中華民國八十六年全國計算機會議

在梯形圖上找關節點與橋的最佳演算法
Optimal Algorithms for Finding Cut Vertices and Bridges on Trapezoid Graphs

陳宏昌 王有禮
Hon-Chan Chen Yue-Li Wang

國立台灣科技大學資訊管理技術系
Department of Information Management
National Taiwan University of Science and Technology
ylwang@cs.ntust.edu.tw

摘 要

在這篇論文中，我們將提出在梯形圖上找關節點與橋的最佳演算法。我們的演算法可以在 $O(n)$ 的時間解出問題，或在 EREW PRAM 的計算模式上用 $O(n / \log n)$ 個處理器在 $O(\log n)$ 的時間內解出。

關鍵字：構成要素、關節點、橋、梯形圖、演算法

Abstract

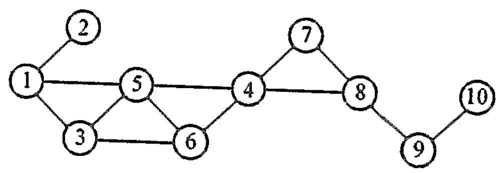
In this paper, we will propose optimal algorithms for finding cut vertices and bridges on trapezoid graphs. Our algorithms can solve this problem in $O(n)$ time, or in $O(\log n)$ time by using $O(n / \log n)$ processors on the EREW PRAM computational model.

Keywords: component, cut vertex, bridge, trapezoid graph, algorithm

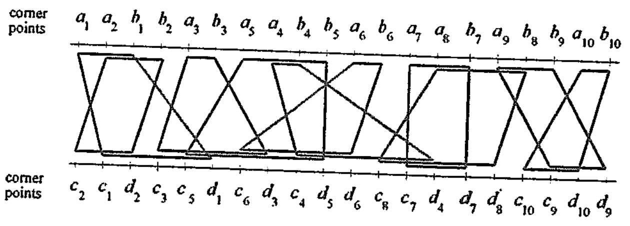
1. Introduction

Let $G = (V, E)$ be a graph with vertex set V and edge set E. A graph $H = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. If H is a maximal connected subgraph of G, then H is a *component* of G. A vertex v is called a *cut vertex* of G if $\kappa(G - v) > \kappa(G)$, where $\kappa(G)$ is the number of components of G. Similarly, graph $G - e$, $e \in E$, is the graph obtained by deleting e from E, and an edge e is a *bridge* of G if $\kappa(G - e) > \kappa(G)$.

The class of trapezoid graphs is introduced by Dagan et al. [4]. A trapezoid i is defined by four corner points a_i, b_i, c_i, d_i such that a_i and b_i are on the top channel while c_i and d_i are on the bottom channel in the trapezoid diagram. A graph $G = (V, E)$ is a *trapezoid graph* if it can be represented by a trapezoid diagram such that each trapezoid corresponds to a vertex in V and $(i, j) \in E$ if and only if trapezoids i and j intersect in the trapezoid diagram. Figure 1 presents a trapezoid graph and the corresponding trapezoid diagram.



(a)


corner points
a_1 a_2 b_1 b_2 a_3 b_3 a_5 a_4 b_4 b_5 a_6 b_6 a_7 a_8 b_7 a_9 b_8 b_9 a_{10} b_{10}
corner points
c_2 c_1 d_2 c_3 c_5 d_1 c_6 d_3 c_4 d_5 d_6 c_8 c_7 d_4 d_7 d_8 c_{10} c_9 d_{10} d_9

(b)

Figure 1. (a) A trapezoid graph
(b) The corresponding trapezoid diagram.

Trapezoid graphs can be recognized in $O(n^2)$ time by Ma and Spinrad's algorithm [10]. Applying their algorithm, trapezoid diagrams can also be constructed. It is easy to show that we can reconstruct a trapezoid diagram into another one, corresponding to the same trapezoid graph, such that each trapezoid has four distinct corner points and no two trapezoids share common corner points. Therefore, for simplicity, we assume the corner points in our trapezoid diagram are all distinct. We also assume that trapezoids are labeled in increasing order of their b corner points for ease of description. Thus, we know that for any three vertices i, j, and k of G, $i < j < k$, if i is adjacent to k, then j is adjacent to i or k.

The problem of finding cut vertices and bridges on graphs is well-known. It can be solved in $O(n + m)$ time by Depth-First-Search algorithm, where n is the number of vertices and m is the number of edges. Some parallel algorithms for this problem can be found in [1, 6, 11, 12, 13].

Several efficient algorithms for trapezoid graphs were proposed recently. In [8, 9], Liang gave sequential algorithms for the dominating problem and the breadth-first spanning tree problem on trapezoid graphs. A linear time algorithm for finding depth-first spanning trees on trapezoid graphs was proposed by Chen et al. [2]. They also gave an efficient parallel algorithm for constructing spanning trees on this class of graphs [3].

In this paper, we will present optimal algorithms for finding cut vertices and bridges on trapezoid graphs. Assuming the trapezoid diagrams are given, our algorithms can solve this problem in $O(n)$ time. Our algorithms can also solve this problem in parallel on the EREW PRAM model in $O(\log n)$ time using $O(n / \log n)$ processors.

The remaining part of this paper is organized as follows. In Section 2, we introduce some notations and the way to find the components of a trapezoid graph. Algorithms for finding cut vertices and bridges are presented in Section 3 and Section 4, respectively. The parallelization of our algorithms is shown in Section 5. Finally, in Section 6, we give the conclusion.

2. Preliminaries

We first introduce some notations. Let G be a trapezoid graph of n vertices and let v be a vertex of G. Denote $N(v)$ the neighbors of v. The *close neighbors* $N[v]$ of v are $N(v) \cup \{v\}$. Define the *maximum neighbor* of v, denoted by $N_{\max}[v]$, to be vertex i, $i \in N[v]$, such that $i \geq j$ for all $j \in N[v]$. Similarly, define the *minimum neighbor* $N_{\min}[v]$ of v to be vertex k, $k \in N[v]$, such that $k \leq j$ for all $j \in N[v]$. All maximum neighbors and minimum neighbors can be found in $O(n)$ time by scanning the corner points [3].

Let $A = (a_1, a_2, \ldots, a_n)$ be an array of n elements. The *prefix maxima* of A is the elements of the array $P = (p_1, p_2, \ldots, p_n)$ such that $p_i = \max\{a_1, a_2, \ldots, a_i\}$, for $i = 1, 2, \ldots, n$. Similarly, the *suffix minima* of A is the elements of the array $S = (s_1, s_2, \ldots, s_n)$ such that $s_i = \min\{a_i, a_{i+1}, \ldots, a_n\}$, for $i = 1, 2, \ldots, n$. Let pm_i be the prefix maxima of $(N_{\max}[1], N_{\max}[2], \ldots, N_{\max}[i])$ and sm_i the suffix minima of $(N_{\min}[i], N_{\min}[i+1], \ldots, N_{\min}[n])$ for $i = 1, 2, \ldots, n$. The following theorem determines if any two consecutive vertices of G are in different components.

Theorem 2.1. *Let i and i+1, $1 \leq i < n$, be two consecutive vertices of G. Then, i and i+1 are in different components of G if $pm_i < i+1$ or $i < sm_{i+1}$.*

Proof. We only prove the part of $pm_i < i+1$. The other part can be shown in a similar way. If $pm_i < i+1$, then $N_{\max}[j] < i+1$ for all j, $1 \leq j \leq i$. This means that no vertex j, $1 \leq j \leq i$, is adjacent to vertices which are greater than i. By the definition of trapezoid graphs, vertices i and i+1 are disconnected and they are in different components of G. □

Using Theorem 2.1, we can find all components of G. Since computing the prefix maxima and the suffix minima of an array takes $O(n)$ time, all components of a trapezoid graph can be found in $O(n)$ time.

3. Finding Cut Vertices

Assume G is connected. A path of G is a *dominating path* if every vertex of G is in this path or adjacent to at least one vertex of this path. There may be many different dominating paths in G, however, a shortest path connecting vertices 1 and n must be a dominating path. We call such a shortest path a *shortest dominating path.* A shortest dominating path of a trapezoid graph can be found in $O(n)$ time [9].

An *adjacent pair* $AP = [l, r]$ of G is composed of two adjacent vertices l and r, in which l and r are called the *left vertex* and the *right vertex* of AP, respectively. Without loss of generality, assume $l < r$. Two adjacent pairs $AP_i = [l_i, r_i]$ and $AP_j = [l_j, r_j]$, $i < j$, are *consecutive adjacent pairs* if $l_i < l_j$, $r_i < r_j$ and $r_i \geq l_j$. Let $H = v_1–v_2–\ldots–v_m$ be a shortest dominating path, where $v_1 = 1$ and $v_m = n$. It is easy to see that $[v_i, v_{i+1}]$ is an adjacent pair if $v_i < v_{i+1}$ for $i = 1, 2, \ldots, m-1$. The adjacent pairs which are obtained from H are called the *dominating pairs* (DPs) of G. For example, the shortest dominating path in Figure 1 (a) is H = 1–5–4–8–9–10. Thus, the dominating pairs are $DP_1 = [1, 5]$, $DP_2 = [4, 8]$, $DP_3 = [8, 9]$, and $DP_4 = [9, 10]$. It is clear that obtaining the dominating pairs can be done in $O(n)$ time.

Let H be the set of vertices in a dominating path of G. Then, any vertex $v \in G - H$ is not a cut vertex since $G - v$ remains connected. However, not all vertices in H are cut vertices. The following paragraphs explain how to determine if l_i is a cut vertex for a dominating pair $D_i = [l_i, r_i]$. With a similar argument, vertex r_i can also be determined.

At first we add two dummy vertices 0 and n+1, where $N_{\max}[0] = N_{\min}[0] = 0$ and $N_{\max}[n+1] = N_{\min}[n+1] = n+1$. We also add two dummy dominating pairs $DP_0 = [0, 1]$ and $DP_{k+1} = [n, n+1]$. Let $pm_0, pm_1, \ldots, pm_{n+1}$ be the prefix maxima of $(N_{\max}[0], N_{\max}[1], \ldots, N_{\max}[n+1])$. By Theorem 2.1, $pm_i \geq i+1$ for $1 \leq i < n$. To determine if l_i is a cut vertex, we let $p_0, p_1, \ldots, p_{n+1}$ be the prefix maxima of $(N_{\max}[0], N_{\max}[1], \ldots, N_{\max}[l_i-1], 0, N_{\max}[l_i+1], \ldots, N_{\max}[n+1])$ and check if there exists some vertex j, $1 \leq j < n$, such that $p_j < j+1$, where $N_{\max}[l_i] = 0$ is for the purpose of $G - l_i$. Since only $N_{\max}[l_i]$ is substituted by 0, $p_j \geq j+1$ for $1 \leq j < l_i$. Moreover, since vertices in DP_{i+1}, DP_{i+2}, ..., DP_k are connected, $p_j \geq j+1$ for $l_{i+1} \leq j < n$. We only need to check if $p_j < j+1$ for $l_i \leq j < l_{i+1}$. It is obvious that $p_j = pm_j$ for $0 \leq j < l_i$ and $p_{l_i} = pm_{l_i-1}$. Thus, $p_{l_i}, p_{l_i+1}, \ldots, p_{l_{i+1}-1}$ is equal to the prefix maxima of $(pm_{l_i-1}, N_{\max}[l_i+1], N_{\max}[l_i+2], \ldots, N_{\max}[l_i+1-1])$. We call sequence $p_{l_i}, p_{l_i+1}, \ldots, p_{l_{i+1}-1}$ the *P-sequence* of DP_i.

Lemma 3.1. *Let $DP_i = [l_i, r_i]$, $1 \leq i \leq k$, be a dominating pair of G with its P-sequence. Then,*
Case 1. $i = 1$. l_1 is a cut vertex if there exists some vertex j, $l_1 < j < l_2$, such that $p_j < j+1$.
Case 2. $2 \leq i \leq k$. l_i is a cut vertex if there exists some vertex j, $l_i \leq j < l_{i+1}$, such that $p_j < j+1$.
Proof. Suppose $i \neq 1$. By Theorem 2.1, if there exists some vertex j, $l_i \leq j < l_{i+1}$, such that $p_j < j+1$, then vertices j and $j+1$ are in different components of G. The reason of the disconnectivity is that we let l_i be not adjacent to any vertices greater than l_i, then $j+1$ cannot be connected to j without the adjacency of l_i and $j+1$. Thus, l_i is a cut vertex. Now, suppose $i = 1$. We need to check if $p_j < j+1$ for $l_1 < j < l_2$ since if $p_1 < 2$ but $p_j \geq j+1$ for all j, $2 \leq j < l_2$, G is still connected. □

A vertex is a *pendant vertex* if its degree is one. An edge incident to a pendant vertex is called a *pendant edge*. The following lemma applies Lemma 3.1 to find pendant edges incident to l_i.

Lemma 3.2. *Let $DP_i = [l_i, r_i]$, $1 \leq i \leq k$, be a dominating pair of G with its P-sequence. Then,*
Case 1. $1 \leq i \leq k-1$. For each vertex j, $l_i < j < l_{i+1}$, if $p_{j-1} < j$ and $p_j < j+1$, then (j, l_i) is a pendant edge
Case 2. $i = k$. For each vertex j, $l_k < j < n$, if $p_{j-1} < j$ and $p_j < j+1$, then (j, l_k) is a pendant edge. Moreover, if $p_{n-1} < n$, then (l_k, n) is a pendant edge.
Proof. By Theorem 2.1 and Lemma 3.1, if there exists some vertex j, $l_i < j < l_{i+1}$, such that $p_{j-1} < j$ and $p_j < j+1$, then l_i is a cut vertex and j is isolated. This implies that j is a pendant vertex of G which is adjacent to l_i. When $i = k$, if $p_{n-1} < n$, then l_k is a cut vertex and vertex n is isolated. Thus, (l_k, n) is a pendant edge. □

To determine r_i of DP_i, $1 \leq i \leq k$, we can apply the above argument. Let $sm_0, sm_1, \ldots, sm_{n+1}$ be the suffix minima of $(N_{\min}[0], N_{\min}[1], \ldots, N_{\min}[n+1])$. Then, the *S-sequence* $s_{r_{i-1}+1}, s_{r_{i-1}+2}, \ldots, s_{r_i}$ of DP_i is the suffix minima of $(N_{\min}[r_{i-1}+1], N_{\min}[r_{i-1}+2], \ldots, N_{\min}[r_i-1], sm_{r_i+1})$.

Lemma 3.3. *Let $DP_i = [l_i, r_i]$, $1 \leq i \leq k$, be a dominating pair of G with its S-sequence. Then,*
Case 1. $1 \leq i \leq k-1$. r_i is a cut vertex if there exists some vertex j, $r_{i-1} < j \leq r_i$, such that $j-1 < s_j$.
Case 2. $i = k$. r_k is a cut vertex if there exists some vertex j, $r_{k-1} < j < r_k$, such that $j-1 < s_j$.

Lemma 3.4. *Let $DP_i = [l_i, r_i]$, $1 \leq i \leq k$, be a dominating pair of G with its S-sequence. Then,*
Case 1. $i = 1$. For each vertex j, $1 < j < r_1$, if $j-1 < s_j$ and $j < s_{j+1}$, then (j, r_1) is a pendant edge. Moreover, if $1 < s_2$, then $(1, r_1)$ is a pendant edge.
Case 2. $2 \leq i \leq k$. For each vertex j, $r_{i-1} < j < r_i$, if $j-1 < s_j$ and $j < s_{j+1}$, then (j, r_i) is a pendant edge.

Algorithm A uses the above lemmas to find all cut vertices and pendant edges of a trapezoid graph.

```
Algorithm A (Finding cut vertices and pendant edges)
Input: $N_{\max}[i]$ and $N_{\min}[i]$ of each vertex $i$, $1 \leq i \leq n$.
Output: All cut vertices and pendant edges of $G$.
Method:
Step 1. Find a shortest dominating path $H$ of $G$.
Step 2. Transform $H$ to the dominating pairs of $G$. Let
        $DP_1 = [l_1, r_1]$, $DP_2 = [l_2, r_2]$, ..., $DP_k = [l_k, r_k]$ be
        the dominating pairs.
Step 3. Add two dummy vertices 0 and $n+1$ as well as
        two dummy dominating pairs $DP_0 = [0, 1]$ and
        $DP_{k+1} = [n, n+1]$.
Step 4. Let $pm_0, pm_1, \ldots, pm_{n+1}$ be the prefix maxima of
        $(N_{\max}[0], N_{\max}[1], \ldots, N_{\max}[n+1])$ and $sm_1, sm_2,$
        $\ldots, sm_n$ be the suffix minima of $(N_{\min}[0],$
        $N_{\min}[1], \ldots, N_{\min}[n+1])$.
Step 5. For each $DP_i = [l_i, r_i]$, $1 \leq i \leq k$, do the following
        substeps.
    Step 5.1. Compute the P-sequence $p_{l_i}, p_{l_i+1}, \ldots,$
              $p_{l_{i+1}-1}$ of $DP_i$.
    Step 5.2. Use Lemma 3.1 to determine if $l_i$ is a cut
              vertex.
    Step 5.3. Use Lemma 3.2 to find pendant edges
              which are incident to $l_i$.
Step 6. For each $DP_i = [l_i, r_i]$, $1 \leq i \leq k$, do the following
        substeps.
```

Step 6.1. Compute the S-sequence $s_{r_{i-1}+1}$, $s_{r_{i-1}+2}$, ..., s_{r_i} of DP_i.

Step 6.2. Use Lemma 3.3 to determine if r_i is a cut vertex.

Step 6.3. Use Lemma 3.4 to find pendant edges which are incident to r_i.

End of Algorithm

Figure 2 uses the example of Figure 1 to illustrate Algorithm A. We take $DP_1 = [1, 5]$ as an instance for explaining how to determine if l_1 is a cut vertex and how to find pendant edges which are incident to l_1. The P-sequence p_1, p_2, p_3 of DP_1 is 0, 2, 6 which is the prefix maxima of (pm_0, $N_{\max}[2]$, $N_{\max}[3]$). Vertex 1 is a cut vertex since $p_2 < 3$. Moreover, edge (1, 2) is a pendant edge since $p_1 < 2$ and $p_2 < 3$. The cut vertices in our example are vertices 1, 4, 8, and 9, while the pendant edges are (1, 2) and (9, 10).

i	0	1	2	3	4	5	6	7	8	9	10	11
$N_{\max}[i]$	0	5	2	6	8	6	6	8	9	10	10	11
pm_i	0	5	5	6	8	8	8	8	9	10	10	11
p_i	0	0	2	6	6	6	6	8	8	9	10	11

(a)

i	0	1	2	3	4	5	6	7	8	9	10	11
$N_{\min}[i]$	0	1	1	1	4	1	3	4	4	8	9	11
sm_i	0	1	1	1	1	1	3	4	4	8	9	11
s_i	0	1	1	1	3	3	3	4	8	9	11	11

(b)

Figure 2. (a) Finding cut vertices and pendant edges by left vertices.
(b) Finding cut vertices and pendant edges by right vertices.

Theorem 3.7. *Algorithm A finds all cut vertices and pendant edges of G in O(n) time.*

Proof. The correctness of Algorithm A follows the above lemmas. Since Steps 1 to 4 can be done in $O(n)$ time and since each DP_i, $1 \le i \le k$, only uses its P-sequence and S-sequence to do Steps 5 and 6, finding all cut vertices and pendant edges takes $O(n)$ time totally. □

4. Finding bridges

An edge (u, v) is a bridge of G if $G - (u, v)$ is disconnected. Pendant edges are certainly bridges, and they can be found by Algorithm A. If bridge (u, v) is not a pendant edge, then both u and v must be cut vertices. However, we cannot conclude that any edge incident to both cut vertices is a bridge. The reason is that there may exist a cycle of G containing this edge. Denote C_n a cycle of n vertices without any chords. By the definition of trapezoid graphs, G can contain only C_3 and C_4 (see Figure 3). Note that any edge incident to both cut vertices is an edge of the dominating path.

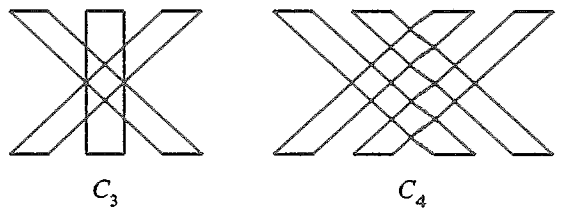


Figure 3. The presentation of C_3 and C_4 in the trapezoid diagram.

Let $DP_1 = [l_1, r_1]$, $DP_2 = [l_2, r_2]$, ..., $DP_k = [l_k, r_k]$ be the dominating pairs of G and $H = l_1\text{–}r_1\text{–}l_2\text{–}r_2\text{–}\ldots\text{–}l_k\text{–}r_k$ the dominating path, where $l_1 = 1$ and $r_k = n$. Then, (l_i, r_i), $1 \le i \le k$, is certainly an edge of H, and (r_i, l_{i+1}) is an edge of H if $r_i > l_{i+1}$, $1 \le i \le k-1$. Lemmas 4.1 and 4.2 will find bridges by using the dominating pairs.

Lemma 4.1. *Let (r_i, l_{i+1}), $1 \le i \le k-1$, be an edge of H with cut vertices r_i and l_{i+1}. Then, (r_i, l_{i+1}) is a bridge if both of the following conditions hold:*
(i) $N_{\max}[r_i] - N_{\min}[l_{i+1}] = 1$;
(ii) $N_{\max}[j] \le r_i$ for all j, $l_i < j < l_{i+1}$.

Proof. Since (r_i, l_{i+1}) is an edge, $r_i > l_{i+1}$. Consider the following two cases.

Case 1. r_i and l_{i+1} are contained in a C_3. Then there must exist a vertex j, $N_{\min}[l_{i+1}] < j < N_{\max}[r_i]$, such that j is adjacent to r_i and l_{i+1}.

Case 2. r_i and l_{i+1} are contained in a C_4. Then, there must exist two adjacent vertices u and v such that each of them is adjacent to different vertex of r_i and l_{i+1}. Without loss of generality, assume $u < v$. There are four subcases that should be considered.

Case 2.1. $u < l_{i+1} < v < r_i$. Then, path $l_1\text{–}r_1\text{–}l_2\text{–}r_2\text{–}\ldots\text{–}l_i\text{–}v\text{–}l_{i+1}\text{–}r_{i+1}\text{–}\ldots\text{–}l_k\text{–}r_k$ is also a dominating path. It contradicts that r_i is a cut vertex.

Case 2.2. $l_{i+1} < u < v < r_i$. Then, path $l_1\text{–}r_1\text{–}l_2\text{–}r_2\text{–}\ldots\text{–}l_i\text{–}v\text{–}u\text{–}r_{i+1}\text{–}\ldots\text{–}l_k\text{–}r_k$ is also a dominating path which contradicts that r_i and l_{i+1} are cut vertices.

Case 2.3. $l_{i+1} < u < r_i < v$. Then, path $l_1\text{–}r_1\text{–}l_2\text{–}r_2\text{–}\ldots\text{–}l_i\text{–}r_i\text{–}u\text{–}r_{i+1}\text{–}\ldots\text{–}l_k\text{–}r_k$ is also a dominating path. It contradicts that l_{i+1} is a cut vertex.

Case 2.4. $u < l_{i+1} < r_i < v$. In this case, both r_i and l_{i+1} can be still cut vertices if $l_i < u$ and $v < r_{i+1}$. Since u is adjacent to v, $N_{\max}[u] \ge v > r_i$.

In condition (i), since $N_{\max}[r_i] - N_{\min}[l_{i+1}] = 1$, there exists no vertex j with $N_{\min}[l_{i+1}] < j < N_{\max}[r_i]$. This implies that vertices r_i and l_{i+1} are not contained in any C_3. In condition (ii), since $N_{\max}[j] \le r_i$ for all j, $l_i < j < l_{i+1}$, Case 2.4 does not take place; vertices r_i and l_{i+1} are not contained in any C_4. Thus, (r_i, l_{i+1}) is a bridge of G if both of the two conditions hold. □

Lemma 4.2. *Let (l_i, r_i), $1 \le i \le k$, be an edge of H with cut vertices l_i and r_i. Then, (l_i, r_i) is a bridge if none of the following conditions holds:*
(i) there exists some vertex j, $r_{i-1} < j < l_{i+1}$, such that j is adjacent to both l_i and r_i;
(ii) there exist some vertex j and $j+1$, $r_{i-1} < j < l_{i+1}-1$, such that j is adjacent to r_i and $j+1$ is adjacent to l_i;
(iii) there exists some vertex j, $r_{i-1} < j < l_{i+1}$, such that j is adjacent to l_i and $N_{\max}[j]$ is adjacent to r_i.
Proof. Observe the following two cases where l_i and r_i are contained in a C_3 and in a C_4.
Case 1. l_i and r_i are contained in a C_3. Then, there exists a vertex j which is adjacent to l_i and r_i. Two subcases should be considered.
Case 1.1. $j < r_{i-1}$. Then, path l_1–r_1–l_2–r_2–...–l_{i-1}–r_{i-1}–j–r_i–...–l_k–r_k is also a dominating path. A contradiction.
Case 1.2. $l_{i+1} < j$. Then, path l_1–r_1–l_2–r_2–...–l_i–j–l_{i+1}–r_{i+1}–...–l_k–r_k is also a dominating path. A contradiction.
Thus, if l_i and r_i are contained in a C_3, then condition (i) holds.
Case 2. l_i and r_i are contained in a C_4. Let u, v be two adjacent vertices such that u, v, l_i, and r_i form a C_4. Without loss of generality, assume $u < v$. Then, $r_{i-1} < u < v < l_{i+1}$; otherwise, l_i and r_i cannot be cut vertices. Since we assume l_i and r_i are contained in a C_4, each vertex j, $r_{i-1} < j < l_{i+1}$, is adjacent to either l_i or r_i.
Case 2.1. u is adjacent to r_i and v is adjacent to l_i. If $v = u+1$, then condition (ii) holds. Suppose $v \ne u+1$. If vertex $u+1$ is adjacent to l_i, then let vertex v be vertex $u+1$ and condition (ii) holds; otherwise, let vertex u be vertex $u+1$. Similarly, if vertex $v-1$ is adjacent to r_i, then let vertex u be vertex $v-1$ and condition (ii) holds; otherwise, let vertex v be vertex $v-1$. With the argument, we can deduce $v = u+1$. Condition (ii) holds.
Case 2.2. u is adjacent to l_i and v is adjacent to r_i. Since u is adjacent to v, $v \le N_{\max}[u]$. Moreover, $N_{\max}[u]$ must be less than l_{i+1}; otherwise, r_i is not a cut vertex. If $N_{\max}[u]$ is adjacent to l_i, then v and $N_{\max}[u]$ are in Case 2.1. Thus, $N_{\max}[u]$ is supposed to be adjacent to r_i, and condition (iii) holds.

Therefore, if l_i and r_i are contained in a C_4, then one of conditions (ii) and (iii) holds.
By the above observation, we can conclude that if none of the three conditions hold, (l_i, r_i) is a bridge of G. □

The following is the algorithm for finding all bridges of a trapezoid graph G.

Algorithm B (Finding all bridges)
Input: (1) $N_{\max}[i]$ and $N_{\min}[i]$ of each vertex i, $1 \le i \le n$.
(2) The dominating pairs $DP_1 = [l_1, r_1]$, $DP_2 = [l_2, r_2]$, ..., $DP_k = [l_k, l_k]$ of G.
(3) Cut vertices of G.
(4) Pendent edges of G.
Output: All bridges of G.
Method:
Step 1. Let each pendant edge be a bridge of G.
Step 2. For each (r_i, l_{i+1}), $1 \le i \le k-1$, if $r_i > l_{i+1}$ and both r_i and l_{i+1} are cut vertices, then use Lemma 4.1 to determine if (r_i, l_{i+1}) is a bridge.
Step 3. For each (l_i, r_i), $1 \le i \le k$, if both l_i and r_i are cut vertices, then use Lemma 4.2 to determine if (l_i, r_i) is a bridge.
End of Algorithm

We use the example of Figure 1 again to illustrate Algorithm B. The dominating pairs in this example are $DP_1 = [1, 5]$, $DP_2 = [4, 8]$, $DP_3 = [8, 9]$, and $DP_4 = [9, 10]$, in which vertices 1, 4, 8, 9 are cut vertices. Since (1, 2) and (9, 10) are pendant edges, (1, 2) and (9, 10) are bridges. In Step 2, only (5, 4) is the edge of (r_i, l_{i+1}). Since vertex 5 is not a cut vertex, (5, 4) is not a bridge. In Step 3, edges (1, 5), (4, 8), (8, 9), (9, 10) are the edges of (l_i, r_i), in which (4, 8) and (8, 9) are incident to both cut vertices. Edge (4, 8) is not a bridge since vertex 7 is adjacent to vertices 4 and 8. Edge (8, 9) is a bridge since none of the three conditions of Lemma 4.2 holds. Thus, bridges in the example are (1, 2), (8, 9), and (9, 10).

Theorem 4.3. *Algorithm B finds all bridges of G in O(n) time.*
Proof. The correctness of Algorithm B follows the definition of pendant edges and Lemmas 4.1 and 4.2. In Step 2, since each edge (r_i, l_{i+1}), $1 \le i \le k-1$, only checks vertices j for $l_i < j < l_{i+1}$, Step 2 takes $O(n)$ time totally. Similarly, in Step 3, since each edge (l_i, r_i), $1 \le i \le k-1$, only checks vertices j for $r_{i-1} < j < l_{i+1}$, Step 3 takes $O(n)$ time totally. Thus, the complexity of Algorithm B is $O(n)$. □

5. The Parallelization of Algorithms A and B

In this section, we will parallelize Algorithms A and B so that they can be done in $O(\log n)$ time using $O(n / \log n)$ processors on the EREW PRAM model.

At first, we need to find the maximum neighbor and the minimum neighbor for each vertex of G. By the prefix maxima computation [5, 7], they can be found in $O(\log n)$ time using $O(n / \log n)$ processors [3].

Let $[i, N_{\max}[i]]$ be an adjacent pair of G for each vertex i, $1 \le i \le n$. We compute the prefix maxima $P = (p_1, p_2, \ldots, p_n)$ of $(N_{\max}[1], N_{\max}[2], \ldots, N_{\max}[n])$ in parallel. Then, let $[1, N_{\max}[1]]$ be a dominating pair of G, and for $2 \le i \le n$, if $p_{i-1} < p_i$, let $[i, N_{\max}[i]]$ be a dominating pair of G. In the example of Figure 1 (a), the dominating pairs are $DP_1 = [1, 5]$, $DP_2 = [3, 6]$, $DP_3 = [4, 8]$, $DP_4 = [8, 9]$, and $DP_5 = [9, 10]$. We can use the prefix maxima computation to find such pairs in parallel.

In the parallelization of Algorithm A, Step 5 needs to use the prefix maxima computation. In Step 5.3, we let $x_j = p_{j-1}$ and $y_j = j+1$ for each vertex j, $l_i < j < l_{i+1}$, then determine if $x_j < j$ and $p_j < y_j$ to avoid concurrent reading. Similarly, we can use the suffix minima computation to parallelize Step 6. Thus, Algorithm A takes $O(\log n)$ time and $O(n / \log n)$ processors.

For parallelizing Algorithm B, Steps 2 and 3 need some additional variables to avoid concurrent reading. In Step 2, we let $x_j = r_i$, for each vertex j, $l_i < j < l_{i+1}$, then determine if $N_{\max}[j] > x_j$ for condition (ii) of Lemma 4.1. In Step 3, let $s_j = l_i$ and $t_j = r_i$ for each vertex j, $r_{i-1} < j < l_{i+1}$. Then, let $x_j = 1$ if j is adjacent to s_j and let $x_j = 0$ otherwise. Similarly, let $y_j = 1$ if j is adjacent to t_j and let $y_j = 0$ otherwise. Moreover, let $w_j = 1$ if $N_{\max}[j]$ is adjacent to t_j and let $w_j = 0$ otherwise. Finally, let $z_j = x_{j+1}$ for each vertex j, $r_{i-1} < j < l_{i+1}-1$. Thus, the three conditions of Lemma 4.2 can be stated by the following.

(i) there exists some vertex j, $r_{i-1} < j < l_{i+1}$, such that $x_j = 1$ and $y_j = 1$;

(ii) there exists some vertex j, $r_{i-1} < j < l_{i+1}-1$, such that $y_j = 1$ and $z_j = 1$;

(iii) there exists some vertex j, $r_{i-1} < j < l_{i+1}$, such that $x_j = 1$ and $w_j = 1$.

Since each step of Algorithm B can be done in $O(\log n)$ time using $O(n / \log n)$ processors, Algorithm B takes $O(\log n)$ time and $O(n / \log n)$ processors.

6. Conclusion

In this paper, we present algorithms for finding cut vertices and bridges on trapezoid graphs in $O(n)$ time and in $O(\log n)$ time using $O(n / \log n)$ processors on the EREW PRAM model. Since the lower bound of find cut vertices and bridges on trapezoid graphs is $\Omega(n)$, our algorithms are optimal.

References

[1] K. Arvind, V. Kamakoti, and C. P. Rangan, Efficient Parallel Algorithms for Permutation Graphs, *Journal of Parallel and Distributed Computing*, Vol. 26, 1995, pp. 116-124.

[2] H. C. Chen and Y. L. Wang, A Linear Time Algorithm for Finding Depth-First Spanning Trees on Trapezoid Graphs, *Information Processing Letters*, to appear.

[3] H. C. Chen and Y. L. Wang, An Efficient Parallel Algorithm for Constructing Spanning Trees on Trapezoid Graphs, *ICS 1996 Proceedings of International Conference on Algorithms*, pp. 15-18.

[4] I. Dagan, M. C. Golumbic, and R. Y. Pinter, Trapezoid Graphs and Their Coloring, *Discrete Applied Mathematics*, Vol. 21, 1988, pp. 35-46.

[5] J. JáJá, *Introduction to Parallel Algorithms*, Addison-Wesley,1992.

[6] T. W. Kao and S. J. Horng, Optimal Algorithms for Computing Articulation Points and Some Related Problems on a Circular-Arc Graph, *Parallel Computing*, Vol. 21, 1995, pp. 953-969.

[7] C. P. Kruskal, L. Rudolph, and M. Snir, The Power of Parallel Prefix, *IEEE Transactions on Computers*, Vol. 34, 1985, pp. 965-968.

[8] Y. D. Liang, Dominations in Trapezoid Graphs, *Information Processing Letters*, Vol. 52, 1994, pp. 309-315.

[9] Y. D. Liang, Steiner Set and Connected Domination in Trapezoid Graphs, *Information Processing Letters*, Vol. 56, 1995, pp. 101-108.

[10] T. H. Ma and J. P. Spinrad, On the 2-Chain Subgraph Cover and Related Problems, *Journal of Algorithms*, Vol. 17, 1994, pp. 251-268.

[11] A. P. Sprague and K. H. Kulkarni, Optimal Parallel Algorithms for Finding Cut Vertices and Bridges of Interval Graphs, Information Processing Letters, Vol. 42, 1992, pp. 229-234.

[12] R. E. Tarjan and U. Vishkin, An Efficient Parallel Biconnectivity Algorithm, *SIAM Journal on Computing*, Vol. 14, 1985, pp. 862-874.

[13] Y. H. Tsin and F. Y. Chen, Efficient Parallel Algorithms for a Class of Graph Theoretic Problems, *SIAM Journal on Computing*, Vol. 13, 1984, pp. 580-599.