ELSEVIER

Theoretical Computer Science 137 (1995) 253–268

Theoretical Computer Science

Mathematical Games

Mixed searching and proper-path-width☆

Atsushi Takahashi[a,*], Shuichi Ueno[a], Yoji Kajitani[a,b]

[a]*Department of Electrical and Electronic Engineering, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152, Japan*

[b]*School of Information Science, Japan Advanced Institute of Science and Technology, East 15 Asahidai, Tatsunokuchi-cho, Ishikawa 923-12, Japan*

Received July 1992; revised April 1994
Communicated by E.R. Berlekamp

Abstract

This paper considers a *mixed search game*, which is a natural generalization of edge-search and node-search 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. We also prove complexity results.

1. Introduction

This paper considers a new version of search game, called mixed searching, which is a natural generalization of edge searching and node searching extensively studied so far. We establish a relationship between the mixed search number of a simple graph G and the proper-path-width of G introduced by the authors in[18]. We also prove complexity results.

Search games were first introduced by Breisch [5] and Parsons [12]. An undirected graph G is thought of 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 *edge searching*, the original search game variant, an edge is cleared by sliding a searcher along the edge. A search is a sequence of operations of placing a searcher on

☆ A preliminary version of this paper appeared in [20].

* Corresponding author. E-mail: atushi@ss.titech.ac.jp.

SSDI 0304-3975(94)00160-X

a vertex, deleting a searcher from a vertex, or sliding a searcher along an edge. The object of such an *edge search* is to clear all edges by a search. An edge search is *optimal* if the maximum number of searchers on G at any operation is minimum over all edge searches of G. This number is called the *edge search number* of G, and is denoted by $es(G)$. La Paugh [9] proved that there exists an optimal edge search without recontamination of cleared edges. This means that the problem of deciding whether $es(G) \leqslant k$ is in NP. Megiddo et al. [10] showed that the problem of computing $es(G)$ is NP-hard for general graphs but can be solved in linear time for trees.

Another variant called *node searching* was introduced by Kirousis and Papadimitriou [8]. In 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 operation is minimum over all node searches of G. This number is called the *node search number* of G, and is 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) $ns(G)-1 \leqslant es(G) \leqslant ns(G)+1$ [8].

The *path-width* of a graph G, denoted by $pw(G)$, was introduced by Robertson and Seymour [13]. (The definition of path-width is given in Definition 1.) The unexpected equality $ns(G)=pw(G)+1$ was mentioned by Möhring [11], and implied by Kirousis and Papadimitriou [7]. This provides a linear time algorithm to compute $ns(G)$ for trees [11, 16].

Mixed searching, a natural generalization of edge searching and node searching, was introduced by Bienstock and Seymour [2], and independently by the authors [19]. In 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 operation is minimum over all mixed searches of G. This number is called the *mixed search number* of G, and is denoted by $ms(G)$. Bienstock and Seymour [2] and the authors [19] independently proved that there exists an optimal mixed search without recontamination of cleared edges. Bienstock and Seymour characterized the mixed search number of a graph with minimum degree at least two by means of the concept of crusade, which is a sequence of sets of edges.

The *proper-path-width* of a graph G, denoted by $ppw(G)$, was introduced by the authors in [18]. (The definition of proper-path-width is given in Definition 1) We prove in Section 2 that the problem of computing $ppw(G)$ is NP-hard for general graphs but can be solved in linear time for trees. In Section 3, we characterize the mixed search number of a simple graph by means of the proper-path-width. That is, we establish the equality $ms(G)=ppw(G)$, so the problem of computing $ms(G)$ is also NP-hard for general graphs but can be solved in linear time for trees.

2. Proper-Path-Width

Graphs we consider have at least one edge and 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.

Definition 1 (*Takahashi et al.* [18]). Let $\mathscr{X}=(X_1, X_2, \ldots, X_r)$ be a sequence of subsets of $V(G)$. The width of $\mathscr{X}$ is $\max_{1\leqslant i\leqslant r}|X_i|-1$. $\mathscr{X}$ is called a *proper-path-decomposition* of G if the following conditions are satisfied:

(i) for any distinct i and j, $X_i \not\subseteq X_j$;
(ii) $\bigcup_{i=1}^{r} X_i = V(G)$;
(iii) for any edge $(u, v)\in E(G)$, there exists an i such that $u, v\in X_i$;
(iv) for all l, m and n with $1\leqslant l\leqslant m\leqslant n\leqslant r$, $X_l\cap X_n\subseteq X_m$;
(v) for all l, m and n with $1\leqslant l<m<n\leqslant r$, $|X_l\cap X_n|\leqslant |X_m|-2$.

The *proper-path-width* of G, denoted by $ppw(G)$, is the minimum width over all proper-path-decompositions of G. If $\mathscr{X}$ satisfies (i)–(iv), $\mathscr{X}$ is called a path-decomposition of G. The *path-width* of G, denoted by $pw(G)$, is the minimum width over all path-decompositions of G. A (proper-)path-decomposition with width k is called a k-(proper-)path-decomposition.

We first show that the path-width and proper-path-width of a graph may differ by at most one.

Theorem 1. *For any graph G, $pw(G)\leqslant ppw(G)\leqslant pw(G)+1$.*

Proof. The first inequality follows from the definition.

To prove the second inequality, we show that a proper-path-decomposition of G with width at most $k+1$ can be obtained from a k-path-decomposition of G. Let $(X_1. X_2, \ldots, X_r)$ be a k-path-decomposition of G. If $|X_l\cap X_n|=|X_i|-1$ for some l and n $(1\leqslant l<i<n\leqslant r)$, let $X'_i=X_{i-1}\cup X_i$; otherwise let $X'_i=X_i$. It is easy to see that the sequence $\mathscr{X}'=(X'_1, X'_2, \ldots, X'_r)$ satisfies conditions (ii)–(iv) in Definition 1. If $X'_i=X_i$ and $X'_{i+1}=X_i\cup X_{i+1}$ $(1\leqslant i<r)$ then $X'_i\subseteq X'_{i+1}$, otherwise $X'_i\not\subseteq X'_{i+1}$. Let $\mathscr{X}$ be the sequence obtained from $\mathscr{X}'$ by deleting every X'_i such that $X'_i\subseteq X'_{i+1}$. We show that $\mathscr{X}$ is a proper-path-decomposition of G with width at most $k+1$. It is easy to see that $\mathscr{X}$ also satisfies conditions (ii)–(iv) in Definition 1. To verify condition (i), assume that $X'_i\subseteq X'_j$ for some distinct i and j. Since $X_i\not\subseteq X_j$, $X'_j=X_{j-1}\cup X_j$. If $i>j$ then $X'_i\cap X_{j-1}\subseteq(X_{i-1}\cup X_i)\cap X_{j-1}\subseteq X_j$ by condition (iv) in Definition 1. Since $X'_i\cap X_j\subseteq X_j$, we have $X_i\subseteq X'_i=X'_i\cap X'_j\subseteq X_j$. However, this is contradicting to $X_i\not\subseteq X_j$. Similarly, if $i<j$ then $X_i\subseteq X'_i\subseteq X_{j-1}$, and we have $i=j-1$. Moreover $X'_i=X_i$, for otherwise $X_{i-1}\subseteq X_{j-1}$. However, $\mathscr{X}$ does not contain such X'_i. Hence $\mathscr{X}$ satisfies condition (i) in Definition 1. To verify condition (v), first, assume that $X'_i=X_{i-1}\cup X_i$ $(1<i<r)$. By condition (i) in Definition 1, $|X'_i|=|X_{i-1}\cup X_i|\geqslant|X_i|+1$. Since $X'_l\cap X'_n\subseteq X_l\cap X_{n-1}$ $(1\leqslant l<n\leqslant r)$ by condition (iv) in Definition 1,

$|X'_l \cap X'_n| \leqslant |X_l \cap X_{n-1}| \leqslant |X_i| - 1$ for any l and n $(1 \leqslant l < i < n \leqslant r)$ by condition (i) in Definition 1. Hence, $|X'_l \cap X'_n| \leqslant |X'_i| - 2$ for any l and n $(1 \leqslant l < i < n \leqslant r)$. Next, assume that $X'_i = X_i$ $(1 < i < r)$. Notice that $|X_{i-1} \cap X_{i+1}| \leqslant |X_i| - 2$ by the definition of X'_i. By the construction of $\mathcal{X}$, we have $X'_{i+1} = X_{i+1}$. Since $X'_l \cap X'_n \subseteq X'_{i-1} \cap X'_{i+1} \subseteq X_{i-1} \cap X_{i+1}$, we have $|X'_l \cap X'_n| \leqslant |X_i| - 2 = |X'_i| - 2$ for any l and n $(1 \leqslant l < i < n \leqslant r)$. Thus, $\mathcal{X}$ satisfies condition (v) in Definition 1. Finally, we show that the width of $\mathcal{X}$ is at most $k+1$. If $X'_i = X_{i-1} \cup X_i$ then $|X_l \cap X_n| = |X_i| - 1$ for some l and n $(1 \leqslant l < i < n \leqslant r)$ by the definition of X'_i. Since $X_l \cap X_n \subseteq X_{i-1} \cap X_i$ by condition (iv) in Definition 1, we have $X_l \cap X_n = X_{i-1} \cap X_i$, for otherwise, $|X_{i-1} \cap X_i| \geqslant |X_i|$ contradicting to condition (i) in Definition 1. Thus, $|X_i - X_{i-1}| = |X_i - (X_l \cap X_n)| = 1$, and $|X'_i| = |X_{i-1} \cup X_i| = |X_{i-1}| + 1 \leqslant k+2$. If $X'_i = X_i$ then $|X'_i| = |X_i| \leqslant k+1$. Hence, the width of $\mathcal{X}$ is at most $k+1$. □

As an example, path-decomposition and proper-path-decomposition of the graph shown in Fig. 1(a) are shown in Fig. 1(b) and (c), respectively.

It is not difficult to see the following lemma.

Lemma 1. (1) *$\mathcal{X}$ satisfies condition* (iv) *in Definition* 1 *if and only if each vertex of G appears in consecutive X_i's.*

(2) *A path-decomposition $\mathcal{X}$ satisfies condition* (v) *in Definition* 1 *if and only if $|X_{i-1} \cap X_{i+1}| \leqslant |X_i| - 2$ holds for any i with $1 < i < r$.*

A k-(proper-)path-decomposition $(X_1, X_2, \ldots, X_r)$ is said to be *full* if $|X_i| = k+1$ $(1 \leqslant i \leqslant r)$ and $|X_j \cap X_{j+1}| = k$ $(1 \leqslant j \leqslant r-1)$. An example of full proper-path-decomposition is shown in Fig. 1(d).

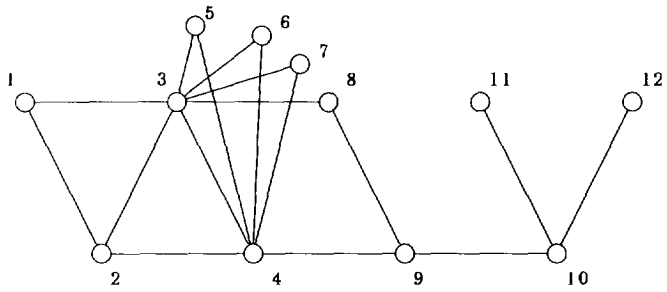
Lemma 2. *If a graph G has a k-path-decomposition $\mathcal{X} = (X_1, X_2, \ldots, X_r)$ such that*

$$(*) \qquad |X_{i-1} \cap X_{i+1}| \leqslant k-1 \quad (1 < i < r),$$

then G has a full k-proper-path-decomposition.

Proof. Let $\mathcal{X} = (X_1, X_2, \ldots, X_r)$ be a k-path-decomposition of G satisfying $(*)$ such that $\sum_{i=1}^{r} (|X'_i| - k)$ is maximum. We shall show that $\mathcal{X}$ is a full k-proper-path-decomposition of G. In the following, $X_j = \emptyset$ if $j \leqslant 0$ or $j > r$.

Assume that $|X_i| \leqslant k$ for some i $(2 \leqslant i \leqslant r)$. If $|X_{i-2} \cap X_i| = k-1$, let $v \in X_{i-1} - X_{i-2}$. Notice that $v \notin X_i$, for otherwise $|X_{i-1} \cap X_i| \geqslant k$ since $X_{i-2} \cap X_i \subseteq X_{i-1} \cap X_i$, and $X_{i-1} \supseteq X_i$, contradicting to condition (i) in Definition 1. If $|X_{i-2} \cap X_i| < k-1$, let $v \in X_{i-1} - X_i$. In either case, we have $v \notin X_i$ and $|X_{i-2} \cap (X_i \cup \{v\})| \leqslant k-1$. By Lemma 1(1), $v \notin X_{i+2}$, and so $|(X_i \cup \{v\}) \cap X_{i+2}| \leqslant k-1$. Thus, the sequence $\mathcal{X}' = (X_1, X_2, \ldots, X_{i-1}, X_i \cup \{v\}, X_{i+1}, \ldots, X_r)$ satisfies condition $(*)$ and conditions (ii)–(iv) in Definition 1. To verify condition (i), assume that $X_j \subseteq X_i \cup \{v\}$ for some j $(\neq i)$. Since $v \notin \bigcup_{p=i+1}^{r} X_p$, $j < i$. Thus $j = i-1$ since $X_j = X_j \cap (X_i \cup \{v\}) \subseteq X_{i-1}$. Therefore, $(X_1, X_2, \ldots, X_{i-2}, X_i \cup \{v\}, X_{i+1}, \ldots, X_r)$ is a k-path-decomposition of G satisfy-



(a) G

$$\left(\begin{matrix}\widehat{1}\\2\\3\end{matrix},\ \begin{matrix}\widehat{2}\\3\\4\end{matrix},\ \begin{matrix}\widehat{3}\\4\\5\end{matrix},\ \begin{matrix}\widehat{3}\\4\\6\end{matrix},\ \begin{matrix}\widehat{3}\\4\\7\end{matrix},\ \begin{matrix}\widehat{3}\\4\\8\end{matrix},\ \begin{matrix}\widehat{4}\\8\\9\end{matrix},\ \begin{matrix}\widehat{9}\\10\end{matrix},\ \begin{matrix}\widehat{10}\\11\end{matrix},\ \begin{matrix}\widehat{10}\\12\end{matrix}\right)$$

(b) A path-decomposition of G

$$\left(\begin{matrix}\widehat{1}\\2\\3\end{matrix},\ \begin{matrix}\widehat{2}\\3\\4\end{matrix},\ \begin{matrix}\widehat{3}\\4\\5\end{matrix},\ \begin{matrix}\widehat{3}\\4\\6\end{matrix},\ \begin{matrix}\widehat{3}\\4\\7\end{matrix},\ \begin{matrix}\widehat{4}\\8\\9\end{matrix},\ \begin{matrix}\widehat{9}\\10\\11\end{matrix},\ \begin{matrix}\widehat{10}\\12\end{matrix}\right)$$

(c) A proper-path-decomposition of G

$$\left(\begin{matrix}\widehat{1}\\2\\3\\4\end{matrix},\ \begin{matrix}\widehat{2}\\3\\4\\5\end{matrix},\ \begin{matrix}\widehat{3}\\4\\5\\6\end{matrix},\ \begin{matrix}\widehat{3}\\4\\6\\7\end{matrix},\ \begin{matrix}\widehat{3}\\4\\7\\8\end{matrix},\ \begin{matrix}\widehat{4}\\7\\8\\9\end{matrix},\ \begin{matrix}\widehat{7}\\8\\9\\10\end{matrix},\ \begin{matrix}\widehat{8}\\9\\10\\11\end{matrix},\ \begin{matrix}\widehat{9}\\10\\11\\12\end{matrix}\right)$$

(d) A full proper-path-decomposition of G

Fig. 1. Path-decompositions.

ing condition (*). But this is contradicting to the choice of $\mathscr{X}$ since $|X_{i-1}| \leqslant k$. Thus $\mathscr{X}'$ is a k-path-decomposition of G. But again this is contradicting to the choice of $\mathscr{X}$. Thus $|X_i| = k+1$ for any i $(2 \leqslant i \leqslant r)$. Since $(X_r, \ldots, X_1)$ is also a path-decomposition of G, $|X_i| = k+1$ for any i $(1 \leqslant i \leqslant r)$.

Assume next that $|X_i \cap X_{i+1}| \leqslant k-1$ for some i $(1 \leqslant i \leqslant r-1)$. If $|X_{i-1} \cap X_{i+1}| = k-1$, let $v \in X_i - X_{i-1}$; otherwise let $v \in X_i - X_{i+1}$. In either case, we have $v \notin X_{i+1}$ and $|X_{i-1} \cap (X_{i+1} \cup \{v\})| \leqslant k-1$. If $|X_{i+1} \cap X_{i+2}| = k$, let $u \in (X_{i+1} \cap X_{i+2}) - X_i$. Note that $(X_{i+1} \cap X_{i+2}) - X_i \neq \emptyset$ since $|X_{i+1} \cap X_{i+2}| = k > k-1 \geqslant |X_i \cap X_{i+1}|$. If $|X_{i+1} \cap X_{i+2}| < k$, let $u \in X_{i+1} - X_i$. In either case, we have $|(X_{i+1} - \{u\}) \cap X_{i+2}| \leqslant k-1$. Since $v \notin \bigcup_{j=i+1}^{r} X_j$ and $u \notin \bigcup_{j=1}^{i} X_j$, the sequence

$\mathscr{X}'=(X_1,\ldots,X_i,(X_{i+1}\cup\{v\})-\{u\},X_{i+1},\ldots,X_r)$ satisfies condition (*) and conditions (ii)–(iv) in Definition 1. To verify condition (i), assume that $X_j\subseteq(X_i\cup\{v\})-\{u\}$ or $(X_i\cup\{v\})-\{u\}\subseteq X_j$ for some j $(1\leqslant j\leqslant r)$. Since $|(X_i\cup\{v\})-\{u\}|=|X_j|=k+1$, $X_j=(X_i\cup\{v\})-\{u\}$. Then $j=i$ or $j=i+1$, since if $j\leqslant i$, $X_j=X_j\cap((X_i\cup\{v\})-\{u\})\subseteq X_i$; otherwise, $X_j=((X_i\cup\{v\})-\{u\})\cap X_j\subseteq X_{i+1}$. But this is contradicting to $v\neq u$ or the assumption that $|X_i\cap X_{i+1}|\leqslant k-1$. Thus $\mathscr{X}'$ satisfies condition (i) in Definition 1, and $\mathscr{X}'$ is a k-path-decomposition of G satisfying condition (*). But this is contradicting to the choice of $\mathscr{X}$ since $|(X_i\cup\{v\})-\{u\}|=k+1$. Thus $|X_i\cap X_{i+1}|=k$ for any i $(1\leqslant i\leqslant r-1)$.

Therefore $\mathscr{X}$ is a full k-path-decomposition of G satisfying (*), and so a full k-proper-path-decomposition of G by Lemma 1(2) since $|X_{i-1}\cap X_{i+1}|\leqslant k-1=|X_i|-2$ $(1<i<r)$. □

Lemma 3. *For any graph G with $ppw(G)=k$, there exists a full k-proper-path-decomposition of G.*

Proof. A k-proper-path-decomposition $(X_1,X_2,\ldots,X_r)$ of G is a k-path-decomposition satisfying condition (*) in Lemma 2. Thus we obtain the lemma from Lemma 2. □

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) $\Omega_1=\{K_{1,3}\}$, where $K_{1,3}$ is a complete bipartite graph shown in Fig. 4(a).
(ii) If Ω_k is defined, a tree T is in Ω_{k+1} if and only if T is a star-composition of three (not necessarily distinct) 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 by the authors in [18], in which Theorem C was used to prove Theorem A.

Theorem A (Takahashi et al. [18]). *For any tree T and an integer k $(k\geqslant 1)$, $ppw(T)\leqslant k$ if and only if T contains no tree in Ω_k as a minor.*

Theorem B (Takahashi et al. [18]). (1) *The number of vertices of a tree in Ω_k is* $(3^{k+1}-1)/2$ $(k\geqslant 1)$.

(2) $|\Omega_k|\geqslant(k!)^2$ $(k\geqslant 1)$.

Theorem C (Takahashi et al. [18]). *For any tree T and an integer k $(k\geqslant 1)$, $ppw(T)\geqslant k+1$ if and only if T has a vertex v such that $T\setminus\{v\}$ has at least three*

connected components with proper-path-width k or more, where $T\backslash\{v\}$ *is the graph obtained from T by deleting v.*

A *k-clique* of a graph G is a complete subgraph of G on k vertices. For a positive integer k, *k-trees* are defined recursively as follows: (1) the complete graph on k vertices is a k-tree; (2) given a k-tree Q on n vertices ($n \geqslant k$), a graph obtained from Q by adding a new vertex adjacent to the vertices of a k-clique of Q is a k-tree on $n+1$ vertices. A k-tree Q is called a *k-path* if either $|V(Q)| \leqslant k+1$ or Q has exactly two vertices of degree k. A *partial k-path* is a subgraph of a k-path. A graph is said to be *simple* if it has neither loops nor multiple edges.

Theorem 2. *For any simple graph G and an integer k* ($k \geqslant 1$), $ppw(G) \leqslant k$ *if and only if G is a partial k-path.*

Proof. Suppose that $ppw(G) = h \leqslant k$. There exists a full h-proper-path-decomposition $\mathscr{X} = (X_1, X_2, \ldots, X_r)$ of G by Lemma 3. If $r = 1$ then G is a subgraph of a complete graph on $h+1$ vertices, and so we conclude that G is a partial h-path. Thus we assume that $r \geqslant 2$. We construct an h-path H from $\mathscr{X}$ 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) Define that Q_2 is the h-path obtained 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 ($2 \leqslant i \leqslant r$), define that Q_{i+1} is the h-path obtained from Q_i by adding $v_i \in X_i - X_{i-1}$ and the edges connecting v_i and the vertices in $X_i - \{v_i\}$.

(iv) Define $H = Q_{r+1}$.

From the definition of full h-proper-path-decomposition, v_i ($2 \leqslant i \leqslant r$) in (iii) is uniquely determined. Since $X_i - \{v_i\} \subset X_{i-1}$ ($2 \leqslant i \leqslant r$), the induced subgraph of Q_i on $X_i - \{v_i\}$ is an h-clique of Q_i ($2 \leqslant i \leqslant r$), and the induced subgraph of Q_{i+1} on X_i is $(h+1)$-clique. Thus H is an h-tree. Notice that $v_i \in X_{i+1}$ ($2 \leqslant i \leqslant r-1$), for otherwise $|X_{i-1} \cap X_{i+1}| = h$. Since only the vertex in $X_2 - X_1$ and v_r have degree h, H is an h-path. Furthermore, we have $V(H) = V(G)$ and $E(H) \supseteq E(G)$ from the definitions of proper-path-decomposition and Q_i. Thus G is a partial h-path, and so a partial k-path.

Conversely, suppose, without loss of generality, that G is a partial h-path ($h \leqslant k$) with n ($n > h$) vertices and H is an h-path such that $V(H) = V(G)$ and $E(H) \supseteq E(G)$. Since a graph obtained from an h-path by deleting a vertex of degree h, if exists, is also and h-path, H can be obtained as follows.

(i) Denote by $Q_1 = R_1$ the complete graph on h vertices.

(ii) Given Q_i and R_i ($1 \leqslant i \leqslant n-h$), denote by Q_{i+1} the h-path obtained from Q_i by adding vertex $v_i \notin Q_i$ and the edges connecting v_i and the vertices of R_i, and let R_{i+1} be an h-clique of Q_{i+1} that contains v_i.

(iii) Define $H = Q_{n-h+1}$.

We define $X_i = V(R_i) \cup \{v_i\}$ ($1 \leqslant i \leqslant n-h$) and $\mathscr{X} = (X_1, X_2, \ldots, X_{n-h})$. It is easy to see that $|X_i| = h+1$ for any i, $\bigcup_{i=1}^{n-h} X_i = V(H)$, and each vertex appears in consecutive

X_i's. Thus $\mathscr{X}$ satisfies conditions (ii) and (iv) in Definition 1, and the width of $\mathscr{X}$ is h. Since $v_i \in X_i - X_{i-1}$ and $\emptyset \neq V(R_{i-1}) - V(R_i) \subseteq X_{i-1} - X_i$, $X_i \not\subseteq X_{i-1}$ and $X_{i-1} \not\subseteq X_i$ for any i. Thus $X_i \not\subseteq X_j$ for any distinct i and j, for otherwise $X_i = X_i \cap X_j \subseteq X_{i+1}$ $(i<j)$ or $X_i = X_i \cap X_j \subseteq X_{i-1}$ $(i>j)$. Hence $\mathscr{X}$ satisfies condition (i) in Definition 1. Each edge of H either connects v_i with a vertex in $V(R_i)$ for some i or connects vertices in $V(R_1)$. So, both ends of each edge of H is contained in some X_i. Thus $\mathscr{X}$ satisfies condition (iii) in Definition 1. Since $V(R_{i+1}) = X_i \cap X_{i+1}$, $|X_i \cap X_{i+1}| = |V(R_{i+1})| = h$ for any i with $1 \leqslant i < n-h$. Since $X_{i+1} - X_{i-1} = \{v_i, v_{i+1}\}$, $|X_{i-1} \cap X_{i+1}| = h-1 = |X_i| - 2$ $(1 < i < n-h)$. Thus the sequence $\mathscr{X}$ is a full h-proper-path-decomposition of H from Lemma 1(2). Therefore, we have that $ppw(G) \leqslant ppw(H) \leqslant h \leqslant k$. □

Arnborg et al. [1] proved that the problem of deciding, given a graph G and an integer k, whether G is a partial k-path is NP-complete. Thus we immediately have the following by Theorem 2.

Theorem 3. *The problem of computing $ppw(G)$ is* NP*-hard.*

It should be noted that Theorem A together with Robertson and Seymour's results on graph minors [14, 15] provides $O(n^2)$ algorithm to decide, given a tree T on n vertices, whether $ppw(T) \leqslant k$ for any fixed integer k, although it is not practical even if we could solve MINOR CONTAINMENT (see [6], for example) efficiently, because $|\Omega_k| \geqslant (k!)^2$ as is shown in Theorem B(2).

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

Theorem 4. *For any tree T, the problem of computing $ppw(T)$ is solvable in linear time.*

Proof. Our algorithm to compute $ppw(T)$ is shown in Figs. 2 and 3. The outline of the algorithm is as follows.

We define the path-vector $\overline{pv}(v, T) = (p, c, n)$ for any tree T with a vertex $v \in V(T)$ as the root to compute $ppw(T)$. p describes the proper-path-width of T. c and n describe condition of T as follows: If there exists $u \in V(T) - \{v\}$ such that $T \backslash \{u\}$ has two connected components with proper-path-width $ppw(T)$ and without v, than $c = 3$ and n is the path-vector of the connected component of $T \backslash \{u\}$ containing v; Otherwise, c is the number of the connected components of $T \backslash \{v\}$ with proper-path-width $ppw(T)$ and $n = nul$. It should be noted that for any vertex $u \in V(T)$ the number of connected components of $T \backslash \{u\}$ with proper-path-width $ppw(T)$ is at most two from Theorem C. Notice also that if there exists u such that $T \backslash \{u\}$ has two connected components with proper-path-width $ppw(T)$ and without v then u is uniquely determined. If there is no such u then the number of connected components of $T \backslash \{w\}$ with proper-path-width $ppw(T)$ and without v is not more than the number of connected components of $T \backslash \{v\}$ with proper-path-width $ppw(T)$. In the following, we denote an element x in $\overline{pv}(v, T)$ by $\overline{pv}(v, T)|x$, where x is either p, c or n.

```
Procedure MERGE( P_s, P_t )
[ Input:   P_s (path-vector of tree T_s rooted at s )
           P_t (path-vector of tree T_t rooted at t )
  Output:  the path-vector of tree rooted at s
           obtained from T_s and T_t by adding an edge (s,t). ]
1.  if P_s|p > P_t|p then
 1.1   if P_s|c ≤ 2 then P_s := (p, c, nul);
 1.2   else if P_s|n*|p < P_t|p then P_s := (p + 1, 0, nul);
 1.3   else if P_s|n*|p = P_t|p then
  1.3.1    if P_s|n*|c ≥ 2 or P_t|c ≥ 2 then P_s := (p + 1, 0, nul);
  1.3.2    else P_s|n* := (p, c + 1, nul);
 1.4   else if P_s|n*|c ≤ 2 then P_s|n* := (p, c, nul);
 1.5   else if P_s|n*|c = 3 then
  1.5.1    P_s|n*|n := MERGE( P_s|n*|n, P_t );
  1.5.2    if P_s|n*|n|p = P_s|n*|p then P_s := (p + 1, 0, nul);
       endif
 1.6   return( P_s );
2.  else if P_s|p = P_t|p then
 2.1   if P_s|c ≥ 2 or P_t|c ≥ 2 then P_s := (p + 1, 0, nul);
 2.2   else P_s := (p, c + 1, nul);
 2.3   return( P_s );
3.  else if P_s|p < P_t|p then
 3.1   if P_t|c ≤ 1 then P_t := (p, 1, nul);
 3.2   else if P_t|c = 2 then P_t := (p, 3, P_s);
 3.3   else if P_s|p > P_t|n*|p then P_t := (p + 1, 0, nul);
 3.4   else if P_s|p = P_t|n*|p then
  3.4.1    if P_s|c ≥ 2 or P_t|n*|c ≥ 2 then P_t := (p + 1, 0, nul);
  3.4.2    else P_t|n* := (p, P_s|c + 1, nul);
 3.5   else if P_t|n*|c ≤ 1 then P_t|n* := (p, 1, nul);
 3.6   else if P_t|n*|c = 2 then P_t|n* := (p, 3, P_s);
 3.7   else if P_t|n*|c = 3 then
  3.7.1    P_t|n*|n := MERGE( P_s, P_t|n*|n );
  3.7.2    if P_t|n*|n|p = P_t|n*|p then P_t := (p + 1, 0, nul);
       endif
 3.8   return( P_t );
   endif
END
```

Fig. 2. Procedure MERGE: The algorithm to compute the path-vector of the join of two subtrees.

Let T_1 be a tree with root $v \in V(T_1)$ and P_1 be the path-vector of T_1. We recursively define T_{i+1} and P_{i+1} $(1 < i \leqslant l)$ while $P_i|c = 3$ as follows: let $u_i \in V(T_i)$ be the vertex such that $T_i \backslash \{u_i\}$ has two connected components with proper-path-width $ppw(T_i)$ and without v, T_{i+1} be the connected component of $T_i \backslash \{u_i\}$ containing v as the root, and P_{i+1} be the path-vector of T_{i+1}. We call such path-vectors $P_1, P_2, \ldots, P_l$ the chain of the path-vector P_1. We define b, n^*, b^*, and btm in the chain of P_1 as follows: define that $P_i|b = P_{i-1}$ $(2 \leqslant i \leqslant l)$; define that $P_i|n^* = P_j$ if $i = 1$ or $P_i|p < P_{i-1}|p - 1$ $(2 \leqslant i \leqslant l)$ where j is the maximum integer such that $j - i = P_i|p - P_j|p$; define that $P_i|b^* = P_j$ if $P_j|n^*$ is defined and $P_j|n^* = P_i$; define that $P_1|btm = P_l$. Thus we extend a path-vector

```
Procedure LMERGE( P_s, P_t )
[ Input:   P_s (path-vector of tree T_s rooted at s )
           P_t (path-vector of tree T_t rooted at t )
  Output:  the path-vector of tree rooted at s
           obtained from T_s and T_t by adding an edge (s,t). ]
1. if P_s|p > P_t|p and P_s|c = 3 then
 1.1   if P_s|btm|b*|p ≥ P_t|p then let P' be P_s|btm|b*;
 1.2   else
           let P' be the path-vector P in the chain of P_s such that P|n* is defined
           and P|p ≥ P_t|p > P|n*|n|p;
 1.3   P' := MERGE( P', P_t );
 1.4   return( P_s );
   endif
2. if P_s|p < P_t|p and P_t|c = 3 then
 2.1   if P_t|btm|b*|p ≥ P_s|p then let P' be P_t|btm|b*;
 2.2   else
           let P' be the path-vector P in the chain of P_t such that P|n* is defined
           and P|p ≥ P_s|p > P|n*|n|p;
 2.3   P' := MERGE( P_s, P' );
 2.4   return( P_t );
   endif
3. return( MERGE( P_s, P_t ) );
END
Procedure DFS( s )
[ Input:   a vertex s
  Output:  the path-vector of the maximal subtree rooted at s ]
1. P_s := (1,0,nul); /* path-vector of a tree with one vertex s */
2. for all children t of s in T do
 2.1   P_t := DFS( t );
 2.2   P_s := LMERGE( P_s, P_t );
   endfor
3. return( P_s );
END
Procedure MAIN( T )
[ Input:   a tree T
  Output:  the proper-path-width of T ]
1. Let r be a vertex in V(T);
2. p̄v(r,T) := DFS( r );
3. return( p̄v(r,T)|p );
END
```

Fig. 3. The algorithm to compute $ppw(T)$.

as $\overline{pv}(v,T)=(p,c,n,b,n^*,b^*,btm)$. This is the same technique to reduce the time to traverse the chain as used in [10].

In the procedure, we omit the description of substitutions for b, n^*, b^*, and btm in the path-vector because no confusion is caused. Moreover, after substitutions, we can update n^*, b^*, and btm in the path-vectors in the chain in constant time. So we also omit the description of these operations. For the simplicity, if the substitution for P uses $P|x$, we abbreviate $P|x$ to x.

Suppose that a tree T_0 rooted at s is obtained from tree T_s rooted at s and tree T_t rooted at t by adding an edge (s,t). Based on Theorem C, Procedure MERGE shown

in Fig. 2 recursively calculates the path-vector of T_0 from the path-vector P_s of T_s and the path-vector P_t of T_t in $O(a)$ time where $a=\max(ppw(T_s), ppw(T_t))$. Note that the time complexity of Procedure MERGE is $O(1)$ except for recursive calls. Since the larger proper-path-width of two merged trees is reduced by at least two whenever Procedure MERGE is recursively called, the number of recursive calls is at most $a-1$.

In Procedure LMERGE shown in Fig. 3, we can determine P' in $O(b)$ time by using btm and b^* in the chain of the path-vector where $b=\min(ppw(T_s), ppw(T_t))$. If P' is determined at 1.2 or 2.2 in Procedure LMERGE then the number of recursive calls of Procedure MERGE is at most $P'|n^*|n|p<b$. Otherwise Procedure MERGE returns the path-vector in $O(1)$ time. Thus Procedure LMERGE calculates the path-vector of the join of two subtrees in $O(b)$ time. Procedure DFS shown in Fig. 3 computes the path-vector of a maximal subtree rooted at s in T from the path-vectors of maximal subtrees rooted at children of s in T by using Procedure LMERGE. Procedure MAIN shown in Fig. 3 obtains the proper-path-width of T from the path-vector of T obtained by Procedure DFS. The algorithm starts with the isolated vertices obtained from T by deleting all edges in T and reconstruct T by adding edge by edge while computing path-vectors of connected components.

Let $S(T)$ denote the time required to compute the path-vector of T, $M(T_1, T_2)$ denote the time required to obtain the path-vector of T from the path-vectors of T_1 and T_2 by Procedure LMERGE. From Theorem B(1), we have $ppw(T)=O(\log|V(T)|)$. Thus we have the following:

$$\begin{aligned} S(T) &\leqslant S(T_1)+S(T_2)+M(T_1,T_2) \\ &\leqslant S(T_1)+S(T_2)+O(\min(ppw(T_1), ppw(T_2))) \\ &\leqslant S(T_1)+S(T_2)+O(\log(\min(|V(T_1)|, |V(T_2)|))). \end{aligned}$$

Notice that the recurrence defined by $f(1)=1$ and, for $n\geqslant 2$,

$$f(n)=\max_{1\leqslant i<n}(f(i)+f(n-i)+\lceil \log_2(\min(i,n-i))\rceil)$$

satisfies $f(n)=O(n)$. An easy way to verify this is to prove that, for $n\geqslant 1$, $f(n)\leqslant 2n-1-\lceil \log_2 n\rceil$ by a straightforward induction. Thus we can prove that the time complexity of the algorithm is $O(n)$ where $n=|V(T)|$. □

We should mention that for any tree T with n vertices and $ppw(T)=k$, we can construct in $O(n\log n)$ time a k-proper-path-decomposition of T by a slight modification of the algorithm shown in Figs. 2 and 3 [21].

3. Mixed Searching

In *mixed search 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 2. 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 such a *mixed search* is to clear all edges by a search. A mixed search is *optimal* if the maximum number of searchers on G at any operation is minimum over all mixed searches of G. This number is called the *mixed search number* of G, and is denoted by $ms(G)$.

We first show a relation to edge searching and node searching: for any graph G, $es(G)-1 \leqslant ms(G) \leqslant es(G)$ and $ns(G)-1 \leqslant ms(G) \leqslant ns(G)$. The edge search and node search are special cases of the mixed search by definition. Thus we have $ms(G) \leqslant es(G)$ and $ms(G) \leqslant ns(G)$. 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 $es(G) \leqslant ms(G)+1$. 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 $ns(G) \leqslant ms(G)+1$. All four cases are possible as shown in Fig. 4.

A *crusade* in G, introduced by Bienstock and Seymour [2], is a sequence $(C_1, C_2, \ldots, C_r)$ of subsets of $E(G)$, such that $C_1=\emptyset$, $C_r=E(G)$, and $|C_i-C_{i-1}| \leqslant 1$ for $1 \leqslant i \leqslant r$. The crusade uses at most k searchers if the number of vertices which are ends of an edge in C_i and also of an edge in $E(G)-C_i$ is at most k for $1 \leqslant i \leqslant r$. Bienstock and Seymour proved the following theorem.

Theorem D (Bienstock and Seymour [2]). *For any graph G with minimum degree at least two, $ms(G) \leqslant k$ if and only if there exists a crusade in G using at most k searchers.*

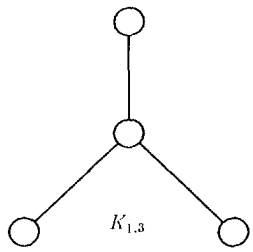
Moreover, they proved the following theorem by using the crusade.

Theorem E (Bienstock and Seymour [2]). *For any graph, there exists an optimal mixed search without recontamination of cleared edges,*

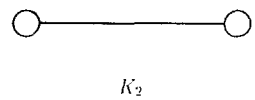
This was proved independently by the authors in [19] using an optimal node search without recontamination of cleared edges.

We obtain the following corollary from Theorem E.

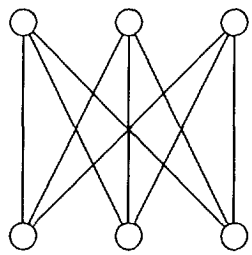
Corollary 1. *For any graph G, there exists an optimal mixed search without recontamination of cleared edges such that it is a sequence of operations (a), (b), or (c) of*



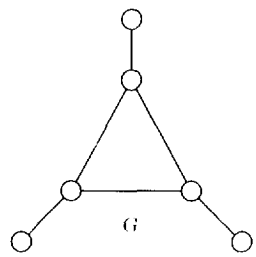
(a) $ms(K_{1,3})=2$, $es(K_{1,3})=2$, $ns(K_{1,3})=2$



(b) $ms(K_2)=1$, $es(K_2)=1$, $ns(K_2)=2$



(c) $ms(K_{3,3})=4$, $es(K_{3,3})=5$, $ns(K_{3,3})=4$



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

Fig. 4. Search numbers of graphs.

Definition 2, *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*.

A mixed search described above is said to be *simple*.

Bienstock and Seymour characterized the mixed search number of a graph with minimum degree at least two by the concept of crusade as shown in Theorem D. In the following, we characterize the mixed-search number of a simple graph by the proper-path-width.

Theorem 5. *For any simple graph* G, $ms(G)=ppw(G)$.

Proof. Suppose that $ppw(G)=k$ and $\mathscr{X}=(X_1, X_2, \ldots, X_r)$ is a full k-proper-path-decomposition of G. If $r=1$ then let v_1 and u_1 be distinct vertices in X_1 and place k searcher on the vertices of $X_1-\{v_1\}$. If $(u_1, v_1)\in E(G)$, slide a searcher on u_1 to v_1 and place it on v_1. Otherwise, delete a searcher from u_1 and place a searcher on v_1. This defines a mixed search with k searchers. Thus we assume $r\geqslant 2$. 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: Let u_1 be a vertex in $X_1 - X_2$. If $(u_1, v_1) \in E(G)$, slide a searcher on u_1 toward v_1 and place it on v_1. Otherwise, delete a searcher from u_1 and place a searcher on v_1. Let $i = 1$.

Step 3: Repeat Step 3 while $i \leqslant r-2$. Let $i = i+1$. Let u_i be a vertex in $X_i - X_{i+1}$ and v_i be a vertex in $X_i - X_{i-1}$. If $(u_i, v_i) \in E(G)$, slide a searcher on u_i toward v_i and place it on v_i. Otherwise, delete a searcher from u_i and place a searcher on v_i.

Step 4: Let u_r be a vertex in $X_{r-1} \cap X_r$ and v_r be a vertex in $X_r - X_{r-1}$. If $(u_r, v_r) \in E(G)$, slide a searcher on u_r toward v_r and place it on v_r. Otherwise, delete a searcher from u_r and place a searcher on v_r.

From the definition of full k-proper-path-decomposition, both u_i $(1 \leqslant i \leqslant r-1)$ and v_i $(2 \leqslant i \leqslant r)$ are uniquely determined. It should be noted that $((X_i - \{v_i\}) - \{u_i\}) \cup \{v_i\} = X_i \cap X_{i+1} = X_{i+1} - \{v_{i+1}\}$ and $u_{i+1} \in X_{i+1} - \{v_{i+1}\}$ for $1 \leqslant i \leqslant r-1$. An edge with both its ends in $X_i - \{v_i\}$ $(1 \leqslant i < r)$ is cleared since the vertices in $X_i - \{v_i\}$ have searchers simultaneously in Step 1, 2, or 3. Also, an edge with both its ends in $X_r - \{u_r\}$ is cleared since the vertices in $X_r - \{u_r\}$ have searchers simultaneously in Step 4. Since G is simple, there exists at most one edge connecting u_i and v_i $(1 \leqslant i \leqslant r)$, and each edge (u_i, v_i), if exists, is cleared by sliding a searcher along the edge. Thus all edges are cleared at least once. Suppose that all edges connecting the vertices in $\bigcup_{1 \leqslant j \leqslant i-1} X_j$ are clear and k searchers are placed on the vertices in $X_i - \{v_i\}$. Since $u_i \notin \bigcup_{i+1 \leqslant j \leqslant r} X_j$, all edges incident to u_i except for (u_i, v_i), if exists, are clear when a searcher on u_i is deleted or slid from u_i. Thus, when the searcher is placed on v_i, all edges in $\bigcup_{1 \leqslant j \leqslant i} X_j$ are clear and k searchers are placed on the vertices in $X_{i+1} - \{v_{i+1}\}$. Thus by induction no edge is recontaminated. Thus the search above is indeed a mixed search with at most $ppw(G)$ searchers, and we have $ms(G) \leqslant ppw(G)$.

Conversely, suppose that we have a simple mixed search $\mathscr{S}$ with k searchers. For the ith operation of $\mathscr{S}$, 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 consisting of u, v, and the vertices having searchers.

Let $\mathscr{X} = (X_1, X_2, \ldots, X_s)$ be the resulting sequence of sets of vertices. Since both ends of an edge which is cleared in the ith operation are contained in X_i, all edges are contained in some X_i. Since $\mathscr{S}$ is simple, $\bigcup_{1 \leqslant i \leqslant s} X_i = V(G)$ and each vertex of G appears in consecutive X_i's. By the definition of X_i, $|X_i| \leqslant k+1$ for any i. Let $\mathscr{X}' = (X'_1, X'_2, \ldots, X'_r)$ be a maximal subsequence of $\mathscr{X}$ such that $X'_i \not\subseteq X'_j$ for any distinct i and j. Notice that $\mathscr{X}'$ satisfies conditions (i)–(iv) in Definition 1. We shall show that k-path-decomposition $\mathscr{X}'$ satisfies condition (*) in Lemma 2. If one of X'_{i-1}, X'_i, and X'_{i+1} is defined by (1), it is easy to see that $|X'_{i-1} \cap X'_{i+1}| \leqslant k-1$. If all X'_{i-1}, X'_i, and X'_{i+1} are defined by (2), then $|X'_i| \leqslant k+1$ and there exist distinct u and v in X'_i such that $u \notin X'_{i+1}$, and $v \notin X'_{i-1}$. Thus we have $|X'_{i-1} \cap X'_{i+1}| \leqslant k-1$. Therefore, $\mathscr{X}'$ satisfies condition (*) in Lemma 2, and there exists a full k-proper-path-decomposition of G by Lemma 2. Thus $ppw(G) \leqslant ms(G)$. □

It should be noted that Theorems A and 5 provide a structural characterization of trees T with $ms(T) \leqslant k$.

From Theorems 3–5, we have the following complexity results on $ms(G)$.

Theorem 6. *The problem of computing $ms(G)$ is* NP*-hard for general graphs but can be solved in linear time for trees.*

We conclude with the following remarks:

1. Notice that Theorem 5 does not hold for multiple graphs. If G is the graph consisting of two parallel edges, $ppw(G)=1$, and $ms(G)=2$. However we can prove that $ppw(G) \leqslant ms(G) \leqslant ppw(G)+1$ for any multiple graph G.
2. Bodlaender and Kloks [4] showed an $O(n \log^2 n)$ time algorithm to decide whether $pw(G) \leqslant k$ for any graph G and a fixed integer k. We can modify their algorithm to decide whether $ppw(G) \leqslant k$ for any graph G and a fixed integer k.
3. A relation between the mixed searching and another search games in which the vertices must be cleared instead of edges [3, 17] is mentioned in [19].

Acknowledgments

The authors would like to thank Prof. Berlekamp for his kind suggestions and an anonymous referee for detailed comments which substantially improved the presentation of this paper.

References

[1] S. Arnborg, D.G. Corneil and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM J. Algebraic Discrete Methods* **8** (1987) 277–284.
[2] D. Bienstock and P. Seymour, Monotonicity in graph searching, *J. Algorithms* **12** (1991) 239–245.
[3] H.L. Bodlaender, Improved self-reduction algorithms for graphs with bounded treewidth, in: *Proc. Workshop on Graph-Theoretic Concepts in Computer Science WG '89*, Lecture Notes in Computer Science, Vol. 411 (Springer, Berlin, 1989) 232–244.
[4] H.L. Bodlaender and T. Kloks, Better algorithm for the pathwidth and treewidth of graphs, in: *Proc. 18th Internat. coll. on Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 510 (Springer, Berlin, 1991) 544–555.
[5] R.L. Breisch, An intuitive approach to speleotopology, *Southwestern Cavers* **6** (the Southwestern Region of the National Speleological Society, 1967) 72–78.
[6] D.S. Johnson, The NP-completeness column: an ongoing guide, *J. Algorithms* **8** (1987) 285–303.
[7] L.M. Kirousis and C.H. Papadimitriou, Interval graphs and searching, *Discrete Math.* **55** (1985) 181–184.
[8] L.M. Kirousis and C.H. Papadimitriou, Searching and pebbling. *Theoret. Comput. Sci.* **47** (1986) 205–218.
[9] A.S. LaPaugh, Recontamination does not help to search a graph, *J. ACM* **40** (1993) 224–245.
[10] M. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson and C.H. Papadimitriou, The complexity of searching a graph, *J. ACM* **35** (1988) 18–44.
[11] R.H. Möhring, Graph problems related to gate matrix layout and PLA folding, in: G. Tinhofer, E. Mayr, H. Noltemeier and M. Syslo, eds., *Computational Graph Theory* (Springer, Wien, computing suppl. 7th ed. 1990) 17–51.

[12] T.D. Parsons, Pursuit-evasion in a graph, in: *Proc. Internat. Conf. on Theory and Applications of Graphs*, Lecture Notes in Mathematics, Vol. 642 (Springer, Berlin, 1976) 426–441.
[13] N. Robertson and P.D. Seymour, Graph minors. I. Excluding a forest, *J, Combin. Theory Ser. B***35** (1983) 39–61.
[14] N. Robertson and P.D. Seymour, Graph minors. XIII. The disjoint paths problem, preprint, 1986.
[15] N. Robertson and P.D. Seymour, Graph minors. XVI. Wagner's conjecture, preprint, 1987.
[16] P. Scheffer, A linear algorithm for the pathwidth of trees, in: R. Bodendiek and R. Henn, eds., *Topics in Combinatorics and Graph Theory* (Physica, Heidelberg, 1990) 613–620.
[17] S. Shinoda, On some problems of graphs – including Kajitani's conjecture and its solution, in: *Proc. 2nd Karuizawa Workshop on Circuits and Systems* (1989) 414–418 (in Japanese).
[18] A. Takahashi, S. Ueno and Y. Kajitani, Minimal acyclic forbidden minors for the family of graphs with bounded path-width, *Discrete Math.* **127** (1994) 293–304.
[19] A. Takahashi, S. Ueno and Y. Kajitani, Mixed-searching and proper-path-width, Tech. Report, SIGAL 91-22-7, IPSJ, 1991.
[20] A. Takahashi, S. Ueno, and Y. Kajitani, Mixed-searching and proper-path-width, in: *Proc. 2nd Internat. Symp. on Algorithms*, Lecture Notes in Computer Science, Vol. 557 (Springer, Berlin, 1991) 61–71.
[21] A. Takahashi, S. Ueno and Y. Kajitani, On the proper-path-decomposition of trees Tech. Report, CAS 91-74, IEICE, 1991.