_j^{s_j-1}$ has been cleared, while $2L + s_j$ searchers are used to guard r and $v_i^{s_i}$, $i = 1, \ldots, j - 1$. By (12), $w(u_j^{s_j-1}) = 2L - s_j + 1$. So, the total number of searchers used while clearing $v_j^{s_j} u_j^{s_j-1}$ is $2L + s_j + 2L - s_j + 1 > 4L$ – a contradiction.

If $x > s_j$, then we can clear $v_j^x u_j^{x-1}$, because as before $2L + s_j$ searchers are used for guarding and $w(u_j^{x-1}) = 2L - (x - 1)$ additional searchers clear $v_j^x u_j^{x-1}$, which means that the number of searchers in use is $4L + s_j - x + 1 \le 4L$. Then, by (13), we can clear $u_j^{x-1} v_j^{x-1}$. By Lemma 2, w.l.o.g. $\mathcal{S}$ clears $v_j^x u_j^{x-1}$ and $u_j^{x-1} v_j^{x-1}$.

By Fact 1, $s_j \le f_j$, for each task $J_j \in \mathcal{J}$, which means that $C_j \le f_j + p_j(s_j) \le d_j$, which proves that D is valid. □

Due to the monotonicity [2,3], the CSFH problem is clearly in NP, and the reduction is polynomial in $n = |\mathcal{J}|$, because L is, by Theorem 2, polynomially bounded in n, which gives us the theorem.

Theorem 3. *Given a weighted tree T rooted at r and an integer $k \ge 0$, deciding whether $\mathtt{cs}(T, r) \le k$ is* NP*-complete.* □

Let $T_r = (V(T), E(T), w)$ and k be an input to the CSFH problem. Define $T_r^2 = (V(T), E(T), 2w)$ to be the tree with the same vertex and edge sets as T_r, while the weight of each vertex v of T_r^2 is two times bigger than the weight of v in T_r. There exists a connected k-search strategy for T_r if and only if there exists a connected $(2k)$-search strategy for T_r^2. Take three copies of T_r^2, and a vertex r' (the weight of r' is 1), and let the roots of the trees T_r^2 be the children of r'. The new tree is denoted by $T'_{r'}$. We obtain that $\mathtt{cs}(T', r') = 2k + 1$. Moreover, if $\mathcal{S}'$ is a connected $(2k + 1)$-search strategy for T' then regardless of the homebase in $\mathcal{S}'$, the strategy is forced to clear one of the subtrees T_r^2 in T'_r by starting at r and using $2k$ searchers. This leads to the following

Corollary 4. *The problem of connected searching of weighted trees is strongly* NP*-complete.* □

6. Conclusions

The main contribution of this work is establishing the complexity status of connected searching of weighted trees. It also follows from the NP-hardness proof and from the algorithm presented that the maximum degree of a given tree is a factor that makes a particular instance computationally tractable or intractable. A natural direction for further research is to consider other parameters that can draw a tight line between computationally easy and hard instances. Investigating weighted trees is also of importance as it may lead to interesting results for a more general class of chordal graphs. One of the interesting open problems is the existence of 'good' approximations for finding connected search strategies for weighted trees.

Acknowledgement

The author was partially supported by the Foundation for Polish Science (FNP) and by MNiSW grant N N206 379337 (2009–2011).

References

[1] B. Alspach, Searching and sweeping graphs: a brief survey, Le Matematiche (Catania) 59 (2004) 5–37.
[2] L. Barrière, P. Flocchini, F.V. Fomin, P. Fraigniaud, N. Nisse, N. Santoro, D.M. Thilikos, Connected graph searching, Research Report RR-7363, INRIA, 2010.
[3] L. Barrière, P. Flocchini, P. Fraigniaud, N. Santoro, Capture of an intruder by mobile agents, in: SPAA'02: Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, NY, USA, 2002, pp. 200–209.
[4] J. Błażewicz, K. Ecker, E. Pesch, G. Schmidt, J. Węglarz, Scheduling Computer and Manufacturing Processes, Springer-Verlag, New York, NY, USA, 1996.
[5] R.B. Borie, C.A. Tovey, S. Koenig, Algorithms and complexity results for pursuit-evasion problems, in: IJCAI '09: Proceedings of the 21st International Joint Conference on Artificial Intelligence, 2009, pp. 59–66.
[6] P. Brucker, Scheduling Algorithms, Springer Publishing Company, Incorporated, 2007.
[7] T.C.E. Cheng, Q. Ding, Scheduling start time dependent tasks with deadlines and identical initial processing times on a single machine, Comput. Oper. Res. 30 (1) (2003) 51–62.
[8] T.C.E. Cheng, Q. Ding, B.M.T. Lin, A concise survey of scheduling with time-dependent processing times, European J. Oper. Res. 152 (1) (2004) 1–13.
[9] D. Dereniowski, Connected searching of weighted trees, in: MFCS'10: Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science, 2010, pp. 330–341.
[10] D. Dereniowski, From pathwidth to connected pathwidth, in: STACS'11: Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science, 2011, pp. 416–427.
[11] R. Diestel, Graph Theory, Springer-Verlag, Heidelberg, 2010.
[12] P. Flocchini, M.J. Huang, F.L. Luccio, Contiguous search in the hypercube for capturing an intruder, in: IPDPS'05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, IEEE Computer Society, Washington, DC, USA, 2005, p. 62.
[13] P. Flocchini, M.J. Huang, F.L. Luccio, Decontaminating chordal rings and tori using mobile agents, Internat J. Found. Comput. Sci. 18 (3) (2007) 547–563.
[14] P. Flocchini, M.J. Huang, F.L. Luccio, Decontamination of hypercubes by mobile agents, Network 52 (3) (2008) 167–178.
[15] Fedor V. Fomin, Pierre Fraigniaud, Dimitrios M. Thilikos, The price of connectedness in expansions, Technical Report, UPC Barcelona, 2004.
[16] F.V. Fomin, D.M. Thilikos, An annotated bibliography on guaranteed graph searching, Theoret. Comput. Sci. 399 (3) (2008) 236–245.
[17] F.V. Fomin, D.M. Thilikos, I. Todinca, Connected graph searching in outerplanar graphs, Electron. Notes Discrete Math. 22 (2005) 213–216.
[18] P. Fraigniaud, N. Nisse, Connected treewidth and connected graph searching, in: Proc. of the 7th Latin American Symposium on Theoretical Informatics, LATIN'06, in: LNCS, vol. 3887, Valdivia, Chile, 2006, pp. 479–490.
[19] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman & Co., New York, NY, USA, 1979.
[20] G. Hollinger, S. Singh, J. Djugash, A. Kehagias, Efficient multi-robot search for a moving target, I. J. Robotic Res. 28 (2) (2009) 201–219.
[21] G. Hollinger, S. Singh, A. Kehagias, Improving the efficiency of clearing with multi-agent teams, I. J. Robotic Res. 29 (8) (2010) 1088–1105.
[22] F. Huc, Conception de réseaux dynamiques tolérants aux pannes,.Ph.D. Thesis, Nice Sophia Antipolis, Projet MASCOTTE (CNRS - UNSA) INRIA, 2008.
[23] A. Kehagias, G. Hollinger, S. Singh, A graph search algorithm for indoor pursuit/evasion, Math. Comput. Modelling 50 (9–10) (2009) 1305–1317.
[24] L.M. Kirousis, C.H. Papadimitriou, Searching and pebbling, Theoret. Comput. Sci. 47 (2) (1986) 205–218.
[25] A. Kolling, S. Carpin, On weighted edge-searching, Technical Report 2009–01, School of Engineering, University of California, Merced, 2009.
[26] A. Kolling, S. Carpin, Pursuit-evasion on trees by robot teams, IEEE Trans. Robotics 26 (1) (2010) 32–47.
[27] W. Kubiak, S. van de Velde, Scheduling deteriorating jobs to minimize makespan, Naval Res. Logis. 45 (5) (1998) 511–523.
[28] R. Mihai, I. Todinca, Pathwidth is NP-hard for weighted trees, in: FAW'09: Proceedings of the Third International Workshop on Frontiers in Algorithmics, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 181–195.
[29] N. Nisse, Connected graph searching in chordal graphs, Discrete Appl. Math. 157 (12) (2008) 2603–2610.
[30] R.J. Nowakowski, N. Zeh, Boundary-optimal triangulation flooding, Internat. J. Comput. Geom. Appl. 16 (2–3) (2006) 271–290.
[31] T.D. Parsons, Pursuit-evasion in a graph, in: Theory and Applications of Graphs, in: Lecture Notes in Mathematics, vol. 642, Springer-Verlag, 1978, pp. 426–441.
[32] P. Shareghi, N. Imani, H. Sarbazi-Azad, Capturing an intruder in the pyramid, in: CSR, 2006, pp. 580–590.
[33] R.J. Wilson, Introduction to Graph Theory, Addison Wesley Longman, 1996.
[34] Ö. Yaşar, D. Dyer, D.A. Pike, M. Kondratieva, Edge searching weighted graphs, Discrete Appl. Math. 157 (8) (2009) 1913–1923.
[35] B. Yang, D. Dyer, B. Alspach, Sweeping graphs with large clique number, Discrete Math. 309 (18) (2009) 5770–5780.