Edge and Node Searching Problems on Trees

Sheng-Lung Peng*
National Tsing Hua University,Taiwan
e-mail: dr818303@cs.nthu.edu.tw

Chin-Wen Ho
National Central University, Taiwan
e-mail: hocw@csie.ncu.edu.tw

Tsan-sheng Hsu*
Academia Sinica, Taiwan
e-mail: tshsu@iis.sinica.edu.tw

Ming-Tat Ko*
Academia Sinica,Taiwan
e-mail: mtko@iis.sinica.edu.tw

Chuan Yi Tang
National Tsing Hua University, Taiwan
e-mail: cytang@cs.nthu.edu.tw

April 13, 1998

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 three from $O(n \log n)$ to $O(n)$.

1 Introduction

The graph searching problem was first proposed by Parsons [Pa76, Pa78] and independently proposed by Petrov [Pe82]. 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.

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

The graph searching problem is not only interesting theoretically, but also have applications on several combinatorial problems [Bi91, CMST85, Ki92, KP85, KT92, Mo90, MPS85, RS83].

In this paper, we consider the edge searching problem and the node searching problem on trees. In *node searching* [KP86], the allowable moves are (1) placing a searcher on a vertex and (2) removing a searcher from a vertex. A contaminated edge is *cleared* if both its two endpoints simultaneously contain searchers. In *edge searching* [Pa76], 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 [KP86, BS91] (respectively, [La93, BS91]) that there always exists an optimal node-search (respectively, edge-search) strategy for a graph that does not recontaminate any edge. Kirousis and Papadimitriou [KP86] proved that for any graph G, $ns(G)-1 \leq es(G) \leq ns(G)+1$. 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 [Mo90]. The problem of finding the node-search number is equivalent to the pathwidth problem [RS83, Mo90], the interval thickness problem [KP85], the narrowness problem [KT92], and the vertex separation problem [KP86, Ki92]. From the equivalent of the above problems, the node searching problem is NP-complete on planar graphs with vertex degree at most three [MoS88], starlike graphs (a proper subclass of chordal graphs) [Gu93], bipartite graphs [Kl93], cobipartite graphs (i.e., complement of bipartite graphs) [ACP87], and bipartite distance-hereditary graphs (a proper subclass of the chordal bipartite graphs and distance-hereditary graphs) [KBMK93]. For some special classes of graphs, it can be solved in polynomial time, as e.g., trees [Mo90, Sc90, EST94], cographs [BM93], permutation graphs [BKK95], trapezoid graphs [BKKM95], split graphs [Gu93, Kl93], partial k-trees [BK96], and k-starlike graphs for a fixed k [Gu93, PKHHT96].

The edge searching problem is equivalent to the min-cut linear arrangement problem for any graph with the maximum degree 3 [MaS89]. The edge searching problem is NP-complete on general graphs [MHGJP88], planar graphs with the maximum vertex degree 3 [MoS88] and starlike graphs [PKHHT96]. However, it can be solved in polynomial time on complete graphs [GP86], trees [MHGJP88], interval graphs, split graphs, and k-starlike graphs for a fixed $k \geq 2$ [PKHHT96].

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 [Sc90, Sc92] (also mentioned in [Mo90, Theorem 4.7]). However, the previous best algorithm [MHGJP88] 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 [MHGJP88]. 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.* [MHGJP88] 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 [MHGJP88]. 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 [Go90, Go91] obtained similar results. In [Go91], Golovach mentioned that if a graph G has no vertices of degree 2 and is different from the complete graph with two vertices then $ns(G) \leq es(G)$. Unfortunately, no detail is given. We were also told that Golovach [Go90] 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, $\deg(v) \geq 3$, 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 $ns(G) \geq es(G)$.

The remains 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 Sections 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 conclusion 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 $(v_i, v_{i+1}) \in E(T)$, $1 \leq i \leq r-1$. A vertex in T with degree 1 is called a *leaf* and a non-leaf 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.

2.1 Edge searching

Lemma 2.1 *[Pa76] If G' is a subgraph of G then $es(G') \leq es(G)$.*

Lemma 2.2 *[Pa76] For any tree T and an integer $k \geq 1$, $es(T) \geq k+1$ 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 $es(T^+_{tu}) \geq k$, $es(T^+_{tv}) \geq k$, and $es(T^+_{tw}) \geq k$.*

From Lemma 2.2, Megiddo *et al.* [MHGJP88] 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, $2 \leq j \leq r-1$, 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, $1 \leq i \leq r$, 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 2.3 *[MHGJP88] For any tree T, T 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 $es(T) \geq 2$, every internal vertex is an e-hub [MHGJP88].

Lemma 2.4 *For any tree T of $es(T) \geq 2$, any leaf of T cannot be an e-hub or a vertex of the e-avenue.*

Proof. Let v be any leaf of T and let u be the neighbor of v. If T has an e-avenue containing v, then u must belong to this e-avenue. Since the e-branch T^+_{uv} contains the two vertices u, v and the edge (u,v), $es(T^+_{uv}) = 1$. By the condition (2) of the e-avenue, u does not belong to the e-avenue. This is a contradiction. That is, v cannot be a vertex of the e-avenue of T. If T has no e-avenue, then we assume T has an e-hub. In this case, the e-branch at v is T itself. By the definition of an e-hub, v cannot be an e-hub of T. □

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* $\mathcal{A}^e(T)$ and the set of nonavenue e-branches $\mathcal{F}(\mathcal{A}^e(T))$ as follows.

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

With respect to $\mathcal{A}^e(T)$, *e-labels* of vertices in T are defined as follows. Firstly, for each tree T' in $\mathcal{F}(\mathcal{A}^e(T))$ with $es(T') \geq 2$, the e-label of any vertex in the e-avenue of T' in $\mathcal{A}^e(T)$ is $es(T')$. Secondly, for each tree T' in $\mathcal{F}(\mathcal{A}^e(T))$ 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 the e-label of a vertex v is at least 2, v belongs to the e-avenue of exactly one tree in $\mathcal{F}(\mathcal{A}^e(T))$. By Lemma 2.4 and the way we labeled, v will not be relabeled. If the e-label of a vertex v is 1, then v cannot have any e-label whose value is not 1. An example of the e-avenue system is shown in Figure 1(b). In the example, let T be the tree in Figure 1(a). We choose $[w]$ as an e-avenue of T and the e-branches at w are T^+_{wq}, T^+_{wu} and T^+_{wx}. The e-avenues of T^+_{wq}, T^+_{wu} and T^+_{wx} are $[c, f, j, l, q]$, $[u]$ and $[x]$, respectively. By the recursive definition, $\mathcal{A}^e(T) = \{[w], [c, f, j, l, q], [u], [x], [a, c], [b, c], [d, f], [f, g, h], [j, k], [l, m, o], [p, q], [q, r], [t, u], [u, v], [x, y], [x, z]\}$ and the corresponding e-labels are depicted in Figure 1(b).

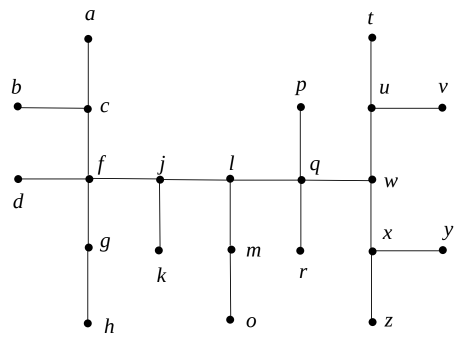
By definition, each vertex v with the e-label at least 2, is in an e-avenue of a subtree of T in $\mathcal{F}(\mathcal{A}^e(T))$. We denote this tree by T^v. Let i be the e-label of v in $\mathcal{A}^e(T)$. Then $es(T^v) = i$ and T^v is a nonavenue e-branch at u of T^u for some u whose e-label is at least $i + 1$. Note that if the e-label of v is $es(T)$, then $T^v = T$. If $es(T^v) \geq 2$, 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. For example, in Figure 1(a), the subtree T^f is T^+_{wq}.

Given $\mathcal{A}^e(T)$, we design the following algorithm to construct an optimal edge-search strategy of T.

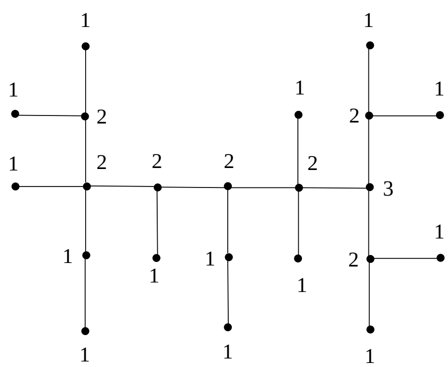
Algorithm ES(T, $\mathcal{A}^e(T)$);
  Let $[v_1, v_2, \ldots, v_r] \in \mathcal{A}^e(T)$ be an e-avenue of T;
  place a searcher on v_1;
  if $es(T) = 1$ **then** move the searcher at v_1 to v_r via v_i, $2 \leq i \leq r - 1$
  else
    for $i := 1$ **to** r **do**
      for each nonavenue e-branch T' at v_i **do** ES(T', $\mathcal{A}^e(T')$); /* $\mathcal{A}^e(T') \subset \mathcal{A}^e(T)$ */
      if $i < r$ **then** move the searcher at v_i to v_{i+1} along (v_i, v_{i+1})
    end for
  end if;
  remove the searcher on v_r
end ES;

Lemma 2.5 *Let T be a tree and $\mathcal{A}^e(T)$ be its e-avenue system. Then Algorithm $ES(T, \mathcal{A}^e(T))$ constructs an optimal edge-search strategy of T.*

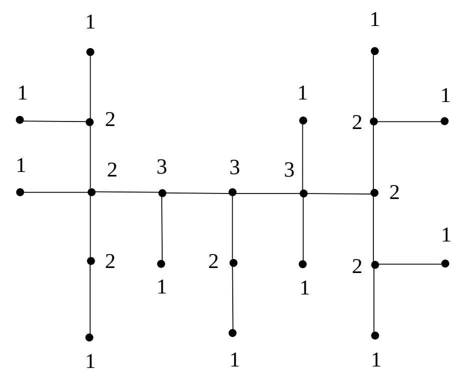
Proof. We prove this lemma by induction on $es(T)$. If $es(T) = 1$, then $T = [u_1, u_2, \ldots, u_s]$ is a path and $\mathcal{A}^e(T) = \{[u_1, u_2, \ldots, u_s]\}$. In Algorithm *ES*($T$, $\mathcal{A}^e(T)$), T is cleared by placing



(a) A tree T.



(b) An e-avenue system with e-labels of T.



(c) An n-avenue system with n-labels of T.

Figure 1: Avenue systems of a tree T.

a searcher on u_1 and then move it to u_s via u_i, $2 \leq i \leq s-1$. Since only one searcher is used, Algorithm $ES(T,\ \mathcal{A}^e(T))$ constructs an optimal edge-search strategy for T.

Assume that for all trees T with $2 \leq es(T) \leq k-1$, $ES(T, \mathcal{A}^e(T))$ constructs an optimal edge-search strategy of T. Now we consider a tree T with $es(T) = k$. Let $[v_1, v_2, \ldots, v_r] \in \mathcal{A}^e(T)$ be its e-avenue. In Algorithm $ES(T, \mathcal{A}^e(T))$, 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. For each nonavenue e-branch T' at v_1, let $\mathcal{A}^e(T')$ denote the e-avenue system of T' contained in $\mathcal{A}^e(T)$. By the induction hypothesis, T' can be cleared by the optimal edge-search strategy constructed by $ES(T', \mathcal{A}^e(T'))$, which uses at most $k-1$ searchers. 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, $2 \leq i \leq r$, one after one using at most $k-1$ searchers. After all the nonavenue e-branches at v_r are cleared, T is cleared. Hence Algorithm $ES(T,\ \mathcal{A}^e(T))$ uses exactly k searchers to clear T. Thus our lemma is proved. $\square$

As an example, for the tree T in Figure 1(a), we can clear T using the e-avenue system depicted in Figure 1(b) as follows. We first place a searcher on w. Secondly, we clear the e-branch T^+_{wq}. Since $[c, f, j, l, q]$ is the e-avenue of T^+_{wq}, we place a second searcher on c. By using the third searcher, we can clear (a, c) and (b, c). Next, we move the searcher at c to f along the edge (c, f). Then the edge (d, f) and the path $[f, g, h]$ can be cleared by the third searcher. By the similar process, we can clear the vertices j, l and q. After the e-branch T^+_{wq} is cleared, we have two free searchers. By using these two searchers, we can clear the e-branches T^+_{wu} and T^+_{wx}. Thus T can be cleared using 3 searchers.

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 2.4 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 2.6 *For any tree T with no vertex of degree 2 and $|V(T)| \geq 4$, no internal vertex is labeled with 1 in any e-avenue system of T.*

Proof. Let $\mathcal{A}^e(T)$ be an e-avenue system of T. By Lemma 2.4, 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. $\square$

2.2 Node searching

Let G be a graph. According to [KP85], an optimal node-search strategy for G can be represented by a sequence of vertex sets $\mathcal{Y} = (Y_1, \ldots, Y_r)$, where $Y_i \subseteq V(G)$ is a set of vertices guarded by searchers at step i for $1 \leq i \leq r$. 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 $i \leq j$. A vertex u is *cleared at step* i if it is the first step that all the incident edges of u are clear. Recall that we only consider the node-search strategies which do not recontaminate any edge. Since recontamination does not occur, if v is guarded at step i and is cleared at step j, then $v \in Y_t$ for $i \leq t \leq j$. The node-search number of $\mathcal{Y}$ is $\max_i |Y_i|$. Note that $(Y_1, \ldots, Y_r)$ is also called a *path-decomposition* of the graph G [Mo90]. For any subgraph G' of G, $\mathcal{Y}' = (Y_1 \cap V(G'), \ldots, Y_r \cap V(G'))$ is a node-search strategy of G' which use at most $\max_i |Y_i \cap V(G')|$ searchers. Thus we have the following lemma.

Lemma 2.7 *If G' is a subgraph of G then $ns(G') \leq ns(G)$.*

Let T be a tree. If T contains any edge, then $ns(T) \geq 2$. For convenience, we define $ns(T) = 1$ if T contains only one vertex. Thus, $ns(T) \geq 2$ if and only if there exists a vertex $t \in V(T)$ with at least one branch. The necessary and sufficient conditions for $ns(T) \geq k+1$, $k \geq 2$, were provided by Scheffler [Sc90]. The following lemma is due to Scheffler [Sc90].

Lemma 2.8 *[Sc90] For any tree T, $ns(T) \geq k+1$ for $k \geq 2$ 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 $ns(T_{tu}) \geq k$, $ns(T_{tv}) \geq k$, and $ns(T_{tw}) \geq k$. For any tree T, $ns(T) \geq 2$ if and only if there exists a vertex $t \in V(T)$ with at least one branch.*

By Lemma 2.8, 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 $ns(T) = s \geq 2$, 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, $2 \leq j \leq r-1$, 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, $1 \leq i \leq r$, 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 2.9 *Any tree has either an n-hub or a unique n-avenue.*

Proof. Our proof is similar to the proof of Lemma 2.3 in [MHGJP88]. Let T be a tree with no n-hub. Let $s = ns(T)$. That is, every vertex in T has at least one branch with the node-search number s. Consider the set B of all edges (u, v) of T with the property that the search numbers of T_{uv} and T_{vu} are both s. In the following, we will show that B itself is an n-avenue.

First, we show that B is nonempty. Since T does not have an n-hub, for every vertex $v \in V(T)$, v has at least a neighbor v' such that $ns(T_{vv'}) = s$. In other words, there is a mapping from $V(T)$ to $E(T)$ such that every vertex v maps to an edge (v, v') where $ns(T_{vv'}) = s$. Since T has n vertices and $n-1$ edges, there must exist two vertices u and v such that both of them map to the same edge (u, v), i.e., $ns(T_{uv}) = s = ns(T_{vu})$. That is, $(u, v) \in B$.

We next show that B is a path. By Lemma 2.8, it is impossible to have a vertex with three branches and whose node-search numbers are all s. Thus we only need to show that B is connected. Suppose that B is not connected. Let $(u, v) \notin B$ be an edge on a path in T joining two disconnected components of B. Both T_{uv} and T_{vu} contain an edge from B. Hence the node-search numbers of T_{uv} and T_{vu} are both s. This implies that (u, v) belongs to B, which is a contradiction. It follows that B is connected and is a path. □

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 can be an n-hub in a tree.

Lemma 2.10 *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 of T.*

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 $T' = T\backslash\{v\}$. 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 2.7, 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. Hence our lemma is proved. □

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* $\mathcal{A}^n(T)$ and the set of nonavenue branches $\mathcal{F}(\mathcal{A}^n(T))$ as follows.

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

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

With respect to $\mathcal{A}^n(T)$, *n-labels* of vertices in T are defined as follows. For each tree T' in $\mathcal{F}(\mathcal{A}^n(T))$, the n-label in $\mathcal{A}^n(T)$ of any vertex in the n-avenue of T' is $ns(T')$. An example of the n-avenue system is shown in Figure 1(c). In the example, let T be the tree in Figure 1(a). The n-avenue of T is $[j, l, q]$. The nonavenue branches at j are T_{jf} and T_{jk}. The n-avenues of T_{jf} and T_{jk} are $[c, f, g]$ and $[k]$, respectively. By the recursive definition, $\mathcal{A}^n(T) = \{[j, l, q], [c, f, g], [m], [u, w, x], [a], [b], [d], [h], [k], [o], [p], [r], [t], [v], [y], [z]\}$ and the corresponding n-labels are depicted in Figure 1(c). 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 $\mathcal{F}(\mathcal{A}^n(T))$, if T' has an n-hub then, by Lemma 2.10, 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 2.11 *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 $\mathcal{F}(\mathcal{A}^n(T))$, we denote this tree by T^v. Let i be the n-label of v in $\mathcal{A}^n(T)$. 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 $ns(T^v) \geq 2$, then nonavenue branches at v in the subtree T^v are referred in the following as nonavenue branches at v without specifying the subtree. For example, in Figure 1(a), the subtree T^c is T_{jf}, i.e., the branch at j containing f.

Given $\mathcal{A}^n(T)$, we also design the following algorithm to construct an optimal node-search strategy of T.

Algorithm NS(T, $\mathcal{A}^n(T)$);
  Let $[v_1, v_2, \ldots, v_r] \in \mathcal{A}^n(T)$ be an n-avenue of T;
  place a searcher on v_1;
  if $ns(T) = 1$ (i.e., $r = 1$) **then** remove the searcher on v_1
  else
    for $i := 1$ **to** r **do**
      for each nonavenue branch T' at v_i **do** NS(T', $\mathcal{A}^n(T')$); /* $\mathcal{A}^n(T') \subset \mathcal{A}^n(T)$ */
      if $i < r$ **then**
        place a searcher on v_{i+1}; /* clear the edge (v_i, v_{i+1}) */
        remove the searcher on v_i
      end if
    end for;
    remove the searcher on v_r
  end if
end NS;

Lemma 2.12 *Let T be a tree and $\mathcal{A}^n(T)$ be its n-avenue system. Then Algorithm $NS(T, \mathcal{A}^n(T))$ constructs an optimal node-search strategy of T.*

Proof. The proof is similar to Lemma 2.5. In the case of $ns(T) = 1$, T consists of a single vertex u and $\mathcal{A}^n(T) = \{[u]\}$. In the case of $ns(T) \geq 2$, let $[v_1, v_2, \ldots, v_r] \in \mathcal{A}^n(T)$ be its n-avenue. While clearing every nonavenue branch at v_i, v_j is cleared for $1 \leq j \leq i-1$. After all the nonavenue branch at v_i are cleared, only v_i contains a searcher. Since $ns(T) \geq 2$, we always have $ns(T) - 1 \geq 1$ free searchers at this time. By placing a free searcher on v_{i+1}, $1 \leq i \leq r-1$, the edge (v_i, v_{i+1}) is cleared and then the searcher at v_i can be removed. The detail is omitted owing to the similarity to the proof of Lemma 2.5. □

For example, for the tree T in Figure 1(a), we can clear T using the n-avenue system depicted in Figure 1(c). Note that the n-avenue of T is $[j, l, q]$. Firstly, we place a searcher on j. There are two nonavenue branches T_{jf} and T_{jk} at j. We first clear T_{jf}. Since $[c, f, g]$ is the n-avenue of T_{jf}, we place a second searcher on c. Next, place the third searcher on a then (a, c) is cleared. Similarly, after a is cleared, we place the third searcher on b and then (b, c) is cleared. Now we place the third searcher on f and then (c, f) and (f, j) are simultaneously cleared. After c is cleared, we have one free searcher again. We then place this free searcher on d and (d, f) is cleared. Next, we place a searcher on g, then f is cleared and we have a free searcher. By placing this free searcher on h, the branch T_{jf} is cleared. Now we have two free searchers. By using a searcher, we can clear T_{jk}. After T_{jf} and T_{jk} are cleared, we place a searcher on l then j is cleared. By using the similar process, we then clear the vertex l and finally the vertex q. After all the nonavenue branches at q are cleared, T is cleared.

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

Lemma 2.13 *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 $\mathcal{A}^n$ of T satisfying Lemma 2.11. By our definition of n-labels, the neighbor of a vertex with n-label 1 cannot be labeled with 1 in $\mathcal{A}^n$. Hence, there is no internal vertex of T whose n-label is 1 in $\mathcal{A}^n$. $\square$

3 Relation between node and edge searching on trees

In this section, we will show a relationship between node-search strategy 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 [KP86, PS89].

Lemma 3.1 *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 T. Let T^+_{vu} be any e-branch at v in T and $es(T^+_{vu}) \geq 2$. Then the e-branch at v in T' which contains u is the sprout of T^+_{vu}.

Lemma 3.2 *Let T' be the sprout of a reduction tree T. Then $es(T) = es(T')$.*

Proof. We first prove $es(T') \leq es(T)$ by induction on $es(T)$. Since T is a reduction tree, $es(T) \geq 2$. If $es(T) = 2$, by Lemma 2.2, it is impossible to have an internal vertex which has three e-branches of edge-search number no less than 2. Thus the internal vertices of T induce a path. Furthermore, T is a reduction tree which has no degree-2 vertex. Thus each internal vertex of T has a neighbor which is a leaf. That is, $T = T'$. Hence $es(T') = es(T)$. Now, we assume $es(T') \leq es(T)$ for every reduction tree T with $2 \leq es(T) \leq k-1$ and its sprout tree T'.

Let us consider a reduction tree T with $es(T) = k$ and its sprout tree T'. Let $A^e(T) = [v_1, \ldots, v_r]$ be the e-avenue of T. In T', for convenience, we also call the e-branches at v_i which do not contain any v_j, $j \neq i$ and $1 \leq j \leq r$, *nonavenue e-branches* without ambiguity. We provide the following edge-search strategy to clear T' according to $A^e(T)$. We first place a searcher on v_1. Then we clear one by one the nonavenue e-branches at v_1 in T' by optimal edge-search strategies. After all the nonavenue e-branches at v_1 are cleared, we move the searcher at v_1 to v_2, then v_1 is cleared. We continue the above clearing process on $v_2, \ldots, v_r$ sequentially until all the e-branches at v_r are cleared. Then T' is cleared.

We compute the search number we used in our edge-search strategy. Let $\mathcal{T}(T) = \{B | B$ is a nonavenue e-branch at v_i, $1 \leq i \leq r$, in $T\}$ and let $\mathcal{T}(T') = \{B | B$ is a nonavenue e-branch at v_i, $1 \leq i \leq r$, in $T'\}$. Let $\mathcal{T}_1(T') = \{B | B$ is the sprout of T^*, $T^* \in \mathcal{T}(T)$ and

$es(T^*) \geq 2\}$ and $\mathcal{T}_2(T') = \{T^* | T^* \in \mathcal{T}(T)$ and $es(T^*) = 1\}$. If v_i is attached a leaf u_i by a sprout operation, then let T_i be the tree with $V(T_i) = \{v_i, u_i\}$ and $E(T_i) = \{(v_i, u_i)\}$. Let $\mathcal{T}_3(T') = \{T_i | v_i \in A^e(T)$, which is attached a leaf by a sprout operation$\}$. Then $\mathcal{T}(T') = \mathcal{T}_1(T') \cup \mathcal{T}_2(T') \cup \mathcal{T}_3(T')$. By the induction hypothesis, for each e-branch $T^* \in \mathcal{T}_1(T')$, $es(T^*) \leq k-1$. For all $T^* \in \mathcal{T}_2(T') \cup \mathcal{T}_3(T')$, $es(T^*) = 1$. Thus our edge-search strategy uses at most k searchers. Hence $es(T') \leq es(T)$.

Since T is a subtree of T', by Lemma 2.1, $es(T) \leq es(T')$. Thus $es(T) = es(T')$. □

Remark We were informed that Lemma 3.2 is implied by results independently shown in [Go90] (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 3.3 *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, $1 \leq i \leq r$, 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 3.4 *For any reduction tree T, $ns(T) \leq es(T)$.*

Proof. We prove this lemma by induction on the number $es(T)$. Firstly, by Lemma 3.3, if $es(T) = 2$, then $ns(T) = 2$. Next, we assume $ns(T) \leq es(T)$ for every reduction tree T with $2 \leq es(T) \leq k-1$. Now we consider a reduction tree T with $es(T) = k$. Let $\mathcal{A}^e(T)$ be an e-avenue system of T and let $[v_1, v_2, \ldots, v_r] \in \mathcal{A}^e(T)$ be an e-avenue of T. For each nonavenue e-branch $T^+_{v_iu}$ at v_i, $1 \leq i \leq r$, if $es(T^+_{v_iu}) \geq 2$, then $T^+_{v_iu}$ is still a reduction tree. By definitions of the e-avenue, $es(T^+_{v_iu}) \leq k-1$. If $es(T^+_{v_iu}) = 1$, then $V(T^+_{v_iu}) = \{v_i, u\}$.

In the following, we will clear T in the context of node searching based on information available in the e-avenue system $\mathcal{A}^e(T)$ using $es(T)$ searchers. Firstly, we place one searcher on v_1. Then we clear all the nonavenue e-branches $T^+_{v_1u}$ at v_1. By the induction hypothesis, for $T^+_{v_iu}$ of $es(T^+_{v_iu}) \geq 2$, $1 \leq i \leq r$, $ns(T^+_{v_iu}) \leq es(T^+_{v_iu}) \leq k-1$. Thus, $T^+_{v_1u}$ of $es(T^+_{v_1u}) \geq 2$, can be cleared with at most $k-1$ searchers in node searching. To clear an e-branch $T^+_{v_1u}$ of $es(T^+_{v_1u}) = 1$, we only place one searcher on u. Thus we can clear all the nonavenue e-branches $T^+_{v_1u}$ at v_1 in the context of node searching using at most $k-1$ searchers. After these e-branches $T^+_{v_1u}$ are cleared, we have $k-1$ free searchers and we place a free searcher on v_2. Then the edge (v_1, v_2) is cleared, the searcher at v_1 is removed and we have $k-1$ free searchers again. We continue the above clearing process on $v_2, \ldots, v_r$ sequentially until all the e-branches at v_r are cleared. Hence $ns(T) \leq es(T)$. □

Remark Lemma 3.4 is implied by results independently mentioned in [Go91].

Lemma 3.5 *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 $2 \leq ns(T) \leq k-1$, $ns(T') \leq ns(T)$. Now we consider a tree T with $ns(T) = k$ and its n-avenue $A^n(T) = [v_1, \ldots, v_r]$. In T', we also call the branches at v_i which do not contain any v_j, $j \neq i$ and $1 \leq j \leq r$, the *nonavenue branches* without ambiguity. We provide the following node-search strategy for T' according to $A^n(T)$. 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. Note that while we clear a branch $T_{v_1 w}$ at v_1, the edge (v_1, w) is cleared. After all the nonavenue branches at v_1 are cleared, we place a searcher on v_2. Then v_1 is cleared. 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 $\mathcal{T}(T) = \{B | B$ is a nonavenue branch at v_i, $1 \leq i \leq r$, in $T\}$ and $\mathcal{T}(T') = \{B | B$ is a nonavenue branch at v_i, $1 \leq i \leq r$, in $T'\}$. We compute the number of searchers used in the following two cases.

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

2. $v \neq v_i$ for all i, $1 \leq i \leq r$. Let T^* be the nonavenue branch at v_i for some i, $1 \leq i \leq r$ which contains u. By the induction hypothesis, $ns(T^*) \leq k-1$. All the other nonavenue branches in $\mathcal{T}(T')$ are also in $\mathcal{T}(T)$, 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, $ns(T') \leq ns(T)$. Since T is a subtree of T', by Lemma 2.7, $ns(T) \leq ns(T')$. Thus $ns(T) = ns(T')$. □

Lemma 3.6 *For any sprout tree T, $es(T) \leq ns(T)$.*

Proof. We prove this lemma by induction on the number $ns(T)$. Firstly, by Lemma 3.3, if $ns(T) = 2$, then $es(T) = 2$. Next, we assume that for every sprout tree T with $2 \leq ns(T) \leq k-1$, $es(T) \leq ns(T)$. Now let T be a sprout tree with $ns(T) = k$. By Lemma 2.13, we have an n-avenue system $\mathcal{A}^n(T)$ in which no internal vertex of T has n-label 1. Let $[v_1, v_2, \ldots, v_r] \in \mathcal{A}^n(T)$ be an n-avenue of T. For each nonavenue branch $T_{v_i u}$ at v_i, $1 \leq i \leq r$, if $ns(T_{v_i u}) \geq 2$, then $T_{v_i u}$ 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 $T_{v_i u}$. 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 3.5, if $ns(T_{v_i u}) \geq 2$, then $ns(T^+_{v_i u}) = ns(T_{v_i u}) \leq k-1$ and $T^+_{v_i u}$ is a sprout tree. If $ns(T_{v_i u}) = 1$, then $V(T_{v_i u}) = \{u\}$.

In the following, we will clear T by edge searching rules based on the n-avenue system $\mathcal{A}^n(T)$ using $ns(T)$ searchers. For each nonavenue branch $T_{v_i u}$ of $ns(T_{v_i u}) \geq 2$ at v_i,

$1 \le i \le r$, $ns(T^+_{v_iu}) = ns(T_{v_iu}) \le k-1$ and $T^+_{v_iu}$ is a sprout tree. By the induction hypothesis, if $ns(T_{v_iu}) \ge 2$, then $es(T^+_{v_iu}) \le ns(T^+_{v_iu}) \le k-1$. 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 $ns(T_{v_1u}) \ge 2$ 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, $es(T) \le ns(T)$. □

Remark We were informed that Lemma 3.6 is implied by results independently shown in [Go90] (in Russian).

Theorem 3.7 *For any sprout tree T, $es(T) = ns(T)$.*

Proof. Since a sprout tree is also a reduction tree, by Lemmas 3.4 and 3.6, this theorem holds. □

Though there is a linear-time algorithm to determine the edge-search number of a tree [MHGJP88], by using our results, we can also obtain a linear-time algorithm to determine the edge-search number of a tree.

Theorem 3.8 *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 [EST94, Mo90, Sc90] to compute $ns(T'')$. By Lemmas 3.1, 3.2, and Theorem 3.7, $es(T) = ns(T'')$. □

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.

Recall that an optimal node-search strategy $\mathcal{Y}$ can be represented by $(Y_1, \ldots, Y_r)$, where $Y_i \subseteq V(T)$ is a set of vertices guarded by searchers at step i for $1 \le i \le r$. For simplicity of presentation, in the following we assume $Y_0 = Y_{r+1} = \emptyset$. The node-search strategy $\mathcal{Y}$ clears T as follows. At the beginning of step i, $1 \le i \le r$, all the vertices in $Y_i \cap Y_{i-1}$ are guarded. In this step, we guard all the vertices in $Y_i \backslash Y_{i-1}$, i.e., the whole Y_i is guarded. Thus the vertices in $Y_i \backslash Y_{i+1}$ are cleared. Then, all the searchers on the vertices of $Y_i \backslash Y_{i+1}$ are removed. Moreover, there exists an optimal node-search strategy $\mathcal{Y}$ satisfying the following assumptions.

(1) For any vertex $u \in Y_i$, $1 \leq i \leq r$, 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 $2 \leq i \leq r$.

In the following, we consider $\mathcal{Y}$ satisfying the above three assumptions. Note that in $\mathcal{Y}$, any leaf occurs exactly in one step by assumptions (1) and (2).

For each vertex $u \in Y_i \backslash Y_{i+1}$, we say that step i is the *clearing step* of u in $\mathcal{Y}$. According to the clearing steps of vertices, all the vertices of T can be sorted into a sequence $\mathcal{C} = (v_1, v_2, \ldots, v_n)$ such that the clearing step of v_i is no later than the clearing step of v_j if $i < j$. We call $\mathcal{C}$ a *clearing sequence* of $\mathcal{Y}$. Note that all the vertices in $Y_i \backslash Y_{i+1}$ 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 $\mathcal{C}$. The clearing sequence $\mathcal{C}$ plays an important role in constructing our optimal edge-search strategy. In the following, we first show that a clearing sequence $\mathcal{C}$ which corresponds to an optimal node-search strategy satisfying the three assumptions can be constructed in linear time. Then, according to $\mathcal{C}$, we design a linear-time algorithm to construct an optimal edge-search strategy of a sprout tree.

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 $\mathcal{Y} = (Y_1, \ldots, Y_r)$, i.e., $u \in Y_t$ for $a_u \leq t \leq b_u$. Let $I_u = [a_u, b_u]$ for all $u \in V(T)$. A set of intervals forms an interval model of T if the interval graph defined by the set of intervals contains T as a subgraph. Since for all $(u, v) \in E(T)$, $I_u \cap I_v \neq \emptyset$, $\{I_u | u \in V(T)\}$ forms an interval model of T. Note that $Y_i = \{u \in V(T) |\ I_u = [a_u, b_u]$ and $a_u \leq i \leq b_u\}$. An interval model of T is *optimal* if the maximum clique size of the defined interval graph is the smallest among all interval models of T. Conversely, an optimal interval model $F = \{I_u = [a_u, b_u] | u \in V(T)\}$ of T corresponds to an optimal node-search strategy of T where a searcher is placed on u at step a_u and removed after step b_u for all $u \in V(T)$ [KP85].

Scheffler mentioned that an optimal interval model F of T can be constructed in linear time [Sc92]. In general, the node-search strategy corresponds to F may not fulfill assumptions (1), (2) and (3). In order to obtain an optimal interval model F^* whose corresponding optimal node-search strategy satisfying the three assumptions, we make the following modification of F.

Let $N(u) = \{v | v \in V(T)$ and $(u, v) \in E(T)\}$ and $N[u] = \{u\} \cup N(u)$. Let $F = \{I_u = [a_u, b_u] | u \in V(T)\}$. We first modify F into $F' = \{I'_u = [a'_u, b'_u] | u \in V(T)\}$ by setting $a'_u = \max\{a_u, \min\{a_v | v \in N(u)\}\}$ and $b'_u = \max\{a_v | v \in N[u]\}$ for all $u \in V(T)$. It can be verified that F' is an interval model of T by showing that $a'_u \leq b'_u$ for all $u \in V(T)$ and $I'_u \cap I'_v \neq \emptyset$ for all $(u, v) \in E(T)$. Let $\mathcal{Y}'$ 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 $\mathcal{Y}'$ for all $u \in V(T)$. Thus, $\mathcal{Y}'$ satisfies assumption (1). By the setting of b'_u, b'_u is the first time at which u is cleared for all $u \in V(T)$. Thus, $\mathcal{Y}'$ 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 $\mathcal{Y}^*$ satisfying 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 $F^* = \{I^*_u = [a^*_u, b^*_u] | u \in V(T)\}$. Note that F^* preserves the intersection relations of intervals in F'. Let $Y^*_i = \{u \in V(T) | I^*_u = [a^*_u, b^*_u] \in F^*$ and $a^*_u \leq i \leq b^*_u\}$ for $1 \leq i \leq r$ and $\mathcal{Y}^* = (Y^*_1, \ldots, Y^*_r)$. It can be verified that $\mathcal{Y}^*$ 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, $\mathcal{Y}^*$ satisfies assumption (3). A clearing sequence $\mathcal{C}$ corresponding to $\mathcal{Y}^*$ 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 [CLR92], the above sorting processes can be done in linear time. Hence $\mathcal{C}$ 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 $\mathcal{Y} = (Y_1, \ldots, Y_r)$ be an optimal node-search strategy corresponding to F and let $\mathcal{C}$ be a clearing sequence corresponding to $\mathcal{Y}$. Next, we construct an optimal edge-search strategy $\mathcal{S}$ from $\mathcal{C}$ in linear time. In $\mathcal{S}$, the vertices are cleared in the same order as $\mathcal{C}$. The moves of $\mathcal{S}$ are as the following algorithm *OES*.

Algorithm OES(T: sprout tree, $\mathcal{C} = (v_1, \ldots, v_n)$);
  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 $\mathcal{S}$ be the edge-search strategy constructed by Algorithm *OES*. In each iteration of *OES*, a vertex is cleared. Let phase j of $\mathcal{S}$ 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 $\mathcal{S}$, 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 $\mathcal{S}$, it should be guaranteed that no extra searcher is needed to clear the vertices.

Let $S_j = \{u|u$ has a searcher during the phase j of $\mathcal{S}\}$. Note that since the vertices cleared at phase j of $\mathcal{S}$ are the same as the vertices cleared at step j of $\mathcal{Y}$, $S_j \backslash S_{j+1} = Y_j \backslash Y_{j+1}$. Before proving that Algorithm *OES* constructs an optimal edge-search strategy, we need the following lemma.

Lemma 4.1 *$S_j \subseteq Y_j$ for $1 \leq j \leq r$.*

Proof. Let W be a vertex set and let $N[W] = \cup_{w \in W} N[w]$. We prove this lemma by induction. As a basis, we consider S_1. In node searching, since all the vertices in $Y_1 \backslash Y_2$ are cleared at step 1 of $\mathcal{Y}$, $N[Y_1 \backslash Y_2] \subseteq Y_1$. Similarly, in edge searching, $\mathcal{S}$ also clears all the vertices in $Y_1 \backslash Y_2$ at phase 1. Since only the vertices in $N[Y_1 \backslash Y_2]$ have a searcher during phase 1 of $\mathcal{S}$, $S_1 = N[Y_1 \backslash Y_2]$. Hence $S_1 \subseteq Y_1$.

We assume $S_i \subseteq Y_i$ for all i, $1 \leq i \leq k-1$. Now we consider S_k. In edge searching, $\mathcal{S}$ clears vertices in $Y_k \backslash Y_{k+1}$ at phase k. Let $W_k = \{w | w \in N[Y_k \backslash Y_{k+1}]$ and $w \notin S_{k-1}\}$. Since vertices in $Y_{k-1} \backslash Y_k$ are cleared at phase $k-1$ of $\mathcal{S}$, $S_k = (Y_k \backslash Y_{k+1}) \cup W_k \cup (S_{k-1} \backslash (Y_{k-1} \backslash Y_k))$. 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 $\mathcal{S}$ is the same as $\mathcal{C}$ (obtained from $\mathcal{Y}$), $(Y_k \backslash Y_{k+1}) \cup W_k \subseteq Y_k$. Therefore $S_k \subseteq Y_k$. □

Lemma 4.2 *Given a sprout tree T and a clearing sequence $\mathcal{C}$ corresponding to an optimal node-search strategy $\mathcal{Y}$ of T, Algorithm OES(T,$\mathcal{C}$) constructs an optimal edge-search strategy of T in linear time.*

Proof. Since *OES* clears all the vertices of T, the strategy $\mathcal{S}$ constructed by *OES* is an edge-search strategy of T. In the following, we consider the phases of $\mathcal{S}$. For simplicity, we assume $S_0 = S_{r+1} = \emptyset$. Now we consider the number of searchers used in *OES*. We show in the following that in iteration i, at most $|S_j|$ searchers are used for all v_i cleared in phase j.

To guarantee that no extra searcher is needed in phase j, we first consider the case that v_i is the first cleared vertex in $S_j \backslash S_{j+1}$.

1. v_i has only one uncleared incident edge. We assume this uncleared edge is (v_i, u). As in the algorithm, the 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.

2. v_i has more than one uncleared incident edges. By definition, v_i must be an internal vertex. By our assumption on $\mathcal{C}$, if $Y_j \backslash Y_{j+1}$ contains a leaf, then the first cleared vertex in phase j of $\mathcal{S}$ is a leaf. Since v_i is an internal vertex, 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 that is cleared before phase j. That is, v_i must have been guarded at phase $j-1$, i.e., $v_i \in S_{j-1}$. Let $U_i = \{x | x$ is an unguarded neighbor of v_i at the beginning of iteration $i\}$. By assumption (2) on $\mathcal{Y}$, U_i is not empty; otherwise since $v_i \in S_{j-1}$, by Lemma

4.1, $S_{j-1} \subseteq Y_{j-1}$ and therefore v_i is cleared at the step $j-1$ in $\mathcal{Y}$ which contradicts to that v_i is cleared at step j in $\mathcal{Y}$. Thus we have at least $|U_i|$ (≥ 1) free searchers at the beginning of iteration i. By using any free searcher, the uncleared edges (v_i, u) with $u \notin U_i$ can be cleared. After all the uncleared edges (v_i, u) with $u \notin U_i$ are cleared, we still have at least $|U_i|$ free searchers. We then clear the uncleared edges (v_i, u) with $u \in U_i$. Once the edge (v_i, u) is cleared, u is guarded. Hence after all the vertices in U_i are guarded, v_i is cleared and the searcher on v_i is removed. Since v_i is the first cleared vertex in phase j, the number of guarded vertices at the beginning of iteration i is $|S_j \cap S_{j-1}|$. Furthermore, $U_i \subseteq S_j \backslash S_{j-1}$ and $|U_i| \geq 1$. Hence we use $|S_j \cap S_{j-1}| + |U_i| \leq |S_j|$ 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 $S_j \backslash S_{j+1}$. Let $U_i = \{x | x$ is an unguarded neighbor 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 4.1, Algorithm *OES* uses at most $\max_j |Y_j| = ns(T)$ searchers to clear T. By Theorem 3.7, $\mathcal{S}$ is optimal.

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

Theorem 4.3 *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 [Sc92] and then transform it to an optimal edge-search strategy $\mathcal{S}''$ for T'' using Algorithm *OES*. We then obtain an edge-search strategy $\mathcal{S}'$ for T' from $\mathcal{S}''$ 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 $\mathcal{S}'$ is to modify $\mathcal{S}'$ 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 $\mathcal{S}$ for T is obtained from $\mathcal{S}'$ by expanding all the edges $(u, v) \in E(T')$ but $(u, v) \notin E(T)$. Since $\mathcal{S}$ uses the same number of searchers as $\mathcal{S}''$, by Lemmas 3.1 and 3.2, $\mathcal{S}$ 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 4.3 answers positively the question proposed by Megiddo *et al.* [MHGJP88] 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 $1 \leq i < n$, let $\sigma(L, i)$ denote the number of edges

(u, v) of T, with $L(u) \leq i < L(v)$. The *cutwidth of T under L*, denoted by $cw(T, L)$, is $\max\{\sigma(L, i) | 1 \leq i < n\}$. The *cutwidth* of T, denoted by $cw(T)$, is $\min\{cw(T, L) | L$ is a linear layout of $T\}$. Given a graph G and a positive integer k, the *cutwidth problem* is the problem to determine whether $cw(G) \leq k$ and the *min-cut linear arrangement problem* is the problem to find a linear layout L of G such that $cw(G, L) \leq k$.

Chung *et al.* [CMST85] 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 [Ya88]. Makedon and Sudborough showed a more general result such that $es(G) = cw(G)$ for an arbitrary graph G with the maximum degree 3 [MaS89]. 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 [MaS89] and Theorem 4.3, we have the following theorem.

Theorem 4.4 *An optimal min-cut linear layout of a tree with the maximum degree* 3 *can be obtained in linear time.*

5 Conclusion

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.* [MHGJP88] 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.

Acknowledgments. We thank anonymous referees for valuable suggestions that improve the presentation of this paper and for pointing out references [Go90] and [Go91].

References

[ACP87] S. Arnborg, D.G. Corneil, and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM J. Alg. Disc. Meth.*, **8**(1987), 277-284.

[Bi91] D. Bienstock, Graph searching, path-width, tree-width and related problems (a survey), in: F. Roberts, F. Hwang and C. Monma, eds., *Reliability of Computer and Communication Networks*, DIMACS series in Disc. Math. and Theoretical Comp. Scie., Vol **5**, American Math. Society, 1991, 33-49.

[BK96] H.L. Bodlaender and T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, *J. Algorithms*, **21**(1996), 358-402.

[BKK95] H.L. Bodlaender, T. Kloks, and D. Kratsch, Treewidth and pathwidth of permutation graphs, *SIAM J. Disc. Math.*, **8**(1995), 606-616.

[BKKM95] H.L. Bodlaender, T. Kloks, D. Kratsch, and 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.

[BM93] H.L. Bodlaender and R.H. Möhring, The pathwidth and treewidth of cographs, *SIAM J. Disc. Math.*, **6**(1993), 181-188.

[BS91] D. Bienstock and P. Seymour, Monotonicity in graph searching, *J. Algorithms*, **12**(1991), 239-245.

[CLR92] T.H. Cormen, C.E., Leiserson and R.L. Rivest, *Introduction to algorithms*, The MIT Press, 1992.

[CMST85] M.-J. Chung, F. Makedon, I.H. Sudborough, and J. Turner, Polynomial time algorithms for the min cut problem on degree restricted trees, *SIAM J. Comput.*, **14**(1985), 158-177.

[EST94] J.A. Ellis, I.H. Sudborough, and J.S. Turner, The vertex separation and search number of a graph, *Information and Computation*, **113**(1994), 50-79.

[GP86] P.A. Golovach and N.N. Petrov, The search number of a complete graph, *Vestn. Leningr. Univ., Math.*, **19**(1986), No. 4, 15-19.

[Go90] P.A. Golovach, *Extremal searching problems on graphs*, Ph. D. thesis, Leningrad, 1990 (in Russian).

[Go91] P.A. Golovach, Search number, node search number, and vertex separator of a graph, *Vestn. Leningr. Univ., Math.*, **24**(1991), No. 1, 88-90.

[Gu93] J. Gustedt, On the pathwidth of chordal graphs, *Disc. Appl. Math.*, **45**(1993), 233-248.

[KBMK93] T. Kloks, H. Bodlaender, H. Müller, and D. Kratsch, Computing treewidth and minimum fill-in: all you need are the minimal separators, *ESA'93, Lecture Notes in Computer Sciense*, **726**, 260-271, 1993. Erratum: ESA'94, *Lecture Notes in Computer Sciense*, **855**, pp. 508, 1994.

[KP85] L.M. Kirousis and C.H. Papadimitriou, Interval graph and searching, *Disc. Math.*, **55**(1985), 181-184.

[KP86] L.M. Kirousis and C.H. Papadimitriou, Searching and pebbling, *Theoretical Comput. Sci.*, **47**(1986), 205-218.

[KT92] A. Kornai and Z. Tuza, Narrowness, pathwidth, and their application in natural language processing, *Disc. Appl. Math.*, **36**(1992), 87-92.

[Ki92] N.G. Kinnersley, The vertex separation number of a graph equals its path-width, *Inform. Process. Lett.*, **42**(1992), 345-350.

[Kl93] T. Kloks, *Treewidth – Computations and Applications*, Lecture Notes in Computer Sciense, **842**, Springer, Berlin, 1994.

[La93] A.S. LaPaugh, Recontamination does not help to search a graph, *J. Assoc. Comput. Mach.*, **40**(1993), 224-245.

[MHGJP88] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, and C.H. Papadimitriou, The complexity of searching a graph, *J. Assoc. Comput. Mach.*, **35**(1988), 18-44.

[MPS85] F. Makedon, C.H. Papadimitriou and I.H. Sudborough, Topological bandwidth, *SIAM J. Alg. Disc. Meth.*, **6**(1985), 418-444.

[MaS89] F. Makedon and I.H. Sudborough, On minimizing width in linear layouts, *Disc. Appl. Math.*, **23**(1989), 243-265.

[Mo90] 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), 17-32.

[MoS88] B. Monien and I.H. Sudborough, Min cut is NP- complete for edge weighted trees, *Theoretical Comput. Sci.*, **58**(1988), 209-229.

[Pa76] T.D. Parsons, Pursuit-evasion in a graph, in Y. Alavi and D.R. Lick, eds., *Theory and applications of graphs*, Springer-Verlag, New York, 1976, 426-441.

[Pa78] T.D. Parsons, The search number of a connected graph, *Proc. 9th S-E Conf. on Combinatorics, Graph Theory, and Computing*, 549-554, 1978.

[Pe82] N.N. Petrov, Pursuit problems without information about the evader, *Differents. Uravn.*, **18**(1982), No. 8, 1345-1352.

[PKHHT96] S.L. Peng, M.T. Ko, C.W. Ho, T.-s. Hsu, and C.Y. Tang, Graph searching on chordal graphs, *ISAAC'96, Lecture Notes in Computer Science*, **1178**, 156-165, 1996.

[PS89] N.N. Petrov and S.A. Starostina, Minimal graphs with a search number less than four, *Vestn. Leningr. Univ., Math.*, **22**(1989), No. 3, 66-68.

[RS83] N. Robertson and P.D. Seymour, Graph minors I. Excluding a forest, *J. Comb. Theory Ser. B*, **35**(1983), 39-61.

[Sc90] P. Scheffler, A linear algorithm for the pathwidth of trees, in: R. Bodendiek and R. Henn, eds., *Topics in Combinatorics and Graph Theory* (Physica-Verlag, Heidelberg, 1990), 613-620.

[Sc92] 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 and M. Fiedler, eds., *Annals of Discrete Mathematics* (**51**) (North-Holland, 1992), 287-291.

[Ya88] M. Yannakakis, A polynomial algorithm for the min cut linear arrangement of trees, *J. Assoc. Comput. Mach.*, **32**(1988), 950-959.