ELSEVIER

Theoretical Computer Science 240 (2000) 429–446

Theoretical Computer Science

www.elsevier.com/locate/tcs

Edge and node searching problems on trees ☆

Sheng-Lung Peng[a,*,1], Chin-Wen Ho[b], Tsan-sheng Hsu[c,1], Ming-Tat Ko[c,1], Chuan Yi Tang[a]

[a] *Department of Computer Science, National Tsing Hua University, Hsinchu 30043, Taiwan*
[b] *Department of Computer Science and Information Engineering, National Central University, Taiwan*
[c] *Institute of Information Science, Academia Sinica, Taiwan*

Abstract

In this paper, we consider the edge searching and node searching problems on trees. Given a tree, we show a transformation from an optimal node-search strategy to an optimal edge-search strategy. Using our transformation, we simplify a previous linear-time algorithm for determining the edge-search number of a tree, and improve the running time of a previous algorithm for constructing an optimal edge-search strategy of an n-vertex tree from $O(n\log n)$ to $O(n)$. We also improve the running time of a previous algorithm for constructing an optimal min-cut linear layout of an n-vertex tree with the maximum degree 3 from $O(n\log n)$ to $O(n)$.

Keywords: Algorithm; Edge searching; Node searching; Pathwidth; Tree

1. Introduction

The graph searching problem was first proposed by Parsons [27, 28] and independently proposed by Petrov [31]. A graph represents a system of tunnels. Initially, all the edges of the graph are contaminated by a gas. We wish to obtain a state of the graph in which all the edges are simultaneously cleared by a sequence of moves using the least number of searchers. The graph searching problem is not only interesting theoretically, but also have applications on several combinatorial problems [2, 9, 15, 20, 17, 22, 25, 33]

In this paper, we consider the edge searching problem and the node searching problem on trees. In *node searching* [16], the allowable moves are (1) placing a searcher on a vertex and (2) removing a searcher from a vertex. A contaminated edge is *cleared*

☆ An extended abstract of this paper appears in COCOON'97, Lecture Note in Computer Science, Vol. 1276, 284–293, 1997.

* Corresponding author.

[1] Part of this research was supported by NSC Grants 86-2213-E-001-012 and NSC 87-2213-E-001-022.

PII: S0304-3975(99)00241-8

if both its two endpoints simultaneously contain searchers. In *edge searching* [27], besides the allowable moves in the node searching, one more move, (3) moving a searcher along an edge, is allowed. In edge searching, a contaminated edge is *cleared* by moving a searcher along this edge. A cleared edge may be recontaminated if there is a path from a contaminated edge to the cleared edge without any searcher on its vertices (or edges). A vertex is *guarded* if it contains a searcher.

A *node-search strategy* is a sequence of moves allowed by node searching rules that clears the initially contaminated graph. The *node searching problem* is the problem to find a node-search strategy to clear the initially contaminated graph using as few searchers as possible. The number of searchers needed to solve the node searching problem on a graph G is called the *node-search number* of G and we denote it as $ns(G)$. We define similarly for the *edge searching problem*, an *edge-search strategy*, and the *edge-search number* $es(G)$ of G. A search strategy is called *optimal* if it uses the minimum number of searchers. It has been shown in [7, 16] (respectively, [7, 21]) that there always exists an optimal node-search (respectively, edge-search) strategy for a graph that does not recontaminate any edge. Kirousis and Papadimitriou [16] proved that for any graph G, $ns(G) - 1 \leqslant es(G) \leqslant ns(G) + 1$. In the rest of paper, we only consider the node- and edge-search strategies which do not recontaminate any edge.

The node searching problem is equivalent to the gate matrix layout problem and interval graph augmentation problem [25]. The problem of finding the node-search number is equivalent to the pathwidth problem [25, 33], the interval thickness problem [15], the narrowness problem [20], and the vertex separation problem [16, 17]. From the equivalent of the above problems, the node searching problem is NP-complete on planar graphs with vertex degree at most 3 [26], starlike graphs (a proper subclass of chordal graphs) [14], bipartite graphs [18], cobipartite graphs (i.e., complement of bipartite graphs) [1], and bipartite distance-hereditary graphs (a proper subclass of the chordal bipartite graphs and distance-hereditary graphs) [19]. For some special classes of graphs, it can be solved in polynomial time, as e.g., trees [10, 25, 34], cographs [6], permutation graphs [3], trapezoid graphs [4], split graphs [14, 18], partial k-trees [5], and k-starlike graphs for a fixed k [14, 30].

The edge searching problem is equivalent to the min-cut linear arrangement problem for any graph with the maximum degree 3 [23]. The edge searching problem is NP-complete on general graphs [24], planar graphs with the maximum vertex degree 3 [26] and starlike graphs [30]. However, it can be solved in polynomial time on complete graphs [13], trees [24], interval graphs, split graphs, and k-starlike graphs for a fixed $k \geqslant 2$ [30].

Though the above two searching problems appear to be similar, the time complexities to solve them are different. There are linear time algorithms on a tree to find both its node-search number and an optimal node-search strategy [34, 35] (also mentioned in [25, Theorem 4.7]). However, the previous best algorithm [24] takes $O(n \log n)$ time to find an optimal edge-search strategy on a tree of n vertices, while its edge-search number can be found in linear time [24]. In this paper, we improve the time complexity

of finding an optimal edge-search strategy on a tree by establishing a relationship between the two searching problems on this tree.

We first extend the concept of an *avenue* of a tree in edge searching as used by Megiddo et al. [24] to an *avenue system*. We show that in node searching, a similar avenue system can be defined. Based on properties of the above two avenue systems, we discover that the two search numbers are equal on trees that have at least four vertices with no degree-2 vertex, and whose every internal vertex is adjacent to at least one leaf, so-called a sprout tree (will be defined in Section 3). We further show that an optimal node-search strategy for a sprout tree can be transformed into an optimal edge-search strategy using the same number of searchers in linear time. For any tree T, if it is not a sprout tree, then we can transform it to a sprout tree T'. We will prove that if T is not a path, then T and T' have the same edge-search number. Our above transformation takes time linear in the size of the input tree. Note that the best previous result for constructing an optimal edge-search strategy for a tree needs $O(n \log n)$ time [24]. Besides the above algorithmic achievement, the relationship between two searching problems we discovered may be of interest by itself.

Recently, we were informed that independently Golovach [11, 12] obtained similar results. In [12], Golovach mentioned that if a graph G has no vertices of degree 2 and is different from the complete graph with two vertices then $ns(G) \leqslant es(G)$. Unfortunately, no detail is given. We were also told that Golovach [11] has the following results. If graph G' is obtained from the graph G by adding of any number of degree-1 vertices adjacent to vertices of G having degrees more than 2, then $es(G) = es(G')$. In the same thesis, Golovach also shows that if there exists an optimal node-search strategy of G such that in which one searcher is placed on a vertex v, $\deg(v) \geqslant 3$, by some move and is removed from v immediately by the next move, and there are less than $ns(G)$ searchers on the graph after the first move, then $ns(G) \geqslant es(G)$.

The remaining of this paper are organized as follows. In Section 2, we define the avenue systems on trees for edge and node searching problems. Our main results about the relationship between the node searching and edge searching on trees are presented in Section 3. The linear time algorithm for constructing an optimal edge-search strategy for a tree and the min-cut linear layout problem on trees with the maximum degree 3 are presented in Section 4. Finally, we give conclusions in Section 5.

2. Avenue system

Let T be an unrooted and connected tree. Let $V(T)$ and $E(T)$ denote the vertex and edge sets of T, respectively. A sequence of vertices $[v_1, v_2, \ldots, v_r]$ is a *path* if $(v_i, v_{i+1}) \in E(T)$, $1 \leqslant i \leqslant r-1$. A vertex in T with degree 1 is called a *leaf* and a nonleaf vertex is called an *internal vertex*. For any vertex $t \in V(T)$, a connected component of $T \backslash \{t\}$ is called a *branch* of T at t. Let v be adjacent to t in T. The branch of T at t containing v is denoted as T_{tv}. Let T_{tv}^+ denote the subtree such that $V(T_{tv}^+) = V(T_{tv}) \cup \{t\}$

and $E(T_{tv}^+) = E(T_{tv}) \cup \{(t,v)\}$. T_{tv}^+ is called an *e-branch* at t. Note that the branch T_{tv} (or e-branch T_{tv}^+) is uniquely determined by the vertex t and its neighbor v.

A full version containing trivial details as well as examples that are omitted in this section can be found in [29].

2.1. Edge searching

Lemma 1 (Parsons [27]). *If G' is a subgraph of G then $es(G') \leqslant es(G)$.*

Lemma 2 (Parsons [27]). *For any tree T and an integer $k \geqslant 1$, $es(T) \geqslant k+1$ if and only if there exists a vertex $t \in V(T)$ with at least three e-branches T_{tu}^+, T_{tv}^+, and T_{tw}^+ such that $es(T_{tu}^+) \geqslant k$, $es(T_{tv}^+) \geqslant k$, and $es(T_{tw}^+) \geqslant k$.*

From Lemma 2, Megiddo et al. [24] proposed the concept of avenue of a tree for the edge searching. For any tree T, let $s = es(T)$. A path $[v_1, v_2, \ldots, v_r]$ of two or more vertices is an *e-avenue* for T if the following conditions hold.

(1) Exactly one e-branch of v_1 (respectively, v_r) has edge-search number s and this e-branch contains v_2 (respectively, v_{r-1}).
(2) For every j, $2 \leqslant j \leqslant r-1$, the edge-search numbers of exactly two e-branches of v_j are s and in these two e-branches, one contains v_{j-1} and the other contains v_{j+1}.

Given an e-avenue $[v_1, v_2, \ldots, v_r]$, an e-branch at v_i, $1 \leqslant i \leqslant r$, is called a *nonavenue e-branch* if it contains no other vertex in the e-avenue but v_i. We call a vertex v in a tree T an *e-hub* of T if the edge-search number of any e-branch at v is less than $es(T)$.

Lemma 3 (Megiddo et al. [24]). *A tree has either an e-hub or a unique e-avenue.*

Note that more than one vertex in a tree can be chosen as an e-hub. A tree T is *minimal with respect to edge searching* if the deletion of any vertex results in a forest T' whose $es(T')$ equals to $es(T) - 1$. We define similarly for T being minimal with respect to node searching. In a tree T that is minimal with respect to edge searching and $es(T) \geqslant 2$, every internal vertex is an e-hub [24]. The following lemma can be easily proved by the definitions of e-avenue and e-hub.

Lemma 4. *For any tree T of $es(T) \geqslant 2$, any leaf of T cannot be an e-hub or a vertex of the e-avenue.*

For convenience, in the rest of this paper, an e-hub is regarded as an e-avenue consisting of a single vertex. Note that if $es(T) = 1$, then T is a path.

Let T be a tree. We define an e-*avenue system* $\mathscr{A}^e(T)$ and the set of nonavenue e-branches $\mathscr{F}(\mathscr{A}^e(T))$ as follows.

(1) If T is a path $[u_1, \ldots, u_k]$, then $\mathscr{A}^e(T) = \{[u_1, \ldots, u_k]\}$ and $\mathscr{F}(\mathscr{A}^e(T)) = \{T\}$.
(2) If T is not a path, then let $[v_1, v_2, \ldots, v_r]$ be its e-avenue and let $\mathscr{T}(T) = \{B \mid B$ is a nonavenue e-branch at v_i, $1 \leqslant i \leqslant r\}$. Then $\mathscr{A}^e(T) = \{[v_1, v_2, \ldots, v_r]\} \cup (\bigcup_{T' \in \mathscr{T}(T)} \mathscr{A}^e(T'))$ and $\mathscr{F}(\mathscr{A}^e(T)) = \{T\} \cup (\bigcup_{T' \in \mathscr{T}(T)} \mathscr{F}(\mathscr{A}^e(T')))$.

With respect to $\mathscr{A}^e(T)$, *e-labels* of vertices in T are defined as follows. First, for each tree T' in $\mathscr{F}(\mathscr{A}^e(T))$ with $es(T') \geqslant 2$, the e-label of any vertex in the e-avenue of T' in $\mathscr{A}^e(T)$ is $es(T')$. Second, for each tree T' in $\mathscr{F}(\mathscr{A}^e(T))$ with $es(T') = 1$, the e-label of any vertex in T' is 1 if this vertex is not labeled above. Note that there is no conflict in labeling a vertex, i.e., a vertex cannot have two different e-labels. If v belongs to the e-avenue of a tree $B \in \mathscr{F}(\mathscr{A}^e(T))$ with $es(B) \geqslant 2$, then v is labeled with $es(B)$ and v becomes a leaf in the e-branches at v in B. By Lemma 4, v does not belong to the e-avenue of any other tree $B' \in \mathscr{F}(\mathscr{A}^e(T))$ with $es(B') \geqslant 2$. By the labeling rules, v is not relabeled for the rest of the labeling process. If the e-label of a vertex v is 1, then v cannot have any e-label whose value is not 1.

By definition, each vertex v whose e-label is at least 2 is in an e-avenue of a subtree of T in $\mathscr{F}(\mathscr{A}^e(T))$. We denote this tree by T^v. Let i be the e-label of v in $\mathscr{A}^e(T)$. Then $es(T^v) = i$ and T^v is a nonavenue e-branch at u of T^u for some u whose e-label is at least $i + 1$. Note that if the e-label of v is $es(T)$, then $T^v = T$. If $es(T^v) \geqslant 2$, then the nonavenue e-branches at v in the subtree T^v are referred in the following as nonavenue e-branches at v without specifying the subtree.

Since more than one vertex in a tree can be chosen as an e-hub, a tree may have many distinct e-avenue systems. In addition, by Lemma 5 and our labeling method, we know that for any e-avenue system of tree T, the labels of the leaves of T are 1. We have the following lemma.

Lemma 5. *For any tree T with no vertex of degree 2 and $|V(T)| \geqslant 4$, no internal vertex is labeled with 1 in any e-avenue system of T.*

Proof. Let $\mathscr{A}^e(T)$ be an e-avenue system of T. By Lemma 4, all the e-labels of the leaves in T are 1. Suppose v is an internal vertex whose e-label is 1. By the definition of e-label, there exists a vertex u such that the e-label of u is at least 2, v belongs to a nonavenue e-branch T' at u and $es(T') = 1$. Since T' is a path and v is not labeled then, the degree of v in T is either 1 or 2. It contradicts to the fact that T has no vertex of degree 2 and v is an internal vertex. □

Based on an $\mathscr{A}^e(T)$, we can construct an optimal edge-search strategy of T. Assume that $es(T) = k$. Let $[v_1, v_2, \ldots, v_r] \in \mathscr{A}^e(T)$ be the e-avenue of T. Our edge-search strategy is as follows. We first place one searcher on v_1 then we recursively clear all the nonavenue e-branches at v_1 using at most $k - 1$ searchers. Note that the edge-search number of any nonavenue e-branch at v_1 is less than k. After all the nonavenue e-branches at v_1 are cleared, we again have $k - 1$ free searchers. We then move the searcher at v_1 to v_2 along the edge (v_1, v_2). By using a process similar to the one we used to clear the nonavenue e-branches at v_1, we can clear each nonavenue e-branch of v_i, $2 \leqslant i \leqslant r$, one after one using at most $k - 1$ searchers. After all the nonavenue e-branches at v_r are cleared, T is cleared. Hence we have the following lemma.

Lemma 6. *An e-avenue system $\mathscr{A}^e(T)$ of T corresponds to an optimal edge-search strategy of T.*

2.2. Node searching

Similar to Lemma 1, we have the following lemma.

Lemma 7. *If G' is a subgraph of G, then $ns(G') \leqslant ns(G)$.*

Let T be a tree. If T contains any edge, then $ns(T) \geqslant 2$. For convenience, we define $ns(T) = 1$ if T contains only one vertex. Thus, $ns(T) \geqslant 2$ if and only if there exists a vertex $t \in V(T)$ with at least one branch. The necessary and sufficient conditions for $ns(T) \geqslant k+1$, $k \geqslant 2$, were provided by Scheffler [34]. The following lemma is due to Scheffler [34].

Lemma 8 (Scheffler [34]). *For any tree T, $ns(T) \geqslant k+1$ for $k \geqslant 2$ if and only if there exists a vertex $t \in V(T)$ with at least three branches T_{tu}, T_{tv}, and T_{tw} such that $ns(T_{tu}) \geqslant k$, $ns(T_{tv}) \geqslant k$, and $ns(T_{tw}) \geqslant k$. For any tree T, $ns(T) \geqslant 2$ if and only if there exists a vertex $t \in V(T)$ with at least one branch.*

By Lemma 8, we can define similarly the avenue of node searching as follows. A path $[v_1, v_2, \ldots, v_r]$ of two or more vertices is an *n-avenue* for a tree T with $ns(T) = s \geqslant 2$, if the following conditions hold.

(1) Exactly one branch of v_1 (respectively, v_r) has node-search number s and this branch contains v_2 (respectively, v_{r-1}).
(2) For every j, $2 \leqslant j \leqslant r-1$, the node-search numbers of exactly two branches of v_j are s and in these two branches, one contains v_{j-1} and the other contains v_{j+1}.

Given an n-avenue $[v_1, v_2, \ldots, v_r]$, a branch at v_i, $1 \leqslant i \leqslant r$, is called a *nonavenue branch* if it contains no other vertex in the n-avenue. We call a vertex v in a tree T an *n-hub* of T if all the branches at v have node-search number less than $ns(T)$.

Lemma 9. *A tree has either an n-hub or a unique n-avenue.*

Proof. Our proof is similar to the proof of Lemma 3 in [24] by observing that the e-branches in the proof of Lemma 3 are replaced by the branches. □

Similar to e-hubs, more than one vertex in a tree can be chosen as an n-hub. In a minimal tree with respect to node searching, every vertex is an n-hub. That is, a leaf of a tree can be an n-hub.

Lemma 10. *Let T be a tree with $|V(T)| > 2$. If T has an n-hub, then there always exists an internal vertex of T which is an n-hub.*

Proof. Consider the case that v is a leaf and v is an n-hub of T. Let u be the neighbor of v. Since v is a leaf, v has only one branch $T' = T \setminus \{v\}$. Note that $ns(T') = ns(T) - 1$. All the branches at u except the one consisting of the single vertex v are subtrees of T'. By Lemma 7, the node-search numbers of the above branches are no greater than

$ns(T')$. The node-search number of the vertex v is 1. Thus, u is also an n-hub of T. Since $|V(T)|>2$, u is an internal vertex of T. □

In the rest of this paper, an n-hub is also regarded as an n-avenue consisting of a single vertex. We define below an *n-avenue system* which is similar to the e-avenue system. Let T be a tree. We define an *n-avenue system* $\mathscr{A}^{\mathrm{n}}(T)$ and the set of nonavenue branches $\mathscr{F}(\mathscr{A}^{\mathrm{n}}(T))$ as follows.

(1) If T consists of one single vertex v, then $\mathscr{A}^{\mathrm{n}}(T)=\{[v]\}$ and $\mathscr{F}(\mathscr{A}^{\mathrm{n}}(T))=\{T\}$.

(2) If T consists of more than one vertex, then let $[v_1,v_2,\ldots,v_r]$ be its n-avenue and let $\mathscr{T}(T)=\{B\mid B \text{ is a nonavenue branch at } v_i,\ 1\leqslant i\leqslant r\}$. Then $\mathscr{A}^{\mathrm{n}}(T)=\{[v_1,v_2,\ldots,v_r]\}\cup(\bigcup_{T'\in\mathscr{T}(T)}\mathscr{A}^{\mathrm{n}}(T'))$ and $\mathscr{F}(\mathscr{A}^{\mathrm{n}}(T))=\{T\}\cup(\bigcup_{T'\in\mathscr{T}(T)}\mathscr{F}(\mathscr{A}^{\mathrm{n}}(T')))$.

With respect to $\mathscr{A}^{\mathrm{n}}(T)$, n-*labels* of vertices in T are defined as follows. For each tree T' in $\mathscr{F}(\mathscr{A}^{\mathrm{n}}(T))$, the n-label in $\mathscr{A}^{\mathrm{n}}(T)$ of any vertex in the n-avenue of T' is $ns(T')$. Since more than one vertex in a tree can be chosen as an n-hub, a tree may have many distinct n-avenue systems.

During the assignment of n-labels, for each branch T' in $\mathscr{F}(\mathscr{A}^{\mathrm{n}}(T))$, if T' has an n-hub then, by Lemma 10, we can always choose an internal vertex as its n-hub. If $ns(T')=2$, $|V(T')|=2$, and a vertex $u\in V(T')$ is a leaf in T, then we label the other vertex, which is an internal vertex of T, in T' with 2. By doing so, we have the following lemma.

Lemma 11. *Let T be a tree with $|V(T)|>2$. Then there exists an n-avenue system of T such that the n-labels of all the leaves of T are* 1.

By the definition of n-label, each vertex v in an n-avenue for a subtree of T in $\mathscr{F}(\mathscr{A}^{\mathrm{n}}(T))$, we denote this tree by T^v. Let i be the n-label of v in $\mathscr{A}^{\mathrm{n}}(T)$. Then $ns(T^v)=i$ and T^v is a nonavenue branch at u of T^u for some u whose n-label is at least $i+1$. Note that if the n-label of v is $ns(T)$, then $T^v=T$. If $ns(T^v)\geqslant 2$, then nonavenue branches at v in the subtree T^v are referred in the following as nonavenue branches at v without specifying the subtree.

In general, besides the leaves of T, internal vertices can be labeled with 1 in an n-avenue system.

Lemma 12. *Let T be a tree with at least one internal vertex and whose every internal vertex is adjacent to at least one leaf. Then there exists an n-avenue system of T such that no internal vertex of T is labeled with* 1.

Proof. Consider an n-avenue system $\mathscr{A}^{\mathrm{n}}$ of T satisfying Lemma 11. By our definition of n-labels, the neighbor of a vertex with n-label 1 cannot be labeled with 1 in $\mathscr{A}^{\mathrm{n}}$. Hence, there is no internal vertex of T whose n-label is 1 in $\mathscr{A}^{\mathrm{n}}$. □

Similar to edge searching, we can construct an optimal node-search strategy of T based on an $\mathscr{A}^{\mathrm{n}}(T)$.

Lemma 13. *An n-avenue system $\mathscr{A}^{\mathrm{n}}(T)$ of T corresponds to an optimal node-search strategy of T.*

3. Relation between node and edge searching on trees

In this section, we show a relationship between node- and edge-search strategy on trees. We first define the *reduction operation* on degree-2 vertices in a tree T. Let v be a vertex of degree 2 which is adjacent to vertices u and w. Let T' be the tree obtained from T by deleting v and its incident edges, and then joining u and w by a new edge. We say that T' is obtained from T by applying a reduction operation on v. The *reduction* of T is the tree obtained from T by applying all possible reduction operations. That is, there is no degree-2 vertex in the reduction of T. A tree of at least four vertices is called a *reduction tree* if it is the reduction of some trees. The following lemma is implied by the results mentioned in [16, 32].

Lemma 14. *Let T' be the reduction of a tree T. Then $es(T)=es(T')$.*

We next define the *sprout operation* on internal vertices of a tree. For an internal vertex v that is not adjacent to any leaf, the sprout operation adds a new leaf to vertex v. The *sprout* of T is the tree obtained from the reduction of T by applying all possible sprout operations. A tree is called a *sprout tree* if it is a sprout of a reduction tree. Let T' be the sprout tree of a reduction tree T. Let T^+_{vu} be any e-branch at v in T and $es(T^+_{vu})\geqslant 2$. Then the e-branch at v in T' which contains u is the sprout of T^+_{vu}.

Lemma 15. *Let T' be the sprout of a reduction tree T. Then $es(T)=es(T')$.*

Proof. Since T is a subtree of T', by Lemma 1, $es(T)\leqslant es(T')$. By definition of sprout tree, all the vertices in $V(T')\backslash V(T)$ are leaves. By Lemma 4 and our labeling method, any e-avenue system of tree T', the labels of the leaves of T' are 1. Let $W=\{[u,v]\mid v\in V(T')\backslash V(T)$ and $(u,v)\in E(T')\}$. Note that for each $[u,v]\in W$, u is an internal vertex and v is a leaf added by a sprout operation. It is not hard to see that $\mathscr{A}^{\mathrm{e}}(T)\cup W$ is an e-avenue system of T'. By using $\mathscr{A}^{\mathrm{e}}(T)\cup W$, it can be proved that $es(T)\geqslant es(T')$. Hence $es(T)=es(T')$. □

Remark. The detail for showing $es(T)\geqslant es(T')$ can be found in [29]. We were informed that Lemma 15 is implied by results independently shown in [11] (in Russian).

A *caterpillar* is a tree consisting of a simple path P (called the *body* or *backbone*) with an arbitrary number of simple paths attached by coalescing an endpoint of the added path with a vertex in P. The attached paths are called *hairs*. A caterpillar is called a *k-caterpillar* if all of its hairs have length at most k.

Lemma 16. *For any reduction tree T, $es(T)=2$ if and only if $ns(T)=2$.*

Proof. Assume that $es(T)=2$. Let $[v_1,v_2,\ldots,v_r]$ be an e-avenue of T. The edge-search numbers of the nonavenue e-branches at v_i, $1\leqslant i\leqslant r$, are 1, i.e., the nonavenue e-branches at v_i are paths. Since T is a reduction tree, the length of each nonavenue e-branch is 1. It implies that T is a 1-caterpillar. On the other hand, a 1-caterpillar with no degree-2 vertex is a reduction tree with the edge-search number 2.

With a similar argument, we can show that a reduction tree of node-search number 2 is a 1-caterpillar with no degree-2 vertex and *vice versa*. The lemma thus follows. □

Lemma 17. *Let T be a tree and let v be a vertex whose n-label is at least 2 in an n-avenue system of T. Let T' be a tree obtained by attaching a new leaf u to v. Then $ns(T)=ns(T')$.*

Proof. We prove this lemma by induction on $ns(T)$. In the case of $ns(T)=2$, by the definition of v, v is a vertex in the n-avenue of T. Thus T_{vu} is a branch at v with $V(T_{vu})=\{u\}$. It is not difficult to see that $ns(T')=2=ns(T)$.

We assume for all trees T with $2\leqslant ns(T)\leqslant k-1$, $ns(T')\leqslant ns(T)$. Now we consider a tree T with $ns(T)=k$ and its n-avenue $A^{\mathrm{n}}(T)=[v_1,\ldots,v_r]$. In T', we also call the branches at v_i which do not contain any v_j, $j\neq i$ and $1\leqslant j\leqslant r$, the *nonavenue branches* without ambiguity. We provide the following node-search strategy for T' according to $A^{\mathrm{n}}(T)$. First, we place a searcher on v_1. Then, we clear one by one the nonavenue branches at v_1 by optimal node-search strategies with no recontamination. During the clearing of a branch T_{v_1w} at v_1, the edge (v_1,w) is cleared once w is guarded by a searcher and is not recontaminated. After all the nonavenue branches at v_1 are cleared, we place a searcher on v_2. Then v_1 is cleared and the searcher at v_1 is removed. We continue the above clearing process on $v_2,\ldots,v_r$ sequentially until all the nonavenue branches at v_r are cleared. Then T' is cleared.

Let $\mathscr{T}(T)=\{B\mid B$ is a nonavenue branch at v_i, $1\leqslant i\leqslant r$, in $T\}$ and $\mathscr{T}(T')=\{B\mid B$ is a nonavenue branch at v_i, $1\leqslant i\leqslant r$, in $T'\}$. We compute the number of searchers used in the following two cases.

(1) $v=v_i$ for some i, $1\leqslant i\leqslant r$. Let T_i be the tree containing only one vertex u. Then $\mathscr{T}(T')=\mathscr{T}(T)\cup\{T_i\}$. Since for all $T^*\in\mathscr{T}(T')$, $ns(T^*)\leqslant k-1$, our node-search strategy uses at most k searchers.

(2) $v\neq v_i$ for all i, $1\leqslant i\leqslant r$. Let T^* be the nonavenue branch at v_i for some i, $1\leqslant i\leqslant r$ which contains u. By the induction hypothesis, $ns(T^*)\leqslant k-1$. All the other nonavenue branches in $\mathscr{T}(T')$ are also in $\mathscr{T}(T)$, which are of node-search number no greater than $k-1$. Thus our node-search strategy uses at most k searchers.

By the above discussion, $ns(T')\leqslant ns(T)$. Since T is a subtree of T', by Lemma 7, $ns(T)\leqslant ns(T')$. Thus $ns(T)=ns(T')$. □

Lemma 18. *For any sprout tree T, $es(T)\leqslant ns(T)$.*

Proof. We prove this lemma by induction on the number $ns(T)$. First, by Lemma 16, if $ns(T)=2$, then $es(T)=2$. Next, we assume that for every sprout tree T with

$2 \leqslant ns(T) \leqslant k-1$, $es(T) \leqslant ns(T)$. Now let T be a sprout tree with $ns(T)=k$. By Lemma 12, we have an n-avenue system $\mathscr{A}^{\mathrm{n}}(T)$ in which no internal vertex of T has n-label 1. Let $[v_1, v_2, \ldots, v_r] \in \mathscr{A}^{\mathrm{n}}(T)$ be an n-avenue of T. For each nonavenue branch T_{v_iu} at v_i, $1 \leqslant i \leqslant r$, if $ns(T_{v_iu}) \geqslant 2$, then T_{v_iu} may contain at most one vertex of degree 2. If it has a degree-2 vertex, then this vertex must be u which is an internal vertex of T_{v_iu}. Furthermore, u is adjacent to a leaf. Since the n-label of each leaf is 1, the n-label of u is at least 2. By Lemma 17, if $ns(T_{v_iu}) \geqslant 2$, then $ns(T^+_{v_iu}) = ns(T_{v_iu}) \leqslant k-1$ and $T^+_{v_iu}$ is a sprout tree. If $ns(T_{v_iu})=1$, then $V(T_{v_iu})=\{u\}$.

In the following, we will clear T by edge searching rules based on the n-avenue system $\mathscr{A}^{\mathrm{n}}(T)$ using $ns(T)$ searchers. For each nonavenue branch T_{v_iu} of $ns(T_{v_iu}) \geqslant 2$ at v_i, $1 \leqslant i \leqslant r$, $ns(T^+_{v_iu}) = ns(T_{v_iu}) \leqslant k-1$ and $T^+_{v_iu}$ is a sprout tree. By the induction hypothesis, if $ns(T_{v_iu}) \geqslant 2$, then $es(T^+_{v_iu}) \leqslant ns(T^+_{v_iu}) \leqslant k-1$. Thus we can clear T in the context of edge searching by first placing one searcher on v_1. Then, we clear $T^+_{v_1u}$ of $ns(T_{v_1u}) \geqslant 2$ using at most $k-1$ searchers by edge searching rules. To clear $T^+_{v_1u}$ of $ns(T_{v_1u})=1$, in edge searching, we only place one searcher on u and move it to v_1 along the edge (u, v_1). After the nonavenue e-branches $T^+_{v_1u}$ at v_1 are cleared, we have $k-1$ free searchers and we move the searcher at v_1 to v_2 along the edge (v_1, v_2). After the edge (v_1, v_2) is cleared, v_1 is cleared and v_2 is guarded. We then continue the above clearing process on $v_2, \ldots, v_r$ sequentially until all the nonavenue e-branches at v_r are cleared. That is, $es(T) \leqslant ns(T)$. □

Remark. We were informed that Lemma 18 is implied by results independently shown in [11] (in Russian).

Lemma 19. *For any reduction tree T, $ns(T) \leqslant es(T)$.*

Proof. The proof is similar to the proof of Lemma 18. The detail can be found in [29] and we omit it. □

Remark. Lemma 19 is implied by results independently mentioned in [12].

Theorem 20. *For any sprout tree T, $es(T) = ns(T)$.*

Proof. Since a sprout tree is also a reduction tree, by Lemmas 18 and 19, this theorem holds. □

Theorem 21. *The edge-search number of a tree can be determined in linear time.*

Proof. We design an algorithm to find the edge-search number of any tree T as follows. If T is a path, then $es(T)=1$. If T is not a path, then we first construct its reduction tree T'. Next, we construct the sprout tree T'' of T'. By using any linear-time algorithm [10, 25, 34] to compute $ns(T'')$. By Lemmas 14, 15, and Theorem 20, $es(T) = ns(T'')$. □

Remark. Though there is a linear-time algorithm to determine the edge-search number of a tree [24], by using our results, we can also obtain a linear-time algorithm to determine the edge-search number of a tree.

4. Construction of an optimal edge-search strategy

As in Section 2, we can construct an optimal edge-search strategy of a tree from its e-avenue system. If the pointers from every e-avenue A to the e-avenues of nonavenue e-branches at vertices of A are provided, then it takes linear time in the construction of corresponding edge-search strategy. However, for the time being, we do not know how to build an e-avenue system with the pointers in linear time. In this section, we present a linear-time algorithm to construct an optimal edge-search strategy of a tree T from an optimal node-search strategy of T, which does not use avenue systems.

According to Kirousis and Papadimitriou [15], an optimal node-search strategy of G can be represented by a sequence of vertex sets $\mathscr{Y}=(Y_1,\ldots,Y_r)$, where $Y_i\subseteq V(G)$ is a set of vertices guarded by searchers at step i for $1\leqslant i\leqslant r$. Recall that we only consider the node-search strategies which do not recontaminate any edge. An edge (u,v) is *cleared at step* i if $\{u,v\}\subseteq Y_i$ and $\{u,v\}\not\subseteq Y_j$ for all $j<i$. An edge (u,v) is *clear at step* j if $u,v\in Y_i$ for some $i\leqslant j$. A vertex u is *cleared at step* i if it is the first step that all the incident edges of u are clear.

For simplicity of presentation, in the following we assume $Y_0=Y_{r+1}=\emptyset$. The node-search strategy $\mathscr{Y}$ clears T as follows. At the beginning of step i, $1\leqslant i\leqslant r$, all the vertices in $Y_i\cap Y_{i-1}$ are guarded. In this step, we guard all the vertices in $Y_i\setminus Y_{i-1}$, i.e., the whole Y_i is guarded. Then, since there is no recontamination in $\mathscr{Y}$, the vertices in $Y_i\setminus Y_{i+1}$ are cleared and we can remove all the searchers on the vertices of $Y_i\setminus Y_{i+1}$. The node-search number of $\mathscr{Y}$ is $\max_i|Y_i|$. Note that $(Y_1,\ldots,Y_r)$ is also called a *path-decomposition* of the graph G [25]. Furthermore, there exists an optimal node-search strategy $\mathscr{Y}$ satisfying the following assumptions.

(1) For any vertex $u\in Y_i$, $1\leqslant i\leqslant r$, at least one incident edge of u is clear at step i.
(2) If u is cleared at step i, then $u\notin Y_j$ for all $j>i$.
(3) $Y_i\not\supseteq Y_{i-1}$ and $Y_i\not\subseteq Y_{i-1}$ for $2\leqslant i\leqslant r$.

In the following, we consider $\mathscr{Y}$ satisfying the above three assumptions. Note that in $\mathscr{Y}$, any leaf occurs in exactly one of its steps by Assumptions (1) and (2). Also, if v is guarded at step i and is cleared at step j, then $v\in Y_t$ for $i\leqslant t\leqslant j$.

For each vertex $u\in Y_i\setminus Y_{i+1}$, we say that step i is the *clearing step* of u in $\mathscr{Y}$. According to the clearing steps of vertices, all the vertices of T can be sorted into a sequence $\mathscr{C}=(v_1,v_2,\ldots,v_n)$ such that the clearing step of v_i is no later than the clearing step of v_j if $i<j$. We call $\mathscr{C}$ a *clearing sequence* of $\mathscr{Y}$. Note that all the vertices in $Y_i\setminus Y_{i+1}$ have the same clearing step i. For vertices with the same clearing step, without loss of generality, we assume in the following that the orders of leaves (if they exist) are smaller than that of the others in $\mathscr{C}$.

The clearing sequence $\mathscr{C}$ plays an important role in constructing our optimal edge-search strategy. In the following, we first show that a clearing sequence $\mathscr{C}$ which corresponds to an optimal node-search strategy satisfying the three assumptions can be constructed in linear time. Then, according to $\mathscr{C}$, we design a linear-time algorithm to construct an optimal edge-search strategy of a sprout tree. Finally, we construct an optimal edge-search strategy of a tree from the optimal edge-search strategy of its sprout tree.

Let F be a set of intervals. Let G be the interval graph defined by the intersection relation of F. If G contains T as a subgraph, then F is an *interval model* of T. For each vertex $u \in V(T)$, let u be guarded at step a_u and be cleared after step b_u in a node-search strategy $\mathscr{Y} = (Y_1, \ldots, Y_r)$, i.e., $u \in Y_t$ for $a_u \leqslant t \leqslant b_u$. Let $I_u = [a_u, b_u]$ for all $u \in V(T)$. Since for all $(u,v) \in E(T)$, $I_u \cap I_v \neq \emptyset$, $\{I_u \mid u \in V(T)\}$ is an interval model of T. Note that $Y_i = \{u \in V(T) \mid I_u = [a_u, b_u] \text{ and } a_u \leqslant i \leqslant b_u\}$. An interval model of T is *optimal* if the maximum clique size of its corresponding interval graph is the smallest among all interval models of T. Conversely, an optimal interval model $F = \{I_u = [a_u, b_u] \mid u \in V(T)\}$ of T corresponds to an optimal node-search strategy of T in which for all $u \in V(T)$, a searcher is placed on u at step a_u and is removed after step b_u [15].

Scheffler mentioned that an optimal interval model F of T can be constructed in linear time [35]. In general, the node-search strategy corresponds to F may not fulfill Assumptions (1)–(3). In order to obtain an optimal interval model F^* whose corresponding optimal node-search strategy satisfies the three assumptions, we make the following modification of F.

Let $N(u) = \{v \mid v \in V(T) \text{ and } (u,v) \in E(T)\}$ and $N[u] = \{u\} \cup N(u)$. Let $F = \{I_u = [a_u, b_u] \mid u \in V(T)\}$. We first modify F into $F' = \{I'_u = [a'_u, b'_u] \mid u \in V(T)\}$ by setting $a'_u = \max\{a_u, \min\{a_v \mid v \in N(u)\}\}$ and $b'_u = \max\{a_v \mid v \in N[u]\}$ for all $u \in V(T)$. It can be verified that F' is an interval model of T by showing that $a'_u \leqslant b'_u$ for all $u \in V(T)$ and $I'_u \cap I'_v \neq \emptyset$ for all $(u,v) \in E(T)$. Let $\mathscr{Y}'$ denote the node-search strategy corresponding to F'. By the setting of a'_u, at least one neighbor of u is guarded at time a'_u in $\mathscr{Y}'$ for all $u \in V(T)$. Thus, $\mathscr{Y}'$ satisfies Assumption (1). By the setting of b'_u, b'_u is the first time step at which u is cleared for all $u \in V(T)$. Thus, $\mathscr{Y}'$ satisfies Assumption (2). In the above modification, for each vertex, we only need to check its neighbors in T and overall it takes linear time.

The interval model F^* whose corresponding node-search strategy $\mathscr{Y}^*$ satisfies the three assumptions is obtained by modifying F' as follows. We first sort the endpoints of all the intervals in F' in nondecreasing order, in which for endpoints with the same value, left endpoints precede right endpoints. After this, we partition the sorted sequence into a consecutive sequence of segments where each segment contains a consecutive sequence of left endpoints followed by a consecutive sequence of right endpoints. Assume there are totally r segments. We number these segments from 1 to r in increasing order. For all vertices u, if a'_u (respectively, b'_u) is in the ith segment, let $a^*_u = i$ (respectively, $b^*_u = i$). Let $F^* = \{I^*_u = [a^*_u, b^*_u] \mid u \in V(T)\}$. Note that F^* preserves the intersection relations of intervals in F'. Let $Y^*_i = \{u \in V(T) \mid I^*_u = [a^*_u, b^*_u] \in F^* \text{ and } a^*_u \leqslant i \leqslant b^*_u\}$

for $1 \leqslant i \leqslant r$ and $\mathscr{Y}^* = (Y_1^*, \ldots, Y_r^*)$. It can be verified that $\mathscr{Y}^*$ satisfies Assumptions (1) and (2). Since there is at least one right (respectively, left) endpoint in the ith (respectively, $(i+1)$th) segment, $Y_i \not\subseteq Y_{i+1}$ (respectively, $Y_{i+1} \not\subseteq Y_i$). That is, $\mathscr{Y}^*$ satisfies Assumption (3). A clearing sequence $\mathscr{C}$ corresponding to $\mathscr{Y}^*$ can be obtained by sorting vertices according to the right endpoints of their corresponding intervals in F^* in nondecreasing order, in which for vertices with the same value of right endpoints, leaves precede internal vertices. By using a linear-time integer sorting algorithm [8], the above sorting processes can be done in linear time. Hence $\mathscr{C}$ can be obtained from F in linear time.

Let T be a sprout tree and F be an optimal interval model of T obtained as in the above. Let $\mathscr{Y} = (Y_1, \ldots, Y_r)$ be an optimal node-search strategy corresponding to F and let $\mathscr{C}$ be a clearing sequence corresponding to $\mathscr{Y}$. Next, we construct an optimal edge-search strategy $\mathscr{S}$ from $\mathscr{C}$ in linear time. In $\mathscr{S}$, the vertices are cleared in the same order as $\mathscr{C}$. The moves of $\mathscr{S}$ are as the following algorithm.

Algorithm OES(T: sprout tree, $\mathscr{C} = (v_1, \ldots, v_n)$);
for $i = 1$ **to** n **do**
  if v_i is not guarded **then** place a searcher on v_i;
  if v_i has only one uncleared incident edge (v_i, u) **then**
    move the searcher on v_i to u along the edge (v_i, u)
  else begin
    for all uncleared edges (v_i, u), where u is guarded, use a free
      searcher to clear (v_i, u);
    for all uncleared edges (v_i, u), where u is unguarded, **do begin**
      place a searcher on v_i;
      move this searcher to u along the edge (v_i, u)
    end for;
    remove the searcher on v_i
  end if
end for
end OES;

Let $\mathscr{S}$ be the edge-search strategy constructed by Algorithm OES. In each iteration of OES, a vertex is cleared. Let phase j of $\mathscr{S}$ be the sequence of moves obtained from a sequence of iterations in OES for clearing the vertices in $Y_j \backslash Y_{j+1}$. The idea of our algorithm is that in phase j of $\mathscr{S}$, it clears all the vertices in $Y_j \backslash Y_{j+1}$ using at most $|Y_j|$ searchers. Note that in edge searching, an edge is cleared by letting a searcher go through it (instead of by just guarding both endpoints as in node searching). Therefore, though an edge is guarded by searchers at both of its endpoints, we need another searcher to clear this edge. In each phase of $\mathscr{S}$, it should be guaranteed that no extra searcher is needed to clear the vertices.

Let $S_j = \{u \mid u$ has a searcher during the phase j of $\mathscr{S}\}$. Note that since the vertices cleared in phase j of $\mathscr{S}$ are the same as the vertices cleared at step j of $\mathscr{Y}$,

$S_j \backslash S_{j+1} = Y_j \backslash Y_{j+1}$. Before proving that Algorithm OES constructs an optimal edge-search strategy, we need the following lemma.

Lemma 22. *$S_j \subseteq Y_j$ for $1 \leqslant j \leqslant r$.*

Proof. Let $N[W] = \bigcup_{w \in W} N[w]$ for a vertex set W. We prove this lemma by induction on the phase number. As a basis, we consider S_1. All the vertices in $Y_1 \backslash Y_2$ are cleared at step 1 of $\mathscr{Y}$ and in phase 1 of $\mathscr{S}$. By the clearing rules of node searching, $N[Y_1 \backslash Y_2] \subseteq Y_1$. By the clearing moves of OES, $S_1 = N[Y_1 \backslash Y_2]$. Hence $S_1 \subseteq Y_1$.

Next, assume that $S_i \subseteq Y_i$ for all i, $1 \leqslant i \leqslant k-1$. Let $W_k = \{w \mid w \in N[Y_k \backslash Y_{k+1}]$ and $w \notin S_{k-1}\}$. Then $S_k = (S_{k-1} \backslash (Y_{k-1} \backslash Y_k)) \cup (Y_k \backslash Y_{k+1}) \cup W_k$. By the induction hypothesis, $S_{k-1} \subseteq Y_{k-1}$. Hence $S_{k-1} \backslash (Y_{k-1} \backslash Y_k) \subseteq Y_{k-1} \backslash (Y_{k-1} \backslash Y_k) = Y_{k-1} \cap Y_k$. Since the clearing sequence of $\mathscr{S}$ is the same as $\mathscr{C}$ (obtained from $\mathscr{Y}$), $(Y_k \backslash Y_{k+1}) \cup W_k \subseteq Y_k$. Therefore $S_k \subseteq Y_k$. □

Lemma 23. *Given a sprout tree T and a clearing sequence $\mathscr{C}$ corresponding to an optimal node-search strategy $\mathscr{Y}$ of T, Algorithm* OES($T, \mathscr{C}$) *constructs an optimal edge-search strategy of T in linear time.*

Proof. Since OES clears all the vertices of T, the strategy $\mathscr{S}$ constructed by OES is an edge-search strategy of T. In the following, we consider the phases of $\mathscr{S}$. For simplicity, we assume $S_0 = S_{r+1} = \emptyset$. For an iteration i, let j be the phase containing this iteration. We prove this lemma by showing that in iteration i, at most $|S_j|$ searchers are used to clear v_i in phase j.

We first consider the case that v_i is the first cleared vertex in $S_j \backslash S_{j+1}$. There are two cases.

Case 1: v_i has only one uncleared incident edge. Let the uncleared edge be (v_i, u). As in OES, edge (v_i, u) is cleared by moving the searcher on v_i to u. Since $\{v_i, u\} \subseteq S_j$, no more than $|S_j|$ searchers are used in iteration i.

Case 2: v_i has more than one uncleared incident edges. In this case, v_i must be an internal vertex. By our assumption on $\mathscr{C}$, if $Y_j \backslash Y_{j+1}$ contains a leaf, then the first cleared vertex in phase j of $\mathscr{S}$ is a leaf. Therefore, all the vertices in $Y_j \backslash Y_{j+1}$ $(= S_j \backslash S_{j+1})$ are internal vertices. Since T is a sprout tree, v_i is adjacent to a leaf, say u. The leaf u must be cleared in some phase k $(< j)$ because $u \notin Y_j \backslash Y_{j+1}$. Thus, v_i must be guarded in phase $j-1$, i.e., $v_i \in S_{j-1}$.

Let $U_i = \{x \mid x$ is an unguarded and uncleared neighbor of v_i at the beginning of iteration $i\}$. By Assumption (2) on $\mathscr{Y}$, U_i is not empty; otherwise by the fact that $v_i \in S_{j-1} \subseteq Y_{j-1}$, v_i is clear at the step $j-1$ in $\mathscr{Y}$, which contradicts to that v_i is cleared at step j in $\mathscr{Y}$. Thus we have at least $|U_i|$ $(\geqslant 1)$ free searchers at the beginning of iteration i. By using any free searcher, the uncleared edges (v_i, u) with $u \notin U_i$ are cleared first. After that all the uncleared edges (v_i, u) with $u \notin U_i$ are cleared, we still have at least $|U_i|$ free searchers. Then clear the uncleared edges (v_i, u) with $u \in U_i$. Once the edge (v_i, u) is cleared, u is guarded for all $u \in U_i$. Hence after all the vertices

in U_i are guarded, v_i is cleared and the searcher on v_i can be removed. Since v_i is the first cleared vertex in phase j, the number of guarded vertices at the beginning of iteration i is $|S_j \cap S_{j-1}|$. Since $U_i \subseteq S_j \backslash S_{j-1}$, we use $|S_j \cap S_{j-1}| + |U_i| \leqslant |S_j|$ searchers in iteration i.

Note that after v_i is cleared, we always have at least one free searcher in the rest of phase j.

Now we consider the case that v_i is not the first cleared vertex in $S_j \backslash S_{j+1}$. Recall that U_i denotes the set of unguarded and uncleared neighbors of v_i at the beginning of iteration i. For uncleared edges (v_i, u) with $u \notin U_i$, we clear them by using a free searcher which is freed from the first cleared vertex of phase j. For uncleared edges (v_i, u) with $u \in U_i$, we clear them by using $|U_i|$ searchers. Since $U_i \subset S_j$, we use at most $|S_j|$ searchers in iteration i. That is, we use $|S_j|$ searchers to clear all the vertices in $S_j \backslash S_{j+1}$ in phase j. By Lemma 22, Algorithm OES uses at most $\max_j |Y_j| = ns(T)$ searchers to clear T. By Theorem 20, $\mathscr{S}$ is optimal.

In Algorithm OES, we scan the vertices according to their orders in $\mathscr{C}$. For each scanned vertex, we only clear its uncleared incident edges. Hence, Algorithm OES runs in linear time. □

Theorem 24. *An optimal edge-search strategy of a tree can be obtained in linear time.*

Proof. We design an algorithm to construct an optimal edge-search strategy for any tree in the following. For any tree T, if T is not a path, then we first obtain the reduction of T, say T'. Next, we obtain the sprout of T', say T''. We first obtain a clearing sequence according to an optimal node-search strategy of T'' by using a linear-time algorithm [35] and then transform it to an optimal edge-search strategy $\mathscr{S}''$ for T'' using Algorithm OES. We then obtain an edge-search strategy $\mathscr{S}'$ for T' from $\mathscr{S}''$ by deleting all allowable moves clearing the leaves which are added by sprout operations. For each edge $(u, v) \in E(T')$ but $(u, v) \notin E(T)$, there exists a path from u to v in T and each vertex $(\neq u, v)$ in this path has degree 2. The *expanding* of (u, v) from $\mathscr{S}'$ is to modify $\mathscr{S}'$ such that the clearing moves of (u, v) is replaced by the clearing moves of a path from u to v. Our edge-search strategy $\mathscr{S}$ for T is obtained from $\mathscr{S}'$ by expanding all the edges $(u, v) \in E(T')$ but $(u, v) \notin E(T)$. Since $\mathscr{S}$ uses the same number of searchers as $\mathscr{S}''$, by Lemmas 14 and 15, $\mathscr{S}$ is an optimal edge-search strategy for T. It is not difficult to see that the deletions of added leaves and the expansions of degree-2 vertices can be done in linear time. □

Theorem 24 answers positively the question proposed by Megiddo et al. [24] of whether an optimal edge-search strategy for any tree can be constructed in linear time.

Let T be a tree and $V(T) = n$. A *linear layout* of T is a one-to-one function L mapping the vertices of T to $\{1, 2, \ldots, n\}$. For $1 \leqslant i < n$, let $\sigma(L, i)$ denote the number of edges (u, v) of T with $L(u) \leqslant i < L(v)$. The *cutwidth of T under L*, denoted by $cw(T, L)$, is $\max\{\sigma(L, i) \mid 1 \leqslant i < n\}$. The *cutwidth* of T, denoted by $cw(T)$, is $\min\{cw(T, L) \mid L$ is

a linear layout of $T\}$. Given a graph G and a positive integer k, the *cutwidth problem* is the problem to determine whether $cw(G)\leqslant k$ and the *min-cut linear arrangement problem* is the problem to find a linear layout L of G such that $cw(G,L)\leqslant k$.

Chung et al. [9] proved that for any tree with the maximum degree 3, its edge-search number and cutwidth are identical. They also gave an $O(n \log n)$-time algorithm to determine the cutwidth and a corresponding linear layout for any tree with the maximum degree 3. Yannakakis improved this result to an arbitrary tree in $O(n \log n)$ time [36]. Makedon and Sudborough showed a more general result such that $es(G)=cw(G)$ for an arbitrary graph G with the maximum degree 3 [23]. They also constructed an optimal linear layout for a graph G with the maximum degree 3 based on an optimal edge-search strategy of G in linear time. By combining results of [23] and Theorem 24, we have the following theorem.

Theorem 25. *An optimal min-cut linear layout of a tree with the maximum degree* 3 *can be obtained in linear time.*

5. Conclusions

In this paper, we establish a relationship between the node searching and edge searching problems on trees. The bridge is built from an n-avenue system and an e-avenue system of a tree. We currently do not know how to construct an optimal edge-search strategy for a tree from any one of its e-avenue systems in linear time. However, we show that for a sprout tree, its optimal edge-search strategy can be obtained from its any optimal node-search strategy without using its avenue systems. This result leads to a linear-time algorithm for constructing an optimal edge-search strategy for any tree. This also answers positively the question proposed by Megiddo et al. [24] of whether an optimal edge-search strategy for any tree can be constructed in linear time. Furthermore, it leads to a linear-time algorithm to construct a min cut linear layout for any tree with the maximum degree 3.

Acknowledgements

We thank anonymous referees for valuable suggestions that improve the presentation of this paper and for pointing out references [11, 12].

References

[1] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebra Discrete Meth. 8 (1987) 277–284.

[2] D. Bienstock, Graph searching, path-width, tree-width and related problems (a survey), in: F. Roberts, F. Hwang, C. Monma (Eds.), Reliability of Computer and Communication Networks, DIMACS series in Disc. Math. and Theoretical Comp. Scie., Vol. 5, American Math. Society, Providence, RI, 1991, pp. 33–49.

[3] H.L. Bodlaender, T. Kloks, D. Kratsch, Treewidth and pathwidth of permutation graphs, SIAM J. Discrete Math. 8 (1995) 606–616.
[4] H.L. Bodlaender, T. Kloks, D. Kratsch, H. Müller, Treewidth and minimum fill-in on d-trapezoid graphs, Technical Report UU-CS-1995-34, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 1995.
[5] H.L. Bodlaender, T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, J. Algorithms 21 (1996) 358–402.
[6] H.L. Bodlaender, R.H. Möhring, The pathwidth and treewidth of cographs, SIAM J. Discrete Math. 6 (1993) 181–188.
[7] D. Bienstock, P. Seymour, Monotonicity in graph searching, J. Algorithms 12 (1991) 239–245.
[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, The MIT Press, New York, 1992.
[9] M.-J. Chung, F. Makedon, I.H. Sudborough, J. Turner, Polynomial time algorithms for the min cut problem on degree restricted trees, SIAM J. Comput. 14 (1985) 158–177.
[10] J.A. Ellis, I.H. Sudborough, J.S. Turner, The vertex separation and search number of a graph, Inform. Comput. 113 (1994) 50–79.
[11] P.A. Golovach, Extremal searching problems on graphs, Ph.D. thesis, Leningrad, 1990 (in Russian).
[12] P.A. Golovach, Search number, node search number, and vertex separator of a graph, Vestn. Leningr. Univ., Math. 24(1) (1991) 88–90.
[13] P.A. Golovach, N.N. Petrov, The search number of a complete graph, Vestn. Leningr. Univ., Math. 19(4) (1986) 15–19.
[14] J. Gustedt, On the pathwidth of chordal graphs, Discrete Appl. Math. 45 (1993) 233–248.
[15] L.M. Kirousis, C.H. Papadimitriou, interval graph and searching, Discrete Math. 55 (1985) 181–184.
[16] L.M. Kirousis, C.H. Papadimitriou, searching and pebbling, Theoret. Comput. Sci. 47 (1986) 205–218.
[17] N.G. Kinnersley, The vertex separation number of a graph equals its path-width, Inform. Process. Lett. 42 (1992) 345–350.
[18] T. Kloks, Treewidth — Computations and Applications, Lecture Notes in Computer Sciense, Vol. 842, Springer, Berlin, 1994.
[19] T. Kloks, H. Bodlaender, H. Müller, D. Kratsch, Computing treewidth and minimum fill-in: all you need are the minimal separators, ESA'93, Lecture Notes in Computer Sciense, Vol. 726, pp. 260–271, 1993. Erratum: ESA'94, Lecture Notes in Computer Sciense, Vol. 855, pp. 508, 1994.
[20] A. Kornai, Z. Tuza, Narrowness, pathwidth, and their application in natural language processing, Discrete Appl. Math. 36 (1992) 87–92.
[21] A.S. LaPaugh, Recontamination does not help to search a graph, J. Assoc. Comput. Mach. 40 (1993) 224–245.
[22] F. Makedon, C.H. Papadimitriou, I.H. Sudborough, Topological bandwidth, SIAM J. Algebra Discrete Meth. 6 (1985) 418–444.
[23] F. Makedon, I.H. Sudborough, On minimizing width in linear layouts, Disc. Appl. Math. 23 (1989) 243–265.
[24] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, C.H. Papadimitriou, The complexity of searching a graph, J. Assoc. Comput. Mach. 35 (1988) 18–44.
[25] R.H. Möhring, Graph problems related to gate matrix layout and PLA folding, in: G. Tinnhofer et al. (Eds.), Computational Graph Theory, Springer, Wien, 1990, pp. 17–32.
[26] B. Monien, I.H. Sudborough, Min cut is NP-complete for edge weighted trees, Theoret. Comput. Sci. 58 (1988) 209–229.
[27] T.D. Parsons, Pursuit-evasion in a graph, in: Y. Alavi, D.R. Lick (Eds.), Theory and Applications of Graphs, Springer, New York, 1976, pp. 426–441.
[28] T.D. Parsons, The search number of a connected graph, Proc. 9th S-E Conf. on Combinatorics, Graph Theory, and Computing, 1978, pp. 549–554.
[29] S.L. Peng, C.W. Ho, T.-s. Hsu, M.T. Ko, C.Y. Tang, Edge and node searching problems on trees, Technical Report TR-IIS-98-015, Institute of Information Science, Academia Sinica, Taiwan, 1998 (ftp.iis.sinica.edu.tw).
[30] S.L. Peng, M.T. Ko, C.W. Ho, T.-s. Hsu, C.Y. Tang, Graph searching on chordal graphs, ISAAC'96, Lecture Notes in Computer Science, Vol. 1178, pp. 156–165, 1996.
[31] N.N. Petrov, Pursuit problems without information about the evader, Differents. Uravn. 18(8) (1982) 1345–1352.

[32] N.N. Petrov, S.A. Starostina, Minimal graphs with a search number less than four, Vestn. Leningr. Univ., Math. 22(3) (1989) 66–68.
[33] N. Robertson, P.D. Seymour, Graph minors I. Excluding a forest, J. Combin. Theory Ser. B 35 (1983) 39–61.
[34] P. Scheffler, A linear algorithm for the pathwidth of trees, in: R. Bodendiek, R. Henn (Eds.), Topics in Combinatorics and Graph Theory, Physica-Verlag, Heidelberg, 1990, pp. 613–620.
[35] P. Scheffler, Optimal embedding of a tree into an interval graph in linear time, Fourth Czechoslovakian Symposium on Combinatorics, Graphs and Complexity, in: J. Nesetril, M. Fiedler (Eds.), Annals of Discrete Mathematics, Vol. 51, North-Holland, Amsterdam, 1992, pp. 287–291.
[36] M. Yannakakis, A polynomial algorithm for the min cut linear arrangement of trees, J. Assoc. Comput. Mach. 32 (1988) 950–959.