アルゴリズム 22－7
（1991. 7. 22）

混合探索と真のパス幅

高橋篤司　　上野修一　　梶谷洋司

東京工業大学 電気電子工学科

グラフの探索問題では，最初すべての枝が汚染されているとし，探索によりすべての枝を清掃することを目的とする．清掃方法の違いにより枝探索，点探索が提案されている．小文では枝，点探索の自然な一般化である混合探索を定義する．そして，目的を達成するのに必要な最小探索者数，混合探索数と枝，点探索数との関係を示す．次に，混合探索において一度清掃された枝が再汚染されない混合探索数の探索者を使った手順が必ず存在することを示す．最後に，混合探索数と真のパス幅との関係について述べ，混合探索数を求めることは一般にはＮＰ困難であるが，木については線形時間で求めることができることを示す．

Mixed-Searching and Proper-Path-Width

Atsushi TAKAHASHI, Shuichi UENO, and Yoji KAJITANI

Department of Electrical and Electronic Engineering
Tokyo Institute of Technology, Tokyo, 152 Japan

This paper introduces the mixed-searching game, which is a natural common generalization of the edge-searching and node-searching games extensively studied so far. We establish a relationship between the mixed-search number of a graph G and the proper-path-width of G introduced by the authors in a previous paper. Complexity results are also shown.

1 Introduction

This paper introduces a new version of searching game, called mixed-searching, which is a natural common generalization of the edge-searching and node-searching extensively studied so far. We establish a relationship between the mixed-search number of a graph G and the proper-path-width of G introduced by the authors in [16]. Complexity results are also shown.

The *searching game* was introduced by Parsons [9]. In the searching game, an undirected graph G is considered as a system of tunnels. Initially, all edges of G are contaminated by a gas. An edge is *cleared* by some operations on G. A cleared edge is *recontaminated* if there is a path from an uncleared edge to the cleared edge without any searchers on its vertices or edges.

In the *edge-searching*, the original version of searching game, an edge is cleared by sliding a searcher along the edge. A search is a sequence of operations of placing a searcher on a vertex, deleting a searcher from a vertex, or sliding a searcher along an edge. The object of edge-searching is to clear all edges by a search. We call such a search an *edge-search*. An edge-search is *optimal* if the maximum number of searchers on G at any point is as small as possible. This number is called the *edge-search number* of G, and denoted by $es(G)$. LaPaugh proved that there exists an optimal edge-search without recontamination of cleared edges [6]. Megiddo, Hakimi, Garey, Johnson, and Papadimitriou showed that the problem of computing $es(G)$ is NP-hard for general graphs but can be solved in linear time for trees [7].

The *node-searching*, a slightly different version of searching game, was introduced by Kirousis and Papadimitriou [5]. In the node-searching, an edge is cleared by placing searchers at both its ends simultaneously. A *node-search* is a sequence of operations of placing a searcher on a vertex or deleting a searcher from a vertex so that all edges of G are simultaneously clear after the last stage. A node-search is optimal if the maximum number of searchers on G at any point is as small as possible. This number is called the *node-search number* of G, and denoted by $ns(G)$. Kirousis and Papadimitriou proved the following results: (1) There exists an optimal node-search without recontamination of cleared edges; (2) The problem of computing $ns(G)$ is NP-hard for general graphs; (3) $ns(G) - 1 \leq es(G) \leq ns(G) + 1$ [5].

The path-width of a graph was introduced by Robertson and Seymour [11]. Given a graph G, a sequence $X_1, X_2, \ldots, X_r$ of subsets of the vertex set of G is a *path-decomposition* of G if the following conditions are satisfied: (i) For every edge e of G, some X_i $(1 \leq i \leq r)$ contains both ends of e; (ii) For $1 \leq l \leq m \leq n \leq r$, $X_l \cap X_n \subseteq X_m$. The *path-width* of G, denoted by $pw(G)$, is the minimum value of $k \geq 0$ such that G has a path-decomposition $X_1, X_2, \ldots, X_r$ with $|X_i| \leq k+1$ for $i = 1, 2, \ldots, r$. The unexpected equality $ns(G) = pw(G) + 1$ was mentioned by Möhring [8], and implicitly by Kirousis and Papadimitriou [4]. This provides a linear time algorithm to compute $ns(G)$ for trees [8, 14].

In this paper, we introduce another version of searching game, called the *mixed-searching*, which is a natural common generalization of the edge-searching and node-searching. In the mixed-searching, an edge is cleared by placing searchers at both its ends simultaneously or by sliding a searcher along the edge. A *mixed-search* is a sequence of operations of placing a searcher on a vertex, deleting a searcher from a vertex, or sliding a searcher along an edge so that all edges of G are simultaneously clear after the last stage. A mixed-search is optimal if the maximum number of searchers on G at any point is as small as possible. This number is called the *mixed-search number* of G, and denoted by $ms(G)$. We first show the inequalities $es(G) - 1 \leq ms(G) \leq es(G)$ and $ns(G) - 1 \leq ms(G) \leq ns(G)$. We next prove that there exists an optimal mixed-search without recontamination of cleared edges. This implies that the problem of deciding, given a graph G and an integer k, whether $ms(G) \leq k$ is in NP.

The proper-path-width of a graph was introduced by the authors in [16]. The path-decomposition $X_1, X_2, \ldots, X_r$ of G with $|X_i| \leq k + 1$ $(k \geq 1)$ for any i is called a *proper-path-decomposition* of G if $|X_l \cap X_m \cap X_n| < k$ holds for any X_l, X_m, and X_n none of which is a subset of the others $(1 \leq l < m < n \leq r)$. The *proper-path-width* of G, denoted by $ppw(G)$, is the minimum value of $k \geq 1$ such that G has a proper-path-decomposition $X_1, X_2, \ldots, X_r$ with $|X_i| \leq k + 1$ for any i. We prove that the problem of computing $ppw(G)$ is NP-hard for general graphs but can be solved in linear time for trees. We establish the equality $ms(G) = ppw(G)$ and show that the problem of computing $ms(G)$ is NP-hard for general graphs but can be solved in linear time for

trees.

The rest of the paper is organized as follows: We study the mixed-searching in Section 2. Section 3 concerns the proper-path-width of a graph. We prove a connection between the mixed-searching and proper-path-width in Section 4. Some remarks will be given in Section 5.

2 Mixed-Searching

Graphs we consider are nontrivial and connected, but may have loops and multiple edges unless otherwise specified. Let G be a graph, and $V(G)$ and $E(G)$ denote the vertex set and edge set of G, respectively.

In the *mixed-searching game*, a graph G is considered as a system of tunnels. Initially, all edges are contaminated by a gas. An edge is *cleared* by placing searchers at both its ends simultaneously or by sliding a searcher along the edge. A cleared edge is *recontaminated* if there is a path from an uncleared edge to the cleared edge without any searchers on its vertices or edges.

Definition 1 *A search is a sequence of the following operations:*

(a) placing a new searcher on a vertex,

(b) deleting a searcher from a vertex,

(c) sliding a searcher on a vertex along an incident edge and placing the searcher on the other end,

(d) sliding a searcher on a vertex along an incident edge,

(e) sliding a new searcher along an edge and placing the searcher on its end,

(f) sliding a new searcher along an edge.

The object of mixed-searching game is to clear all edges by a search. We call such a search a *mixed-search*. A mixed-search is *optimal* if the maximum number of searchers on G at any point is as small as possible. This number is called the *mixed-search number* of G, and denoted by $ms(G)$.

We first show a relation to the edge-searching and node-searching.

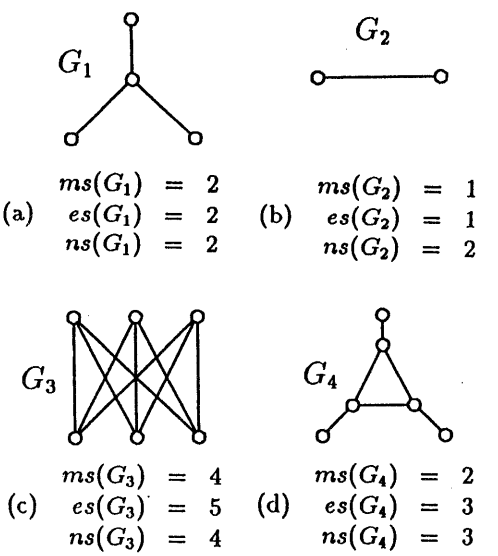
Theorem 1 *For any graph G, $es(G) - 1 \leq ms(G) \leq es(G)$ and $ns(G) - 1 \leq ms(G) \leq ns(G)$.*

Sketch of proof: The edge-search and node-search are special cases of the mixed-search by definition. Thus we have $ms(G) \leq es(G)$ and $ms(G) \leq ns(G)$.

Using at most one more searcher to traverse an edge that is cleared by placing searchers at both its ends, we can convert any mixed-search to an edge-search. Thus $es(G) \leq ms(G) + 1$.

Similarly, using at most one more searcher to clear an edge that is cleared by sliding a searcher along the edge, we can convert any mixed-search to a node-search. Thus $ns(G) \leq ms(G) + 1$. □

We can easily construct examples showing that all four cases are possible (See Fig. 1).



(a) $ms(G_1) = 2$, $es(G_1) = 2$, $ns(G_1) = 2$

(b) $ms(G_2) = 1$, $es(G_2) = 1$, $ns(G_2) = 2$

(c) $ms(G_3) = 4$, $es(G_3) = 5$, $ns(G_3) = 4$

(d) $ms(G_4) = 2$, $es(G_4) = 3$, $ns(G_4) = 3$

Figure 1: Search numbers of graphs

Kirousis and Papadimitriou proved that recontamination does not help in node-searching.

Theorem A ([5]) *For any graph G, there exists an optimal node-search without recontamination of cleared edges.*

Corollary A ([5]) *For any graph G, there exists an optimal node-search without recontamination of cleared edges satisfying the following two conditions:*

(i) every vertex is visited exactly once by a searcher,

(ii) every searcher is deleted immediately after all the edges incident to it have been cleared (ties are broken arbitrarily).

We shall prove now that recontamination does not help even in mixed-searching.

Theorem 2 *For any graph G, there exists an optimal mixed-search without recontamination of cleared edges.*

Sketch of proof: Let G be a graph, and G_m be the graph obtained from G by subdividing every edge of G. We call the vertices of $V(G) \subseteq V(G_m)$ original vertices of G_m, and the vertices of $V(G_m) - V(G)$ middle vertices of G_m. We shall prove that $ms(G) = ns(G_m) - 1$ and an optimal mixed-search of G without recontamination of cleared edges can be obtained from an optimal node-search of G_m of the form described in Corollary A.

It is almost obvious that $ns(G_m) \leq ms(G) + 1$ since by one extra searcher we can carry out node-search of G_m, simulating a mixed-search of G.

Conversely, we can carry out a mixed-search of G, simulating an optimal node-search of G_m of the form described in Corollary A as follows. We can assume that a searcher is placed on a middle vertex of G_m after a searcher is placed on one of its neighbors. The rules for the simulation are the following:

- When a searcher is placed on an original vertex v of G_m, a searcher is placed on v of G if v has no searcher.
- When a searcher is deleted from an original vertex v of G_m, delete the searcher from v of G if v has a searcher.
- When a searcher is placed on a middle vertex of G_m, clear the corresponding edge (u, v) of G, if it is contaminated, as follows: We can assume that u has a searcher and v does not have a searcher in G. If no recontamination is caused, clear $(u, v) \in E(G)$ by sliding a searcher on u along (u, v), and place it on v. Otherwise, clear $(u, v) \in E(G)$ by placing a new searcher on v.
- Do nothing in any other case.

It is not difficult to see that the simulation based on the rules above defines a mixed-search of G without recontamination of cleared edges, and the number of searchers used on G is at most $ns(G_m)$. We will show that $ns(G_m) - 1$ searchers are enough. Suppose that the number of searchers on G_m raises to $ns(G_m)$ when a searcher is placed on v of G_m. The next operation on G_m must be deleting a searcher from a vertex. A searcher on v or a vertex adjacent to v must be deleted in the next operation by the assumption that the node-search is of the form described in Corollary A. There are the following four cases to be considered:

(1) v is an original vertex of G_m and the searcher on v is deleted in the next operation.

(2) v is an original vertex of G_m and a searcher on a vertex adjacent to v is deleted in the next operation.

(3) v is a middle vertex of G_m and the searcher on v is deleted in the next operation.

(4) v is a middle vertex of G_m and a searcher on a vertex adjacent to v is deleted in the next operation.

In the case of (1), all edges of G incident to v have been cleared before placing a searcher on v of G_m since all middle vertices of G_m adjacent to v have accepted searchers. Thus placing a new searcher on v of G is redundant. Similarly, we can show that no new searcher on v or a vertex adjacent to v is necessary for the other three cases. Thus we have $ms(G) \leq ns(G_m) - 1$. □

It should be noted that Theorem 2 implies that the problem of deciding, given a graph G and an integer k, whether $ms(G) \leq k$ is in NP.

We obtain the following corollary from Theorem 2.

Corollary 1 *For any graph G, there exists an optimal mixed-search without recontamination of cleared edges such that it is a sequence of operations (a), (b), or (c) of Definition 1, and satisfying the following two conditions:*

(i) every vertex is visited exactly once by a searcher,

(ii) every edge is visited at most once by a searcher.

3 Proper-Path-Width

Definition 2 ([16]) *A sequence $X_1, X_2, \ldots, X_r$ of subsets of $V(G)$ is a* path-decomposition *of G if the following two conditions are satisfied:*

(i) *For every edge $e \in E(G)$, some X_i $(1 \leq i \leq r)$ contains both ends of e.*

(ii) *For $1 \leq l \leq m \leq n \leq r$, $X_l \cap X_n \subseteq X_m$.*

A path-decomposition $X_1, X_2, \ldots, X_r$ of G with $|X_i| \leq k + 1$ $(k \geq 1)$ for any i is called a proper-path-decomposition *of G if $|X_l \cap X_m \cap X_n| < k$ holds for any X_l, X_m, and X_n none of which is a subset of the others $(1 \leq l < m < n \leq r)$. The* proper-path-width *of G, denoted by $ppw(G)$, is the minimum value of $k \geq 1$ such that G has a proper-path-decomposition $X_1, X_2, \ldots, X_r$ with $|X_i| \leq k + 1$ for any i.*

Notice that a path-decomposition such that $X_i \not\subseteq X_j$ for any distinct i and j is a proper-path-decomposition if $|X_l \cap X_n| < k$ for any X_l and X_n $(l + 2 \leq n)$. Notice also that $pw(G) \leq ppw(G) \leq pw(G) + 1$ for any graph G.

A graph obtained from connected graphs H_1, H_2, and H_3 by the following construction is called a *star-composition* of H_1, H_2, and H_3: (i) Choose a vertex $v_i \in V(H_i)$ for $i = 1, 2$, and 3; (ii) Let v be a new vertex not in H_1, H_2, or H_3; (iii) Connect v to v_i by an edge (v, v_i) for $i = 1, 2$, and 3. We define the family Ω_k of trees recursively as follows: (i) $\Omega_1 = \{K_{1,3}\}$; (ii) If Ω_k is defined, a tree T is in Ω_{k+1} if and only if T is a star-composition of (not necessarily distinct) three trees in Ω_k. A graph H is a *minor* of G if H is isomorphic to a graph obtained from a subgraph of G by contracting edges.

The following theorems were proved in [16].

Theorem B ([16]) *For any tree T and an integer k ($k \geq 1$), $ppw(T) \leq k$ if and only if T contains no tree in Ω_k as a minor.*

Corollary B ([16]) *(1) The number of vertices of a tree in Ω_k is $\frac{3^{k+1}-1}{2}$ ($k \geq 1$).*
(2) $|\Omega_k| \geq k!^2$ ($k \geq 1$).

Theorem C ([16]) *For any tree T and an integer k ($k \geq 1$), $ppw(T) \geq k+1$ if and only if T has a vertex v such that T/v has at least three connected components with proper-path-width k or more, where T/v is the graph obtained from T by deleting v.*

Theorem C was used to prove Theorem B.

A *k-clique* of a graph G is a complete subgraph of G with k vertices. For a positive integer k, *k-trees* are defined recursively as follows: (i) The complete graph with k vertices is a k-tree; (ii) Given a k-tree Q with n vertices ($n \geq k$), a graph obtained from Q by adding a new vertex adjacent to the vertices of a k-clique of Q is a k-tree with $n+1$ vertices. A k-tree Q is called a *k-path* [10] or *k-chordal path* [1] if $|V(Q)| \leq k+1$ or Q has exactly two vertices of degree k. A *partial k-path* is a subgraph of a k-path.

Before proving Theorem 3 below, we need the following lemma.

Lemma 1 *For any graph G with $ppw(G) = k$, there exists a proper-path-decomposition $X_1, X_2, \ldots, X_r$ of G satisfying the following two conditions:*
(i) $|X_i| = k+1$ *for any i,*
(ii) $|X_i \cap X_{i+1}| = k$ *for $1 \leq i \leq r-1$.*

Sketch of proof: We will show that a proper-path-decomposition of G of the form described in this lemma can be obtained from any proper-path-decomposition $X = (X_1, X_2, \ldots, X_s)$ with $|X_i| \leq k+1$ for any i by the operations of deleting X_i from the sequence, adding a vertex to X_i, or inserting a subset of $V(G)$ between X_i and X_{i+1}. □

Theorem 3 *For any simple graph G and an integer k ($k \geq 1$), $ppw(G) \leq k$ if and only if G is a partial k-path.*

Sketch of proof: Suppose that $ppw(G) = h \leq k$ and $X_1, X_2, \ldots, X_r$ is a proper-path-decomposition of G of the form described in Lemma 1. We construct a h-path H as follows:
(i) Let v_1 be a vertex in $X_1 \cap X_2$. Define that Q_1 is the complete graph on $X_1 - \{v_1\}$.
(ii) Given Q_1 and the vertex v_1, define that Q_2 is the h-path obtain from Q_1 by adding v_1 and the edges connecting v_1 and the vertices in $X_1 - \{v_1\}$.
(iii) Given Q_i and the vertex $v_i \in X_i - X_{i-1}$ ($2 \leq i \leq r$), define that Q_{i+1} is the h-path obtained from Q_i by adding v_i and the edges connecting v_i and the vertices in $X_i \cap X_{i-1}$.
(iv) Define $H = Q_{r+1}$.
From Lemma 1 and the definition of proper-path-decomposition, v_i is uniquely determined and $v_{i-1} \in X_i \cap X_{i-1}$ for $2 \leq i \leq r$. Furthermore, we have $V(H) = V(G)$ and $E(H) \supseteq E(G)$. Thus G is a partial h-path, and so a partial k-path.

Conversely, suppose, without loss of generality, that G is a partial k-path with n vertices and H is a k-path such that $V(H) = V(G)$ and $E(H) \supseteq E(G)$. It is well known that H can be obtained as follows:
(i) Define that $Q_1 = R_1$ is the complete graph with k vertices.
(ii) Given Q_i, R_i, and a new vertex v_i, define that Q_{i+1} is the k-path obtained from Q_i by adding v_i and the edges connecting v_i and the vertices of R_i, and R_{i+1} is a k-clique of Q_{i+1} that contains v_i.
(iii) Define $H = Q_{n-k+1}$.
We define $X_i = V(R_i) \cup \{v_i\}$ for $1 \leq i \leq n-k$. It is not difficult to see that the sequence $X_1, X_2, \ldots, X_{n-k}$ is a path-decomposition of H with $|X_i| = k+1$ for any i. Since $X_{i+1} - X_{i-1} = \{v_i, v_{i+1}\}$, $|X_{i-1} \cap X_{i+1}| = k-1$ for $1 < i < n-k$. It follows that $|X_a \cap X_b \cap X_c| < k$ for any a, b, and c ($1 \leq a < b < c \leq n-k$). Thus the sequence $X_1, X_2, \ldots, X_{n-k}$ is a proper-path-decomposition

of H with $|X_i| = k+1$ for any i and we have $ppw(H)$ is at most k, and so $ppw(G)$ is at most k. □

Arnborg, Corneil, and Proskurowski proved that the problem of deciding, given a graph G and an integer k, whether G is a partial k-path is NP-complete [1]. Thus we immediately have the following by Theorem 3.

Theorem 4 *The problem of computing $ppw(G)$ is NP-hard.*

It should be noted that Theorem B together with Robertson and Seymour's results on graph minors [12, 13] provides $O(n^2)$ algorithm to decide, given a tree T on n vertices, whether $ppw(T) \leq k$ for any fixed integer k, although it is not practical even if we could solve MINOR CONTAINMENT (see [3], for example) efficiently, because $|\Omega_k| \geq k!^2$ as is shown in Corollary B(2).

We show a practical algorithm to compute $ppw(T)$ for trees T based on Theorem C, and prove the following.

Theorem 5 *For any tree T, the problem of computing $ppw(T)$ is solvable in linear time.*

Sketch of proof: Our algorithm to compute $ppw(T)$ is shown in Fig. 2. The outline of the algorithm is as follows.

For any tree T with a vertex $v \in V(T)$ as the root, we define the path-vector $\overline{pv}(v,T) = (p_v, c_v, S_v)$. p_v describes the proper-path-width of T. c_v and S_v describe the condition of T as follows: If there exists $u \in V(T) - \{v\}$ such that T/u has two connected components with proper-path-width p_v and without v, then $c_v = 3$ and S_v is the path-vector of the connected component of T/u containing v; Otherwise, c_v is the number of the connected components of T/v with proper-path-width p_v and $S_v = nul$. Suppose that a tree T rooted at s is obtained from tree T_1 rooted at s and tree T_2 rooted at t by joining an edge (s,t). Based on Theorem C, the Procedure MERGE recursively calculating the path-vector $\overline{pv}(s,T)$ of T from the path-vector $\overline{pv}(s,T_1) = (p_s, c_s, S_s)$ of T_1 and the path-vector $\overline{pv}(t,T_2) = (p_t, c_t, S_t)$ of T_2.

The algorithm computes the path-vector of T rooted at r from the path-vectors of isolated vertices obtained from T by deleting all edges in T. The Procedure DFS computes the path-vector of a maximal subtree of T rooted at s from the path-vectors of maximal subtrees rooted at children of s in T by using the Procedure MERGE. The Procedure MAIN obtains the proper-path-width of T from the path-vector of a maximal subtree rooted at r by the Procedure DFS.

The Procedure MERGE calculates the path-vector of the join of two subtrees T_1 and T_2 in $O(p)$ where $p = max(ppw(T_1), ppw(T_2))$. From Corollary B(1), we have $p = O(\log n)$ where $n = |V(T)|$. Since the Procedure MERGE is called at most once for any vertex, the time complexity of the algorithm is essentially $O(n \log n)$. By a careful use of pointers, the Procedure MERGE calculates the path-vector in $O(q)$ where $q = min(ppw(T_1), ppw(T_2))$. Thus the time complexity of the algorithm is $O(n)$. □

We will also mention that for any tree T with n vertices we can construct a proper-path-decomposition $X_1, X_2, \ldots, X_r$ with $|X_i| \leq ppw(T)+1$ for any i in $O(n \log n)$ time.

4 Mixed-Searching and Proper-Path-Width

We establish the following equality for simple graphs in this section.

Theorem 6 *For any simple graph G, $ms(G) = ppw(G)$.*

Sketch of proof: Suppose that $ppw(G) = k$ and $X_1, X_2, \ldots, X_r$ is a proper-path-decomposition of G of the form described in Lemma 1. We can obtain a mixed-search with k searchers as follows:

- Step 1: Let v_1 be a vertex in $X_1 \cap X_2$. Place the k searchers on the vertices of $X_1 - \{v_1\}$.
- Step 2: Slide a searcher on $u_1 \in X_1 - X_2$ toward v_1 and place it on v_1 if $(u_1, v_1) \in E(G)$. Otherwise, delete a searcher from u_1 and place a searcher on v_1.
- Step 3: Let $i = 2$.
- Step 4: Slide a searcher on $u_i \in X_i - X_{i+1}$ toward $v_i \in X_i - X_{i-1}$ and place it on v_i if $(u_i, v_i) \in E(G)$. Otherwise, delete a searcher from u_i and place a searcher on v_i.
- Step 5: Let $i = i+1$ and repeat Step 4 while $i \leq r-1$.
- Step 6: Let u_r be a vertex in $X_{r-1} \cap X_r$. Slide a searcher on u_r toward $v_r \in X_r - X_{r-1}$ and place it on v_r if $(u_r, v_r) \in E(G)$.

```
Procedure MERGE( p̄v(s,T1), p̄v(t,T2) )
{ input:p̄v(s,T1), p̄v(t,T2) }
{ output:p̄v(s,T) }
{ p̄v(tmp) = (p_tmp, c_tmp, s_tmp) }

  1. if p_s > p_t then
          if c_s ≤ 2 then
               p̄v(s,T) = p̄v(s,T1);
          else
               p̄v(tmp) = MERGE( S_s, p̄v(t,T2) );
               if p_s = p_tmp then
                    p̄v(s,T) = (p_s + 1, 0, nul);
               else
                    p̄v(s,T) = (p_s, 3, p̄v(tmp));
          endif
     endif

  2. if p_s = p_t then
          if c_s ≥ 2 or c_t ≥ 2 then
               p̄v(s,T) = (p_s + 1, 0, nul);
          else if c_s = 0 then
               p̄v(s,T) = (p_s, 1, nul);
          else if c_s = 1 then
               p̄v(s,T) = (p_s, 2, nul);
     endif

  3. if p_s < p_t then
          if c_t ≤ 1 then
               p̄v(s,T) = (p_t, 1, nul);
          else if c_t = 2 then
               p̄v(s,T) = (p_t, 3, p̄v(s,T1));
          else if c_t = 3 then
               p̄v(tmp) = MERGE( p̄v(s,T1), S_t );
               if p_t = p_tmp then
                    p̄v(s,T) = (p_t + 1, 0, nul);
               else
                    p̄v(s,T) = (p_t, 3, p̄v(tmp));
          endif
     endif

  4. return( p̄v(s,T) );
     end

Procedure DFS( s )
{ input: a vertex s }
{ output: the path-vector of the maximal subtree rooted at s }

  1. p̄v(s) = (1, 0, nul);

  2. for all children t of s in T do
          p̄v(t) = DFS( t );
          p̄v(s) = MERGE( p̄v(s), p̄v(t) );
     endfor

  3. return( p̄v(s) );
     end

Procedure MAIN( T, r )
{ input: a tree T with a vertex r as the root }
{ output: proper-path-width ppw(T) }

  1. (p_r, c_r, S_r) = DFS( r );

  2. return( p_r );
     end
```

Figure 2: The algorithm to compute $ppw(T)$

Otherwise, delete a searcher from u_r and place a searcher on v_r.

From Lemma 1, both u_i $(1 \leq i \leq r-1)$ and v_i $(2 \leq i \leq r)$ are uniquely determined. From Lemma 1 and the definition of proper-path-decomposition, $u_i \in X_{i-1}$ $(2 \leq i \leq r)$ and $v_i \in X_{i+1}$ $(1 \leq i \leq r-1)$. Notice that before sliding or deleting a searcher from u_i in Steps 4 or 6, the searchers are on the vertices $X_i \cap X_{i-1}$ and after placing a searcher on v_i in Steps 2 or 4, the searchers are on the vertices $X_i \cap X_{i+1}$. It is easy to see that all edges incident to u_i except for (u_i, v_i) have been cleared when the searcher on u_i is deleted in Steps 2, 4, or 6. Since G is a simple graph, there is at most one edge connecting u_i and v_i. Thus the edge (u_i, v_i) for $1 \leq i \leq r$, if exists, is cleared by sliding a searcher along it and the other edges of G are cleared by placing searchers at both its ends simultaneously. Thus the search above is indeed a mixed-search with at most $ppw(G)$ searchers, and we have $ms(G) \leq ppw(G)$.

Conversely, suppose that we have a mixed-search with k searchers of the form described in Corollary 1. For the i-th operation of the mixed-search, we define X_i as follows:

(1) When a searcher is placed on (deleted from) a vertex, we define X_i as the set of vertices having searchers.

(2) When a searcher is slid from u to v, we define X_i as the set of u, v, and the vertices having the other searchers.

It is not difficult to see that the sequence $X_1, X_2, \ldots, X_r$ thus obtained is a path-decomposition of G with $|X_i| \leq k+1$ for any i. If X_i is defined by (1) then $|X_i| \leq k$. If X_i is defined by (2) then $u \notin X_j$ for any $j > i$ and $v \notin X_j$ for any $j < i$. Thus $|X_l \cap X_m \cap X_n| < k$ holds for any X_l, X_m, and X_n none of which is a subset of the others $(1 \leq l < m < n \leq r)$. Therefore, the sequence $X_1, X_2, \ldots, X_r$ is a proper-path-decomposition of G with $|X_i| \leq k+1$ for any i. Thus we have $ppw(G) \leq ms(G)$. □

It should be noted that Theorems B and 6 provide a structural characterization of trees T with $ms(T) \leq k$.

From Theorems 4, 5, and 6, we have the following complexity results on $ms(G)$.

Theorem 7 *The problem of computing $ms(G)$ is NP-hard for general graphs but can be solved in linear time for trees.*

5 Concluding Remarks

Notice that Theorem 6 does not hold for multiple graphs. If G is the graph consisting of three parallel edges, $ppw(G) = 1$, and $ms(G) = 2$. However we can prove that $ppw(G) \leq ms(G) \leq ppw(G) + 1$ for any multiple graph G.

We should mention the relation of mixed-searching with the virus-searching introduced by Shinoda [15]. In virus-searching, initially, all vertices are contaminated by a virus. A vertex is cleared by placing a searcher on it. A cleared vertex is recontaminated if there is a path from an uncleared vertex to the cleared vertex without any searchers on its vertices or edges. A search is a sequence of operations of placing a searcher on a vertex, deleting a searcher from a vertex, or sliding a searcher along an edge. The object of virus-searching is to clear all vertices by a search. We call such a search a *virus-search*. A virus-search is optimal if the maximum number of searchers on G at any point is as small as possible. This number is called the *virus-search number* of G, and denoted by $vs(G)$. Any virus-search S can be considered as a mixed-search, and vice versa. It is easy to see that an edge (u, v) is cleared by S as a mixed-search if and only if both its ends u and v are cleared by S as a virus-search. Thus $vs(G) = ms(G)$ for any graph G.

We learned recently that Bienstock and Seymour introduced independently the mixed-searching game [2]. They prove directly that there exists an optimal mixed-search without recontamination of cleared edges. They also mentioned that monotonicity result for mixed-searching implies monotonicity for both edge- and node-searching.

References

[1] S. Arnborg, D. G. Corneil, and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM J. Alg. Disc. Meth.*, 8(2), pp. 277–284, April 1987.

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

[3] D. S. Johnson, The NP-completeness column: an ongoing guide, *Journal of Algorithms*, 8, pp. 285–303, 1987.

[4] L. M. Kirousis and C. H. Papadimitriou, Interval graphs and searching, *Discrete Mathematics*, 55, pp. 181–184, 1985.

[5] L. M. Kirousis and C. H. Papadimitriou, Searching and pebbling, *Theoretical Computer Science*, 47, pp. 205–218, 1986.

[6] A. LaPaugh, *Recontamination does not help to search a graph*, Technical Report, Electrical Engineering and Computer Science Department, Princeton University, 1983.

[7] M. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, The complexity of searching a graph, *Journal of the Association for Computing Machinery*, 35(1), pp. 18–44, January 1988.

[8] R. H. Möhring, Graph problems related to gate matrix layout and PLA folding, in G. Tinhofer, E. Mayr, H. Noltemeier, and M. Syslo, editors, *Computational Graph Theory*, pp. 17–51, Springer-Verlag, Wien New York, 1990.

[9] T. D. Parsons, Pursuit-evasion in a graph, in Y. Alavi and D. Lich, editors, *Theory and Applications of Graphs*, pp. 426–441, Springer-Verlag, Berlin, 1976.

[10] A. Proskurowski, Separating subgraphs in k-trees: cables and caterpillars, *Discrete Mathematics*, 49, pp. 275–285, 1984.

[11] N. Robertson and P. D. Seymour, Graph minors. I. Excluding a forest, *Journal of Combinatorial Theory*, Series B(35), pp. 39–61, 1983.

[12] N. Robertson and P. D. Seymour, Graph minors. XIII. The disjoint paths problem, 1986, preprint.

[13] N. Robertson and P. D. Seymour, Graph minors. XVI. Wagner's conjecture, 1987, preprint.

[14] P. Scheffler, A linear algorithm for the pathwidth of trees, in R. Bodendiek and R. Henn, editors, *Topics in Combinatorics and Graph Theory*, pp. 613–620, Physica-Verlag, Heidelberg, 1990.

[15] S. Shinoda, On some problems of graphs — including Kajitani's conjecture and its solution —, in *Proc. of 2nd Karuizawa Workshop on Circuits and Systems*, pp. 414–418, 1989, in Japanese.

[16] A. Takahashi, S. Ueno, and Y. Kajitani, Minimal acyclic forbidden minors for the family of graphs with bounded path-width, to appear in *Annals of discrete mathematics* (*Proceedings of 2nd Japan conference on graph theorey and combinatorics*, 1990). Also: *SIGAL* 91-19-3, IPSJ, 1991.