Discrete Mathematics and Theoretical Computer Science **6**, 2004, 483–496

Optimal Sequential and Parallel Algorithms for Cut Vertices and Bridges on Trapezoid Graphs[†]

Hon-Chan Chen[1‡]

[1]*Department of Information Management, National Chin-Yi Institute of Technology, Taichung, Taiwan*

received Aug, 13 2004, *accepted Nov, 16 2004*.

Let G be a graph. A component of G is a maximal connected subgraph in G. A vertex v is a cut vertex of G if $\kappa(G-v) > \kappa(G)$, where $\kappa(G)$ is the number of components in G. Similarly, an edge e is a bridge of G if $\kappa(G-e) > \kappa(G)$. In this paper, we will propose new $O(n)$ algorithms for finding cut vertices and bridges of a trapezoid graph, assuming the trapezoid diagram is given. Our algorithms can be easily parallelized on the EREW PRAM computational model so that cut vertices and bridges can be found in $O(\log n)$ time by using $O(\frac{n}{\log n})$ processors.

Keywords: 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, then H is a *component* of G. Let $G-v$, $v \in V$, be the graph obtained by deleting v and all edges incident to v from 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 in G. Similarly, let $G-e$, $e \in E$, be the graph obtained by deleting e from G, 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] and independently by Corneil et al. [3]. It is a superclass of interval graphs and permutation graphs. A trapezoid graph G can be represented by a trapezoid diagram. In the diagram, a trapezoid i is defined by four corner points a_i, b_i, c_i, and d_i, which stand for the upper left, the upper right, the lower left, and the lower right corner of i, respectively. We assume that trapezoids are labeled in increasing order of their b corner points. Each trapezoid in the diagram corresponds to a distinct vertex of G, and (i,j) is an edge of G if and only if trapezoid i intersects trapezoid j in the diagram. Figure 1 shows an example of a trapezoid graph.

[†]This work was supported by National Science Council, Republic of China, under contract NSC89-2218-E-022-002.

[‡]All correspondence should be addressed to Dr. Hon-Chan Chen, Department of Information Management, National Chin-Yi Institute of Technology, 35, Lane 215, Section 1, Chung-Shan Road, Taiping City, Taichung County, Taiwan 411, R.O.C. (E-mail: chenhc@chinyi.ncit.edu.tw)

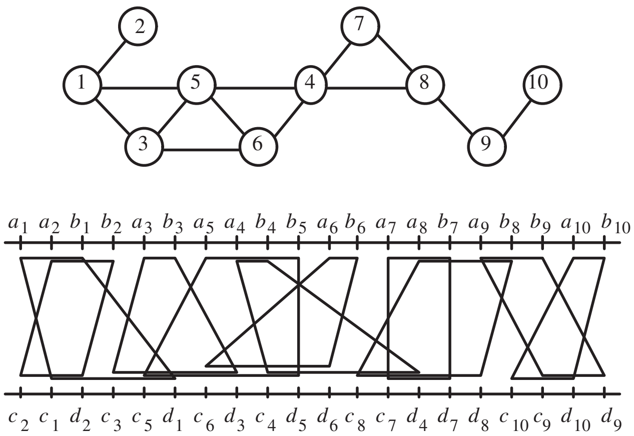

Fig. 1: A trapezoid graph and the corresponding diagram.

A trapezoid graph can be recognized in $O(n^2)$ time by Ma and Spinrad's algorithm [8]. Applying their algorithm, the corresponding trapezoid diagram can be constructed. It is easy to see that for any three vertices i, j, and k of a trapezoid graph, where $i < j < k$, if i is adjacent to k, then j is adjacent to i or k. Generally, the input for a trapezoid graph is $O(n)$, according to the number of vertices. The number of edges could be more, but it is generally not an issue for algorithms in this class of graphs.

The problem of finding cut vertices and bridges on graphs is well-known. It can be solved sequentially in $O(n+m)$ time by a depth-first search algorithm [10], where n is the number of vertices and m the number of edges. Parallel algorithms for this problem on general graphs can be found in [11, 12]. If we restrict ourselves to special types of graphs, the complexity of finding cut vertices and bridges can be reduced. For example, in [9], Sprague et al. presented optimal parallel algorithms for this problem on interval graphs. The complexity of their algorithms is $O(\log n)$ time using $O(\frac{n}{\log n})$ processors on the EREW PRAM computational model if the endpoints of intervals are sorted. Optimal parallel algorithms for finding cut vertices and bridges on permutation graphs were proposed by Arvind et al. [1], which also take $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM model.

An $O(n)$ time algorithm for the depth-first search on trapezoid graphs was proposed by Chen et al. [2], but they did not describe the way to find cut vertices. Recently, Hota et al. have presented optimal sequential and parallel algorithms to compute all cut vertices on trapezoid graphs [5]. The complexities of their algorithms are $O(n)$ time for sequential computation and $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM model for parallel computation. However, their algorithms are not simple because they recognize a cut vertex by many types of connectivity. Moreover, they did not address how to find bridges by cut vertices. In this paper, we propose a simpler $O(n)$ algorithm for finding cut vertices and the first $O(n)$ algorithm for finding bridges on a trapezoid graph, assuming the trapezoid diagram is given. Our algorithms can be easily parallelized on the EREW PRAM model so that cut vertices and bridges can be found in $O(\log n)$ time by using $O(\frac{n}{\log n})$ processors. The algorithms on this paper use the properties of vertex adjacency instead of the corner points. These properties are new and interesting.

The rest of this paper is organized as follows. In Section 2, we introduce an easy way to find a dominating path of a trapezoid graph. In Sections 3 and 4, the algorithms of finding cut vertices and bridges are presented respectively. The parallelization of our algorithms is shown in Section 5. Finally, in Section 6, we give the concluding remarks of this paper.

2 Preliminaries

Let G be a connected trapezoid graph of n vertices which are labeled by $1,2,\cdots,n$ in increasing order of their b corner points. A path P of G is a *dominating path* if every vertex v of G, $v \notin P$, is adjacent to at least one vertex of P. It is obvious that any vertex $v \notin P$ is not a cut vertex since graph $G-v$ remains connected. There may be many different dominating paths in G, and the minimum dominating path can be found in $O(n)$ time by Köhler's algorithm [7]. However, for a trapezoid graph, a path connecting vertices 1 and n is also a dominating path. We call such a path a $(1,n)$*-path*. The following describes an easy way to find a $(1,n)$-path.

Let v be a vertex of G. The *maximal neighbor* (*minimal neighbor*, respectively) of v, denoted by $N_{\max}[v]$ ($N_{\min}[v]$, respectively), is the vertex in $N[v]$ with the maximal label (minimal label, respectively), where $N[v]$ is the set of vertices adjacent to v and v itself. To determine $N_{\max}[v]$, we need to scan a and c corner points from left to right respectively on the trapezoid diagram. Let x_v (y_v, respectively) be the maximal label of a (c, respectively) corner point left to b_v (d_v, respectively), then $N_{\max}[v] = \max\{x_v, y_v\}$. Similarly, we can determine $N_{\min}[v]$ by scanning b and d corner points from right to left respectively on the trapezoid diagram then choosing the minimal label among those whose b or d corner points are right to a_v or c_v. For example, in Figure 1, the maximal label of a corner point left to b_5 is 5, and the maximal label of c corner point left to d_5 is 6. Then, $N_{\max}[5] = \max\{5,6\} = 6$. Moreover, $N_{\min}[5] = \min\{4,1\} = 1$ since the minimal label of b corner point right to a_5 is 4 and the minimal label of d corner point right to c_5 is 1. Trivially, all maximal and minimal neighbors can be determined in $O(n)$ time if we record each latest maximal label and minimal label respectively during our scanning.

For any vertex v of G, the *superior pair* of v, denoted by A_v, is the pair $[i, N_{\max}[i]]$ such that $i = \max\{j | N_{\max}[j] = \max\{N_{\max}[1], N_{\max}[2], \cdots, N_{\max}[v]\}, 1 \leq j \leq v\}$. A $(1,n)$-path can be constructed by some or all superior pairs, and a superior pair on the $(1,n)$-path is called a *dominating pace* (*DP*). Let $[l_i, r_i]$ be a dominating pace DP_i. The dominating paces of G can be determined by the following manner:

Step 1. Let $DP_1 = [l_1, r_1] = A_1$ and let $i = 1$.

Step 2. While $r_i \neq n$, do loop: let $i = i+1$ and $DP_i = [l_i, r_i] = A_{r_{i-1}}$.

In the example of Figure 1, the dominating paces are $DP_1 = A_1 = [1,5]$, $DP_2 = A_5 = [4,8]$, $DP_3 = A_8 = [8,9]$, and $DP_4 = A_9 = [9,10]$ as shown in Table 1.

Suppose there are k dominating paces in G. With vertices $l_1, r_1, l_2, r_2, \cdots, l_k, r_k$, we can construct a $(1,n)$-path, denoted by l_1-r_1-l_2-r_2-…-l_k-r_k. Note that if $r_i = l_{i+1}$, $1 \leq i \leq k-1$, we omit one of r_i and l_{i+1}. The correctness of our method to find a $(1,n)$-path can be proved as follows.

Lemma 1 *Let $[l_i, r_i]$ and $[l_{i+1}, r_{i+1}]$, $1 \leq i \leq k-1$, be two consecutive dominating paces. If $r_i \neq l_{i+1}$, then r_i must be adjacent to l_{i+1}.*

Proof. Assume to the contrary that $r_i \neq l_{i+1}$ and r_i is not adjacent to l_{i+1}. Since $r_i \neq l_{i+1}$, we obtain $l_{i+1} < r_i$ according to Step 2. Moreover, since $l_{i+1} < r_i < r_{i+1}$ and r_i is not adjacent to l_{i+1}, vertices r_i and

Tab. 1: The process of determining the dominating paces.

vertex i	1	2	3	4	5	6	7	8	9	10
$N_{\max}[i]$	5	2	6	8	6	6	8	9	10	10
$N_{\min}[i]$	1	1	1	4	1	3	4	4	8	9
$[i, N_{\max}[i]]$	1,5	2,2	3,6	4,8	5,6	6,6	7,8	8,9	9,10	10,10
A_i	1,5	1,5	3,6	4,8	4,8	4,8	7,8	8,9	9,10	10,10
dominating paces	1,5				4,8			8,9	9,10	

r_{i+1} are adjacent. By the definition of a superior pair, $[r_i, r_{i+1}]$ is a superior pair and is the next dominating pace of $[l_i, r_i]$. It is a contradiction. □

Lemma 2 *A $(1,n)$-path can be constructed by vertices $l_1, r_1, l_2, r_2, \ldots, l_k, r_k$.*

Proof. It is clear that $l_1 = 1$ and $r_k = n$ by Steps 1 and 2. By definitions, we have $l_i < l_{i+1}$ and $r_i < r_{i+1}$ and l_i is adjacent to r_i.

Case 1. $r_i = l_{i+1}$ for all i, $1 \le i \le k-1$. Then, l_1-r_1-l_2-r_2-…-l_k-r_k is the same as l_1-l_2-…-l_k-r_k, which is trivially a $(1,n)$-path.

Case 2. $r_i \neq l_{i+1}$ for some i, $1 \le i \le k-1$. By Lemma 1, r_i is adjacent to l_{i+1}. Since $l_i < l_{i+1}$ and $r_i < r_{i+1}$, there is no cycle in l_1-r_1-l_2-r_2-…-l_k-r_k, and it is a $(1,n)$-path.

□

The $(1,n)$-path in our example is 1-5-4-8-9-10. It is easy to see that the $(1,n)$-path can be found in $O(n)$ time. We will use the dominating paces to find cut vertices and bridges in the following sections. For the sake of computational boundary, we add two dummy dominating paces $DP_0 = [0,1]$ and $DP_{k+1} = [n, n+1]$.

3 Finding cut vertices

Cut vertices of a trapezoid graph are in the vertices of the $(1,n)$-path. However, not all vertices of the $(1,n)$-path are cut vertices. We need to examine each vertex of the $(1,n)$-path.

Let array $S = (s_1, s_2, \cdots, s_n)$ be the *prefix maxima* of all maximal neighbors; i.e. $s_i = \max\{N_{\max}[1], N_{\max}[2], \cdots, N_{\max}[i]\}$ for $i = 1, 2, \cdots, n$. Similarly, let array $T = (t_1, t_2, \cdots, t_n)$ be the *suffix minima* of all minimal neighbors; i.e. $t_i = \min\{N_{\min}[i], N_{\min}[i+1], \cdots, N_{\min}[n]\}$ for $i = 1, 2, \cdots, n$. From Table 1, we obtain arrays $S = (5,5,6,8,8,8,8,9,10,10)$ and $T = (1,1,1,1,1,3,4,4,8,9)$.

Observation 3 *A trapezoid graph is connected if and only if*

(i) $s_i > i$ for all i, $i = 1, 2, \cdots, n-1$, and

(ii) $t_i < i$ for all i, $i = 2, 3, \cdots, n$.

Observation 4 *A trapezoid graph is disconnected if and only if*

(i) $s_i = i$ for some i, $1 \le i \le n-1$, and

(ii) $t_i = i$ *for some* i, $2 \le i \le n$.

Assume trapezoid graph G is connected. Let $S_v = (s_1^{'}, s_2^{'}, \cdots, s_n^{'})$ be the prefix maxima of all $N_{\max}^{'}[i], i = 1,2,\cdots,n$, where $N_{\max}^{'}[i] = i$ if $i = v$ and $N_{\max}^{'}[i] = N_{\max}[i]$ otherwise. Obviously, in S_v, $s_i^{'} = s_i$ for all $i < v$ and $s_v^{'} = s_{v-1}^{'} = s_{v-1}$. For example, $S_4 = (5,5,6,6,6,6,8,9,10,10)$ since $N_{\max}^{'}[4] = 4$. We will use S_v to determine if v is a cut vertex.

Lemma 5 *Let* $[l_i, r_i]$, $2 \le i \le k$, *be a dominating pace of* G. *Then, vertex* l_i *is a cut vertex if and only if there exists some* v, $l_i \le v \le l_{i+1} - 1$, *such that* $s_v^{'} = v$ *in* S_{l_i}.

Proof. Since $N_{\max}^{'}[l_i] = l_i$ in S_{l_i}, vertex l_i is assumed to be adjacent to vertices whose labels are not greater than l_i. We denote the new graph by $G^{'}$. By Observation 3, we know $s_v^{'} > v$ in S_{l_i} for all v, $1 \le v \le l_i - 1$. Moreover, l_{i+1}-r_{i+1}-l_{i+2}-r_{i+2}-…-l_k-n is a path connecting vertices l_{i+1} and n. Vertices from l_{i+1} to n are connected, and $s_v^{'} > v$ in S_{l_i} for all v, $l_{i+1} \le v \le n-1$. Thus, if there exists some v, $l_i \le v \le l_{i+1} - 1$, such that $s_v^{'} = v$ in S_{l_i}, then $G^{'}$ is disconnected. It means that the removal of edges (l_i, x), $l_i < x$, makes G disconnected, and l_i is certainly a cut vertex.

For the other part of the proof, since l_i is a cut vertex, $G - l_i$ is disconnected. Since $l_i \le r_{i-1}$ and 1-r_1-l_2-r_2-…-l_{i-1}-r_{i-1} is a path, vertices from 1 to r_{i-1} are connected, and the removal of any edge (l_i, x), $x < l_i$, still remains these vertices connected. Thus, we only consider the removal of edge (l_i, x), $l_i < x$, and we set $N_{\max}^{'}[l_i] = l_i$. Since l_{i+1}-r_{i+1}-l_{i+2}-r_{i+2}-$\cdots$-l_k-n is a path connecting vertices from l_{i+1} to n, we have $s_v^{'} > v$ in S_{l_i} for all v, $l_{i+1} \le v \le n-1$. Thus, if l_i is a cut vertex, then there exists some v, $l_i \le v \le l_{i+1} - 1$, such that $s_v^{'} = v$ in S_{l_i}. □

Similarly, we can let $T_v = (t_1^{'}, t_2^{'}, \cdots, t_n^{'})$ be the suffix minima of all $N_{\min}^{'}[i], i = 1,2,\cdots,n$, where $N_{\min}^{'}[i] = i$ if $i = v$ and $N_{\min}^{'}[i] = N_{\min}[i]$ otherwise; then determine if v is a cut vertex by T_v.

Lemma 6 *Let* $[l_i, r_i]$, $1 \le i \le k-1$, *be a dominating pace of* G. *Then, vertex* r_i *is a cut vertex if and only if there exists some* v, $r_{i-1} + 1 \le v \le r_i$, *such that* $t_v^{'} = v$ *in* T_{r_i}.

Corollary 7 *Vertex* 1 *is a cut vertex if and only if there exists some* v, $2 \le v \le l_2 - 1$, *such that* $s_v^{'} = v$ *in* S_1. *Moreover, vertex* n *is a cut vertex if and only if there exists some* v, $r_{k-1} + 1 \le v \le n-1$, *such that* $t_v^{'} = v$ *in* T_n.

For any two dominating paces $[l_i, r_i]$ and $[l_j, r_j]$, the examination of l_i and l_j uses mutually independent vertex sets, and so does the examination of r_i and r_j. It is reasonable to design a linear time algorithm for finding all cut vertices. Given $S = (s_1, s_2, \cdots, s_n), T = (t_1, t_2, \cdots, t_n)$, and all dominating paces $[l_i, r_i]$, $1 \le i \le k$, define p_v and q_v, $v = 1,2,\cdots,n$, as follows:

$$p_v = \begin{cases} 1 & \text{if } v = 1; \\ s_{v-1} & \text{if } v \in \{l_2, l_3, \cdots, l_k\}; \\ N_{\max}[v] & \text{otherwise}; \end{cases} \tag{1}$$

and

$$q_v = \begin{cases} n & \text{if } v = n; \\ t_{v+1} & \text{if } v \in \{r_1, r_2, \cdots, r_{k-1}\}; \\ N_{\min}[v] & \text{otherwise}. \end{cases} \tag{2}$$

Let $S^* = (s_1^*, s_2^*, \cdots, s_n^*)$ be the prefix maxima of all p_v and $T^* = (t_1^*, t_2^*, \cdots, t_n^*)$ be the suffix minima of all q_v for $v = 1, 2, \cdots, n$. In our example, $S^* = (1,2,6,6,6,6,8,8,9,10)$ and $T^* = (1,1,1,3,3,3,4,8,9,10)$. For any dominating pace $[l_i, r_i]$ with $S_{l_i} = (s_1^{'}, s_2^{'}, \cdots, s_n^{'})$ and $T_{r_i} = (t_1^{'}, t_2^{'}, \cdots, t_n^{'})$, we can find that $s_v^{'} = s_v^*$, $l_i \leq v \leq l_{i+1} - 1$, and $t_v^{'} = t_v^*$, $r_{i-1} + 1 \leq v \leq r_i$. Thus, all cut vertices can be found directly from S^* and T^*. The following is our algorithm to find cut vertices.

Algorithm A (Finding cut vertices of a trapezoid graph)

Input: *The dominating paces of a trapezoid graph G*

Output: *All cut vertices of G*

Step 1. *Compute* $S = (s_1, s_2, \cdots, s_n)$ *and* $T = (t_1, t_2, \cdots, t_n)$.

Step 2. *Compute* $S^* = (s_1^*, s_2^*, \cdots, s_n^*)$ *and* $T^* = (t_1^*, t_1^*, \cdots, t_n^*)$.

Step 3. *For each dominating pace* $[l_i, r_i]$, $1 \leq i \leq k$, *do case:*

- *Case 1.* $i = 1$. *If there exists some v,* $2 \leq v \leq l_2 - 1$, *such that* $s_v^* = v$, *then vertex 1 is a cut vertex.*
- *Case 2.* $2 \leq i \leq k$. *If there exists some v,* $l_i \leq v \leq l_{i+1} - 1$, *such that* $s_v^* = v$, *then vertex* l_i *is a cut vertex.*

Step 4. *For each dominating pace* $[l_i, r_i]$, $1 \leq i \leq k$, *do case:*

- *Case 1.* $1 \leq i \leq k - 1$. *If there exists some v,* $r_{i-1} + 1 \leq v \leq r_i$, *such that* $t_v^* = v$, *then vertex* r_i *is a cut vertex.*
- *Case 2.* $i = k$. *If there exists some v,* $r_{k-1} + 1 \leq v \leq n - 1$, *such that* $t_v^* = v$, *then vertex n is a cut vertex.*

We use dominating pace $[1,5]$ of our example to explain Steps 3 and 4. Vertex 1 is a cut vertex since $s_2^* = 2$; however, vertex 5 is not a cut vertex since there exists no vertex v, $2 \leq v \leq 5$, such that $t_v^* = v$. The cut vertices in our example are vertices 1, 4, 8, and 9.

Theorem 8 *Algorithm A finds all cut vertices of a trapezoid graph in* $O(n)$ *time.*

Proof. Since cut vertices are in $\{l_1, r_1, l_2, r_2, \cdots, l_k, r_k\}$ and Algorithm A examines l_i and r_i for each dominating pace $[l_i, r_i]$, $1 \leq i \leq k$, all cut vertices can be found by Algorithm A. By the computation of prefix maxima and suffix minima, Steps 1 and 2 can be done in $O(n)$ time. Steps 3 and 4 can also be done in $O(n)$ time totally. The complexity of Algorithm A is therefore $O(n)$. □

4 Finding bridges

An edge e is a bridge of G if $G - e$ is disconnected. A bridge must be incident to one or two cut vertices. However, we cannot conclude that an edge incident to any cut vertices is a bridge. In this section, we will find bridges by cut vertices. We use the same notation as in the previous section.

A vertex is *isolated* if it is not adjacent to any other vertices in the graph. From Observation 4, we can make the following observation.

Observation 9 *Let i be a vertex of a disconnected trapezoid graph. Then,*

(i) vertex i, $2 \leq i \leq n-1$, is isolated if and only if $s_{i-1} = i-1$ and $s_i = i$ (or if $t_i = i$ and $t_{i+1} = i+1$);

(ii) vertex 1 is isolated if and only if $s_1 = 1$ (or $t_2 = 2$);

(iii) vertex n is isolated if and only if $s_{n-1} = n-1$ (or $t_n = n$).

An edge is a *pendant edge* if its removal makes one vertex isolated. A pendant edge is certainly a bridge, which is incident to one cut vertex. Lemmas 10 and 11 will show how to use dominating paces, S^*, and T^* to determine pendant edges. Their correctness is directly from the above observation.

Lemma 10 *Suppose l_i, $1 \leq i \leq k$, is a cut vertex of G. Then, edge (l_i, v), $l_i < v < l_{i+1}$, is a pendant edge if and only if $s^*_{v-1} = v-1$ and $s^*_v = v$. Moreover, edge (l_k, n) is a pendant edge if and only if $s^*_{n-1} = n-1$.*

Lemma 11 *Suppose r_i, $1 \leq i \leq k$, is a cut vertex of G. Then, edge (r_i, v), $r_{i-1} < v < r_i$, is a pendant edge if and only if $t^*_v = v$ and $t^*_{v+1} = v+1$. Moreover, edge $(1, r_1)$ is a pendant edge if and only if $t^*_2 = 2$.*

A pendant edge is a bridge; however, not all bridges are pendant edges. A bridge, not a pendant edge, must be incident to two cut vertices. Let C_m be a cycle of m vertices without any chord. By the definition of trapezoid graphs [4], a trapezoid graph G may contain only C_3 or C_4, and we have the following lemma.

Lemma 12 *An edge e incident to two cut vertices of G is a bridge if and only if it is an edge of the $(1,n)$-path and not contained in any C_3 or C_4.*

Proof. It is trivial that a bridge cannot be contained in a cycle. Let $e = (u, v)$. Since u and v are cut vertices, they are vertices of the $(1,n)$-path. Assume to the contrary that e is not an edge of the $(1,n)$-path, then the path connecting u and v forms a cycle with edge (u, v). It contradicts that e is a bridge. □

To determine if an edge of the $(1,n)$-path is contained in a cycle, we need to know what vertices edge (l_i, r_i) and (r_i, l_{i+1}) may be incident to, respectively.

Lemma 13 *Suppose l_i and r_i, $1 \leq i \leq k$, are two cut vertices of G, and let v be a vertex other than l_i and r_i.*

(i) If v is adjacent to r_i, then $v > r_{i-1}$ when $l_i \neq r_{i-1}$ or $v > l_{i-1}$ when $l_i = r_{i-1}$;

(ii) If v is adjacent to l_i, then $v < l_{i+1}$ no matter $r_i \neq l_{i+1}$ or $r_i = l_{i+1}$.

Proof. Consider the following two cases for condition i:

Case 1. $l_i \neq r_{i-1}$. If $v < r_{i-1}$, then v must be adjacent to some l_j or r_j, $1 \leq j \leq i-1$. Without loss of generality, assume v is adjacent to r_j. Then, l_1-r_1-l_2-r_2-…-l_j-r_j-v-r_i-l_{i+1}-r_{i+1}-…-l_k-r_k is a $(1,n)$-path, and it contradicts that l_i is a cut vertex.

Case 2. $l_i = r_{i-1}$. If $v < l_{i-1}$, then v must be adjacent to l_{i-1} since $v < l_{i-1} < r_{i-1} < r_i$ and v is adjacent to r_i and l_{i-1} is not adjacent to r_i. We can find that l_1-r_1-l_2-r_2-…-l_{i-1}-v-r_i-l_{i+1}-r_{i+1}-…-l_k-r_k is a $(1,n)$-path, and l_i cannot be a cut vertex.

For condition ii, we need to consider the following two cases:

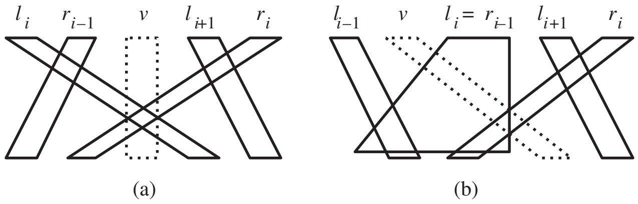

Fig. 2: The illustration of Lemma 15. (a) condition i. (b) condition ii.

Case 1. $r_i \neq l_{i+1}$. If $v > l_{i+1}$, then v must be adjacent to some l_j or r_j, $i+1 \leq j \leq k$, and it will contradict that r_i is a cut vertex.

Case 2. $r_i = l_{i+1}$. Since v is adjacent to l_i, we have $v < N_{\max}[l_i] = r_i = l_{i+1}$.

□

Lemma 14 *Suppose r_i and l_{i+1}, $1 \leq i \leq k-1$, are two distinct cut vertices of G, and let v be a vertex other than r_i and l_{i+1}.*

(i) If v is adjacent to l_{i+1}, then $v > r_{i-1}$;

(ii) If v is adjacent to r_i, then $v < l_{i+2}$.

Proof. For condition (i), if $v < r_{i-1}$, then v must be adjacent to some l_j or r_j, $1 \leq j \leq i-1$, and it will contradict that r_i is a cut vertex. For condition (ii), if $v > l_{i+2}$, then v must be adjacent to some l_j or r_j, $i+2 \leq j \leq k$, and l_{i+1} cannot be a cut vertex. □

The following lemmas show the ways to determine whether an edge incident to both cut vertices is contained in a cycle.

Lemma 15 *Suppose l_i and r_i, $1 \leq i \leq k$, are two cut vertices of G. If edge (l_i, r_i) is contained in a C_3, then one of the following conditions holds:*

(i) When $l_i \neq r_{i-1}$, there exists some vertex v, $r_{i-1} < v < l_{i+1}$, such that v is adjacent to l_i and r_i;

(ii) When $l_i = r_{i-1}$, there exists some vertex v, $l_{i-1} < v < l_{i+1}$, such that v is adjacent to l_i and r_i.

Proof. The correctness is from Lemma 13. (See Figure 2.) □

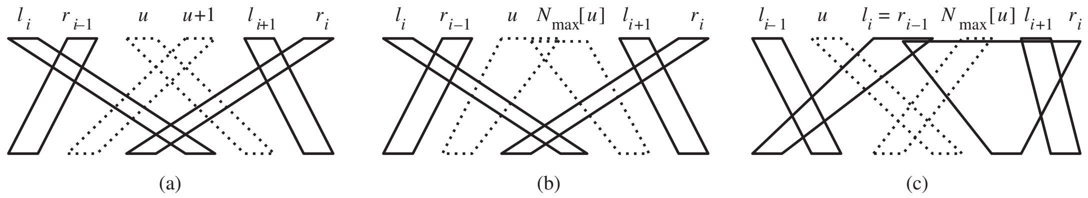

Fig. 3: The illustration of Lemma 16. (a) condition i. (b) condition ii. (c) condition iii.

Lemma 16 *Suppose l_i and r_i, $1 \le i \le k$, are two cut vertices of G. If edge (l_i, r_i) is contained in a C_4, then one of the following conditions holds:*

(i) No matter $l_i \neq r_{i-1}$ or $l_i = r_{i-1}$, there exist two consecutive vertices v and $v+1$, $r_{i-1} < v < l_{i+1} - 1$, such that v ($v+1$, respectively) is adjacent to r_i (l_i, respectively) but not adjacent to l_i (r_i, respectively);

(ii) When $l_i \neq r_{i-1}$, there exist two vertices v and $N_{\max}[v]$, $r_{i-1} < v \le N_{\max}[v] < l_{i+1}$, such that v ($N_{\max}[v]$, respectively) is adjacent to l_i (r_i, respectively) but not adjacent to r_i (l_i, respectively);

(iii) When $l_i = r_{i-1}$, there exist two vertices v and $N_{\max}[v], l_{i-1} < v \le N_{\max}[v] < l_{i+1}$, such that v ($N_{\max}[v]$, respectively) is adjacent to l_i (r_i, respectively) but not adjacent to r_i (l_i, respectively).

Proof. We assume that no one vertex is adjacent to both l_i and r_i to form a C_3. Let u and v be two adjacent vertices such that u, v, l_i, and r_i form a C_4. Without loss of generality, assume $u < v$. Consider the following two cases:

Case 1. u is adjacent to r_i and v is adjacent to l_i. Assume $v > u+1$. If vertex $u+1$ is adjacent to l_i, then u, $u+1$, l_i, and r_i form a C_4; otherwise, $u+1$ is adjacent to r_i and we let u be vertex $u+1$. Similarly, if vertex $v-1$ is adjacent to r_i, then $v-1$, v, l_i, and r_i form a C_4; otherwise, $v-1$ is adjacent to l_i and we let v be vertex $v-1$. With this argument, we can deduce that vertices u, $v = u+1$, l_i, and r_i form a C_4. Since u is adjacent to r_i and v is adjacent to l_i, we obtain $r_{i-1} < u < v < l_{i+1}$ if $l_i \neq r_{i-1}$ and $l_{i-1} < u < v < l_{i+1}$ if $l_i = r_{i-1}$ by Lemma 13. However, in the case of $l_i = r_{i-1}$, if $l_{i-1} < u < l_i = r_{i-1}$, then u must be adjacent to l_{i-1} since l_{i-1} is adjacent to r_{i-1} and u is not adjacent to $l_i = r_{i-1}$. It contradicts that l_i is a cut vertex since l_1-r_1-l_2-r_2-$\ldots$-l_{i-1}-u-r_i-l_{i+1}-r_{i+1}-$\ldots$-l_k-r_k is a $(1,n)$-path. Thus, we have $r_{i-1} < u < v < l_{i+1}$ no matter $l_i \neq r_{i-1}$ or $l_i = r_{i-1}$. (See Figure 3 (a).)

Case 2. u is adjacent to l_i and v is adjacent to r_i. Since u is adjacent to v, we have $v \le N_{\max}[u]$. If $N_{\max}[u]$ is adjacent to l_i, then v, $N_{\max}[u]$, l_i, and r_i form a C_4, which is in Case 1. Thus, $N_{\max}[u]$ is adjacent to r_i, and we can let v be $N_{\max}[u]$. If $N_{\max}[u] > l_{i+1}$, then $N_{\max}[u]$ must be adjacent to some l_j or r_j, $i+1 \le j \le k$. Without loss of generality, assume $N_{\max}[u]$ is adjacent to l_j. Then, l_1-r_1-l_2-r_2-$\ldots$-l_i-u-$N_{\max}[u]$-l_j-r_j-$\ldots$-l_k-r_k is a $(1,n)$-path, and r_i cannot be a cut vertex. Therefore, we have $N_{\max}[u] < l_{i+1}$. Consider the following two subcases:

Subcase 1. $l_i \neq r_{i-1}$. If $u < r_{i-1}$, then u must be adjacent to some l_j or r_j, $1 \leq j \leq i-1$, and it will contradict that l_i is a cut vertex. We obtain $r_{i-1} < u \leq N_{\max}[u] < l_{i+1}$. (See Figure 3 (b).)

Subcase 2. $l_i = r_{i-1}$. Since $N_{\max}[u]$ is adjacent to r_i, we have $l_{i-1} < N_{\max}[u]$ by Lemma 13. Moreover, l_{i-1} cannot be adjacent to $N_{\max}[u]$; otherwise, l_1-r_1-l_2-r_2-…-l_{i-1}-$N_{\max}[u]$-r_i-l_{i+1}-r_{i+1}-…- l_k-r_k is a $(1,n)$-path which contradicts that l_i is a cut vertex. If $u < l_{i-1}$, then u must be adjacent to l_{i-1} since $u < l_{i-1} < N_{\max}[u]$ and l_{i-1} is not adjacent to $N_{\max}[u]$. However, this contradicts that l_i is a cut vertex since l_1-r_1-l_2-r_2-…-l_{i-1}-u-$N_{\max}[u]$-r_i-l_{i+1}-r_{i+1}-…- l_k-r_k is a $(1,n)$-path. Thus, we have $l_{i-1} < u \leq N_{\max}[u] < l_{i+1}$. (See Figure 3 (c).)

□

Lemma 17 *Suppose r_i and l_{i+1}, $1 \leq i \leq k-1$, are two distinct cut vertices of G.*

(i) If edge (r_i, l_{i+1}) is contained in a C_3, then there exists some vertex v, $r_{i-1} < v < l_{i+2}$, such that v is adjacent to r_i and l_{i+1};

(ii) If edge (r_i, l_{i+1}) is contained in a C_4, then there exist two vertices v and $N_{\max}[v]$, where $r_{i-1} < v < l_{i+1}$ and $r_i < N_{\max}[v] < l_{i+2}$, such that v ($N_{\max}[v]$, respectively) is adjacent to r_i (l_{i+1}, respectively) but not adjacent to l_{i+1} (r_i, respectively).

Proof. Condition i is directly from Lemma 14. (See Figure 4 (a).) In condition ii, we assume that no one vertex is adjacent to both r_i and l_{i+1} to form a C_3. Let u and v be two adjacent vertices such that u, v, r_i, and l_{i+1} form a C_4. Without loss of generality, assume $u < v$. Consider the following two cases:

Case 1. u is adjacent to r_i and v is adjacent to l_{i+1}. If $u < r_{i-1}$, then u must be adjacent to some l_j or r_j, $1 \leq j \leq i-1$, and it will contradict that r_i is a cut vertex. Note that $u < l_{i+2}$ by Lemma 14 and u is not adjacent to l_{i+1}. If $u > l_{i+1}$, then u must be adjacent to r_{i+1} since $l_{i+1} < u < l_{i+2} \leq r_{i+1}$ and u is not adjacent to l_{i+1}. This will contradict that l_{i+1} is a cut vertex since l_1-r_1-l_2-r_2-…-l_i-r_i-u-r_{i+1}-…-l_k-r_k is a $(1,n)$-path. Therefore, $r_{i-1} < u < l_{i+1}$. Similarly, we can obtain $r_i < v < l_{i+2}$. Since u is adjacent to v, $v \leq N_{\max}[u]$. It is easy to see that $N_{\max}[u] < l_{i+2}$; otherwise, $N_{\max}[u]$ is adjacent to some l_j or r_j, $l_{i+2} < j < k$, and l_{i+1} cannot be a cut vertex. Thus, we have $r_i < v \leq N_{\max}[u] < l_{i+2}$. It means that if u, v, r_i, and l_{i+1} form a C_4, then u, $N_{\max}[u]$, r_i, and l_{i+1} also form a C_4. (See Figure 4 (b).)

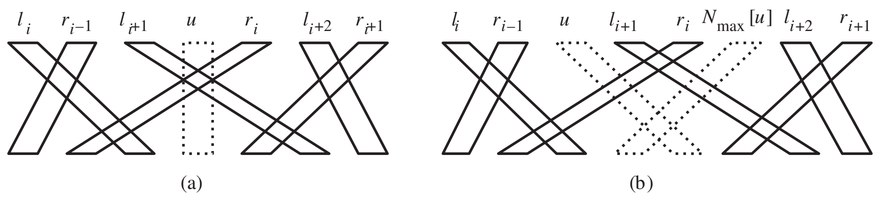

Fig. 4: The illustration of Lemma 17. (a) condition i. (b) condition ii.

Case 2. u is adjacent to l_{i+1} and v is adjacent to r_i. By Lemma 14, $r_{i-1} < u$ and $v < l_{i+2}$. If $r_{i-1} < u < r_i$, then u must be adjacent to l_i since $l_i \leq r_{i-1} < u < r_i$ and u is not adjacent to r_i. We can find that l_1-r_1-l_2-r_2-…-l_i-u-l_{i+1}-r_{i+1}-…-l_k-r_k is a $(1,n)$-path, and it contradicts that r_i is a cut vertex. Thus, $r_i < u$. Similarly, we can obtain $v < l_{i+1}$. However, it contradicts $u < v$ since $v < l_{i+1} < r_i < u$. Case 2 cannot hold.

□

Algorithm B below describes the steps of finding bridges of a trapezoid graph.

Algorithm B (Finding bridges of a trapezoid graph)

Input: *Dominating paces, S^*, T^*, and cut vertices of G.*

Output: *All bridges of G.*

Step 1. *If $s^*_{n-1} = n-1$, then (l_k, n) is a pendant edge. Moreover, if $t^*_2 = 2$, then $(1, r_1)$ is a pendant edge.*

Step 2. *For each dominating pace $[l_i, r_i]$, $1 \leq i \leq k$, if l_i is a cut vertex, then find pendant edges incident to l_i as follows: for each vertex v, $l_i < v < l_{i+1}$, if $s^*_{v-1} = v-1$ and $s^*_v = v$, then (l_i, v) is a bridge.*

Step 3. *For each dominating pace $[l_i, r_i]$, $1 \leq i \leq k$, if r_i is a cut vertex, then find pendant edges incident to r_i as follows: for each vertex v, $r_{i-1} < v < r_i$, if $t^*_v = v$ and $t^*_{v+1} = v+1$, then (r_i, v) is a bridge.*

Step 4. *For i = 1 to k, if both l_i and r_i are cut vertices, then (l_i, r_i) is a bridge if none of the conditions in Lemmas 15 and 16 holds.*

Step 5. *For i = 1 to $k-1$, if both r_i and l_{i+1} are cut vertices, then (r_i, l_{i+1}) is a bridge if none of the conditions in Lemma 17 holds.*

We use our example again to illustrate Algorithm B. The dominating paces are $[1,5]$, $[4,8]$, $[8,9]$, $[9,10]$, and vertices 1, 4, 8, 9 are cut vertices. In dominating pace $[1,5]$, since $s^*_1 = 1$ and $s^*_2 = 2$, edge $(1,2)$ is a pendant edge. In dominating pace $[9,10]$, since $s^*_9 = 9$, edge $(9,10)$ is a pendant edge. In Step 4, $(4,8)$ and $(8,9)$ are the edges of (l_i, r_i) incident to two cut vertices. Edge $(4,8)$ is not a bridge since vertex 7 is adjacent to vertices 4 and 8, while edge $(8,9)$ is a bridge since none of the conditions in Lemmas 15 and 16 holds. In Step 5, 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. All bridges in our example are $(1,2)$, $(8,9)$, and $(9,10)$.

Theorem 18 *Algorithm B finds all bridges of a trapezoid graph in $O(n)$ time.*

Proof. A bridge is an edge incident to one cut vertex (a pendant edge) or two cut vertices (an edge of the $(1,n)$-path). Since cut vertices are in $\{l_1, r_1, l_2, r_2, \cdots, l_k, r_k\}$ and the $(1,n)$-path is composed of (l_i, r_i) and (r_i, l_{i+1}), all bridges can be found by Algorithm B. It is trivial that each step of Algorithm B takes $O(n)$ time, thus the complexity of Algorithm B is $O(n)$. □

5 The parallel algorithms

In this section, we will parallelize algorithms A and B so that the problem of finding cut vertices and bridges of a trapezoid graph G can be solved in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors on the EREW PRAM model.

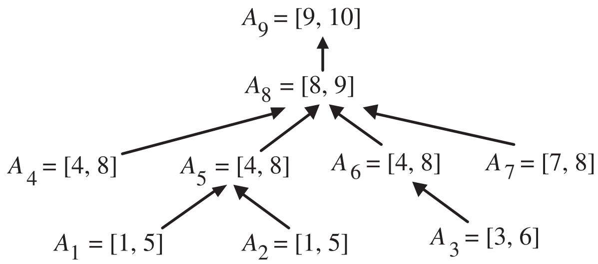

Fig. 5: The rooted tree of superior pairs.

At first, we can use the parallel algorithms of prefix maxima and suffix minima [6] to find the maximal and minimal neighbors for each vertex of G. These parallel algorithms take $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM model.

Secondly, we need to find the dominating paces of G in parallel. Using the parallel computation of prefix maxima on $[l_i, r_i]$, $i = 1, 2, \cdots, n$, we can obtain all superior pairs A_i. For each superior pair $A_i = [p_i, q_i]$, $i = 1, 2, \cdots, n-1$, let A_{q_i} be the successor (or parent) of A_i if $q_i \neq n$. It is trivial that these superior pairs form a rooted tree (or forest) which has $O(n)$ nodes and $O(n)$ edges. The $(1, n)$-path is identical to the chain from A_1 to its root $A_k = [p_k, q_k]$, $1 \leq k \leq n-1$, where $q_k = n$. Figure 5 shows the rooted tree formed by the superior pairs in Table 1. Applying the technique of list ranking [6], all dominating paces can be determined in $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM model.

Also by the parallel algorithms of prefix maxima and suffix minima, arrays S, T, S^*, and T^* of G can be obtained. Steps 3 and 4 of Algorithm A find cut vertices by using l_i and r_i. It is easy to see that there is no concurrent memory access in these two steps, and Algorithm A can be done in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors on the EREW PRAM model.

In Algorithm B, Step 1 can be done in $O(1)$ time with $O(1)$ processor. In Step 2, we can first simultaneously examine if $s^*_{v-1} = v-1$ for all vertices v, $l_i < v < l_{i+1}$, then simultaneously examine if $s^*_v = v$ for the same vertices. Step 3 can use the similar way to examine if $t^*_v = v$ and $t^*_{v+1} = v+1$ for all vertices v, $r_{i-1} < v < r_i$. Since there is no concurrent memory access in these three steps, all pendant edges can be found in $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM model.

To transform Lemmas 15, 16, and 17 into parallel steps, we need some additional variables to avoid concurrent memory access. Lemma 15 has two conditions. In condition (i), we can let $x_v = l_i$ and $y_v = r_i$ for each vertex v, $r_{i-1} < v < l_{i+1}$, then examine if v is adjacent to both x_v and y_v. There is no concurrent read and write in condition (i). Condition (ii) can be implemented by two stages for avoiding concurrent memory access. The first stage uses vertices v, $l_{i-1} < v < r_{i-1}$, while the second stage uses vertices v, $r_{i-1} < v < l_{i+1}$. We examine if v is adjacent to both x_v and y_v for the first stage then for the second stage, where $x_v = l_i$ and $y_v = r_i$. Thus, all dominating paces of $l_i = r_{i-1}$ can be done without any concurrent read and write.

Lemma 16 has three conditions. In condition (i), we can let $x_v = l_i$, $y_v = r_i$, and $z_v = v+1$ for all vertices v, $r_{i-1} < v < l_{i+1} - 1$, then examine if v is adjacent to y_v and z_v is adjacent to x_v. In condition (ii), we can let $x_v = l_i$, $y_v = r_i$, and $z_v = N_{\max}[v]$ for all vertices v, $r_{i-1} < v < l_{i+1}$, then examine if v is adjacent to x_v

and z_v is adjacent to y_v. In condition (iii), we need two stages for parallel computation. The first stage uses vertices v, $l_{i-1} < v < r_{i-1}$, and the second stage uses vertices v, $r_{i-1} < v < l_{i+1}$. Then, we examine if v is adjacent to x_v and z_v is adjacent to y_v for these two stages respectively, where $x_v = l_i$, $y_v = r_i$, and $z_v = N_{\max}[v]$.

Condition (i) of Lemma 17 also needs two stages: vertices v, $r_{i-1} < v < l_{i+1}$, are for the first stage and vertices v, $l_{i+1} < v < l_{i+2}$ are for the second stage. Let $x_v = r_i$ and $y_v = l_{i+1}$, then we examine if v is adjacent to both x_v and y_v for each stage. In condition (ii), we can let $x_v = r_i$, $y_v = l_{i+1}$, and $z_v = N_{\max}[v]$ for all vertices v, $r_{i-1} < v < l_{i+1}$, then examine if v is adjacent to x_v and z_v is adjacent to y_v, where $r_i < N_{\max}[v] < l_{i+2}$. Therefore, Lemmas 15, 16, and 17 can be implemented in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors on the EREW PRAM model. We have the following theorem.

Theorem 19 *All cut vertices and bridges of a trapezoid graph can be found in parallel in $O(\log n)$ time with $O(\frac{n}{\log n})$ processors on the EREW PRAM computational model.*

6 Concluding remarks

In this paper, we use dominating paces $[l_i, r_i]$ to find cut vertices and bridges. This idea is different from other proposed algorithms for trapezoid graphs, which almost use the corner points. The properties of l_i and r_i are new and interesting.

Using the properties of the dominating paces, we presented $O(n)$ time algorithms to find cut vertices and bridges on trapezoid graphs. Our algorithms can be easily parallelized so that cut vertices and bridges can be found in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors on the EREW PRAM computational model. Since the lower bound of complexity for finding 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*, vol. 63, 1997, pp. 13-18.

[3] D. G. Corneil and P. A. Kamula, Extensions of Permutation and Interval Graphs, *Congressus Numerantium*, vol. 58, 1987, pp. 267-275.

[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] M. Hota, M. Pal, and T. K. Pal, Optimal Sequential and Parallel Algorithms to Compute All Cut Vertices on Trapezoid Graphs, *Computational Optimization and Applications*, vol. 27, 2004, pp. 95-113.

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

[7] E. Köhler, Connected Domination and Dominating Clique in Trapezoid Graphs, *Discrete Applied Mathematics*, vol. 99, 2000, pp. 91-110.

[8] 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.

[9] 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.

[10] R. E. Tarjan, Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing*, vol. 1, 1972, pp. 146-160.

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

[12] 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.